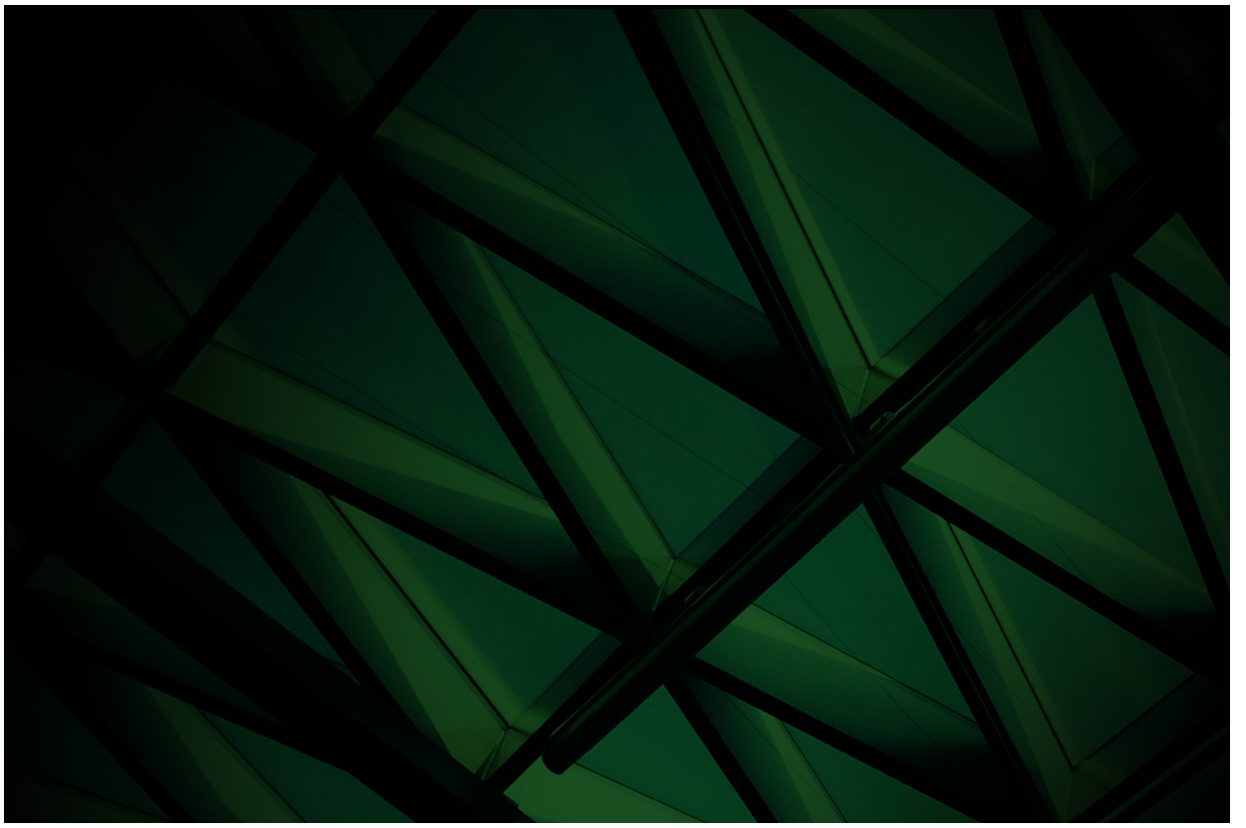


Development Environment User's Guide

VERSION 2022.0.05

jade platform

Copyright©2025 Jade Software Corporation Limited. All rights reserved.

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2025 Jade Software Corporation Limited.

All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

For details about other licensing agreements for third-party products, you must read the Jade **Readme.txt** file.

Contents

Contents ... **iii**

Before You Begin ... **xxi**

Who Should Read this Guide ... xxi
What's Included in this Guide ... xxi
Conventions ... xxii
Related Documentation ... xxii

Chapter 1 Jade Platform Concepts and Terminology ... **24**

Overview ... 24
Jade Object Model ... 25
Types ... 25
Objects and Classes ... 25
Object Lifetimes ... 26
Features ... 26
Properties ... 26
Methods ... 26
Conditions ... 27
Constraints ... 27
Encapsulation ... 27
Inheritance and Polymorphism ... 28
Schemas ... 28
Accessing and Updating Instances of Classes in Other Schemas ... 29
Controlling the Use of Elements in Other Schemas ... 31
final Option ... 31
subschemaFinal Option ... 32
subschemaCopyFinal Option ... 32
subschemaHidden Option ... 33
Class Lifetime Options ... 33
persistentAllowed Class Option ... 33
transientAllowed Class Option ... 34
sharedTransientAllowed Class Option ... 34
subclassPersistentAllowed Class Option ... 35
subclassTransientAllowed Class Option ... 35
subclassSharedTransientAllowed Class Option ... 35
Packages ... 35
Applications ... 36
Relationship between Schemas and Applications ... 36
Jade Language ... 36
Exception Handling ... 36
System Exceptions ... 36
User Exceptions ... 36
Jade Platform Development Environment ... 37
Jade Platform Runtime Environment ... 37
Jade Object Manager ... 37
What the Jade Object Manager Handles ... 38

Chapter 2 Getting Started ... **40**

Signing On and Off ... 41
Initiating a Jade Session ... 41
What Happens Next ... 41
Signing On to the Jade Platform Development Environment ... 42
Signing Off from the Jade Platform Development Environment ... 43
Signing Off from a Runtime Jade Application ... 44
Using the Jade Platform Development Environment ... 44
Development Source Control ... 45
Source Control Authentication ... 47
Authentication Experience ... 48

Contents iv

Personal Access Tokens ... 48
Authentication Methods ... 48
Authentication Workflow ... 49
Basic Authentication ... 49
Microsoft or Azure Authentication ... 49
GitHub Authentication ... 50
Bitbucket Authentication ... 50
Credential Configuration ... 50
Credential Authority Setting ... 50
Configuring Team Options ... 52
Cloning the Remote Repository ... 53
Committing Source Changes to the Local Repository ... 54
Source Control Staging Categories and Statuses ... 57
Pushing Source Changes to the Remote Repository ... 58
Pulling Commits from the Remote Repository ... 61
Checking Out a Source Control Branch or Tag ... 63
Checking Out a Remote Branch ... 65
Checking Out a Tag ... 66
Navigating around the Jade Platform Development Environment ... 68
Using the Navigation Bar in Hierarchy Browsers ... 70
Using Hierarchy Nodes to Navigate around a Browser Window ... 72
Expanding a Node Branch ... 73
Using Browser Toolbar Buttons ... 73
Changing the Size of Toolbar Icons ... 75
New Workspace Toolbar Button ... 76
Open Workspace Toolbar Button ... 76
Save Toolbar Button ... 76
Cut Toolbar Button ... 76
Copy Toolbar Button ... 76
Paste Toolbar Button ... 76
Print Selected Toolbar Button ... 77
Using the Print Options Dialog ... 77
Browse Classes Toolbar Button ... 77
Browse Primitive Types Toolbar Button ... 77
Browse Interfaces Toolbar Button ... 77
Browse Maps Toolbar Button ... 77
Browse Applications Toolbar Button ... 78
Browse Schema Toolbar Button ... 78
Run Application Toolbar Button ... 78
Load Toolbar Button ... 80
Extract Toolbar Button ... 81
Painter Toolbar Button ... 81
Monitor Toolbar Button ... 82
Toggle Breakpoint Toolbar Button ... 82
Bookmark Toolbar Button ... 82
Reorg Toolbar Button ... 83
History Back Toolbar Button ... 83
History Next Toolbar Button ... 83
Context Help Toolbar Button ... 83
General Help Toolbar Button ... 84
Using the Editor Clipboard Toolbar ... 84
Floating the Editor Clipboard ... 85
Toggling the Display of the Editor Clipboard ... 86
Using Custom Toolbar Buttons ... 87
Adding a Toolbar Button to Run a Script or Application ... 87
Using Jade Platform Development Environment Menus ... 89
Using Popup Menus ... 91
Using the Refresh Facility ... 91
Displaying or Clearing a History of Accessed Entities ... 91
Clearing the History of Accessed Entities ... 92
Using the Quick Navigation Facility ... 92

Contents v

Navigating to a Specified Class in the Class List 93
Navigating to a Specified Interface in the Interface Browser 93
Navigating to the Methods for a Specified Class 94
Displaying a Light or Dark Development Environment Theme 96
Using Function Keys and Shortcut Keys 97
Using Function Keys 98
Using Browser Shortcut Keys 100
Browser Shortcut Keys 100
Caret Movement Shortcut Keys 101
Editor Pane Shortcut Keys 102
Accelerator Shortcut Keys 103
Context-Sensitive Options 103
Extending Selected Blocks Shortcut Keys 104
Insert and Delete Shortcut Keys 105
Using the Mouse within the Editor Text Area 105
Using the Mouse or Shortcut Keys within the Fold Margin 106
Using the Mouse within the Line Number Margin 106
Using Multiple Clipboards when Refactoring Code 106
Painter Shortcut and Function Keys 107
Using Shortcut Keys in Controls 108
Filtering Entries in Combo Box and List Box Text 109
Using Jade AutoComplete Functionality 110
AutoComplete Editor Options 111
Displaying Identifier Usages from the Editor Pane 112
Contents Displayed in the AutoComplete List Box 114
AutoComplete Dictionary Keys 116
AutoComplete List Box Behavior 116
Using the AutoComplete History 118
Toggling the Display of RootSchema Entities 118
Method Definition Error Analysis 118
Method Call Signature Handling 120
InheritMethod Handling 120
Mouse Hover Investigation 120
Manually Invoking the AutoComplete List Box 120
AutoComplete Caching 120
AutoComplete Persistent File 121
AutoComplete Failures 121
AutoComplete Limitations 121
Customizing the Layout of Hierarchy Browser Forms 122
Using View Menu Layout Commands 124
Applying a Layout 124
Applying the Default Jade Layout 124
Applying the Layout of a Specific Schema 125
Clearing the Default Layout 125
Clearing the Layout of a Schema 125
Saving the Default Layout 125
Saving as the Default Layout for a Schema 125
Using Concurrent Windows 125
Using Window Menu Commands 126
Arranging Window Icons 127
Cascading Open Windows 127
Closing All MDI Forms 127
Tiling Windows Horizontally 128
Tiling Windows Vertically 128
Setting User Preferences 128
Maintaining Accelerator Keys 129
Maintaining Browser Options 130
Maintaining Compiler Output Options 134
Maintaining the Editor Display 135
Maintaining Editor Options 136
Maintaining Exit Options 139

Contents vi

Maintaining Lock Options 140
Maintaining Message Box Suppression Options 140
Maintaining Miscellaneous Options 141
Maintaining Relationship Options 144
Maintaining Schema Options 146
Maintaining Shortcut Keys 147
Action Shortcut Key Category 150
Browse Shortcut Key Category 150
Debugger Shortcut Key Category 151
Edit Shortcut Key Category 152
Editor Shortcut Key Category 153
Find Shortcut Key Category 160
Inspect Shortcut Key Category 160
Monitor Shortcut Key Category 161
Painter Shortcut Key Category 162
UnitTest Shortcut Key Category 163
Windows Shortcut Key Category 164
Maintaining Source Management Options 164
Maintaining Status List Options 166
Maintaining Text Templates 166
Maintaining Window Options 168
Adding a Theme 170
Editing a Theme 171
Sending Messages to Other Developers 172
Maintaining Debugger Breakpoints from the Editor Pane 173
Performing Edit Actions 174
Undoing an Action 175
Redoing an Action 175
Cutting Text 176
Copying Text 176
Pasting Text 177
Editing Multiple Selections and Rectangular Selections 177
Multiple Carets 178
Rectangular Selection Virtual Space 178
Pasting Behavior 178
Stream format pasting examples 179
Rectangular format pasting examples 179
Editor Clipboard Notes 180
Selecting All Editor Pane Content 180
Replacing Editor Pane Indentation 181
Recording and Replaying a Series of Keystrokes 181
Starting to Record the Temporary Macro 182
Stopping the Recording of the Temporary Macro 182
Playing the Temporary Macro 182
Editing the Temporary Macro 182
Selecting a Persistent Macro from the Library 186
Keystroke Macro Language 186
Example Keystroke Macros 188
Comment Selection 188
Uncomment Selection 189
Return to Saved Position 189
Add an Entry to the Method Change History 189
Jade Platform Online Help 189
HTML5 Online Help 190
Product Information Library in Portable Document Format 190
Overview 191
Getting Adobe Reader 192
Obtaining Assistance when Using Adobe Reader 192
Searching the Product Information Library using Adobe Reader 192
Selecting Indexes 193
Printing PDF Documents 194

Obtaining Help in a Window or Dialog 194
Obtaining Help in the Editor Pane 195
Obtaining General Help 196
Obtaining Jade Version Information 196
Creating Context Links to Your Own Application Help File 196
Web-based HyperText Markup Language (HTML) Online Help 197
Adobe Portable Document Format (PDF) Online Help 197
Using Bookmarks for Navigation around a PDF File 200
Creating Context Links for a Windows Help File 202

Chapter 3 Defining Classes in Your Own Schema 204

Defining Your Own Schema 205
Defining a Schema 205
Removing a User-Defined Schema 208
Accessing a Schema 209
What Happens Next 210
Validating Schema Definitions 210
System Map Files 213
Using the Class Maps Browser 215
Maintaining Map Files 216
Adding or Changing a Map File 217
Defining Applications 217
Setting an Application 223
What Happens Next 225
Using the Class, Primitive Types, or Interface Browser 225
Opening a Class, Primitive Types, or Interface Browser 227
Displaying Bubble Help in Browser Lists 228
Displaying Lists and the Editor in the Current Browser Window 228
Toggling Property, Method, and Constant Access Type Display 229
Toggling the Display of Methods and Conditions 230
Toggling the Display of Entities in Imported Classes 230
Changing the Sort Order of Displayed Classes 230
Showing Inherited Methods, Conditions, Properties, and Constants 231
Refreshing the Browser 232
Using the Class List, Primitive Type List, or Interface List Window 233
Using the Properties List Window 233
Displaying Control Property Values in the Editor Pane 234
Using the Methods List Window 235
Notes about the Methods List Window 237
Using the Editor Pane 238
Splitting the Editor Pane View 240
Adding a Method from a Free-Standing Method Editor Pane 240
Reusing the Form Displaying Method Source 241
Tips for Using the Editor Pane 243
Printing Editor Text 245
Displaying Multiple Sheets in the Editor Pane 246
Adding Classes to Your Schema 249
Finding a Schema, Class, Interface, or Primitive Type 249
Setting a Class as the Root Class 252
Displaying All Superschema Classes in Your Class Browser 253
Defining Your Own Classes 253
Defining a Class 255
Defining Membership of a Collection Class 257
Defining Keys for a Dictionary Class 258
Maintaining Dictionary Key Paths 260
Automatic Key Maintenance 261
Resolving Exceptions 261
Setting Key Path References to Null 262
Specifying Class Lifetimes 263
Specifying Text for a Class 265

Tuning Collection Classes 266
Maintaining Class Instance Volatility 267
Defining Options for a Control Class 268
Displaying all References to the Selected Class 269
Using the References Browser 269
Customizing the Class Browser of the Current Schema 270
Adding a Schema View 271
Maintaining a Schema View 271
Opening a Customized Schema View 273
Removing a Class from a Schema View 273
Setting a Schema View 274
Defining Dynamic Clusters 274
Browsing Classes that Are in Use 276
Browsing Classes by Name 278
Browsing Classes by Number 278
Browsing Applications 279
Browsing Processes 279
Browsing Nodes 279
Displaying Process Usages of a Class 280
Displaying Subschema Copies of a Class 280
Compiling All Methods Defined in a Class, Primitive Type, or Interface 282
Searching for or Replacing an Element in a Schema 283
Locating an Element in a Browser Editor Pane 283
Finding the Next Occurrence of Specified Text 286
Searching for Text in a Reverse Direction 286
Locating Text on Which the Caret is Positioned 287
Searching for an Element in all Classes in the Current Schema 287
Viewing Located Search Results 292
Searching the Results of a Previous Search 294
Searching a Browser for an Entity 294
Specifying Text for a Schema Element 295
Using the Free-Standing Editor Window to Define Text 296
Removing a Schema Element 297
Printing a Selected Schema Element 299
Setting Up Your Printer 302
Performing a Reorganization 303
Using the Reorg Command 304
Reorganizing Your Schema 304
Reorganizing All Schemas that Require Reorganization 306
Initiating the Reorganization Transition 307
Aborting an Incomplete Reorganization 308
Restarting an Interrupted Reorganization 308
Exiting from Jade when a Reorganization is Required 308
Reorganizing the Database from the Command Line 309
action 310
fastBuildBTreeCollections 311
initiateTransition 311
noReorgRecovery 311
reorgAllowUpdates 312
replayableReorg 312
schemas 312

Chapter 4 Defining Methods, Properties, Constants, and Conditions 313

Defining and Compiling Jade Methods and Conditions 314
Adding Jade Methods to Classes or Primitive Types 315
Adding Conditions to Classes or Primitive Types 321
Condition Examples 323
Conditions on Primitive Types 324
Using the Method Status List Browser 325
Examples of Jade Methods 327

Contents ix

Using Bubble Help in the Editor Pane 328
Saving Methods as Another Name 330
Defining Translatable Strings from the Editor Pane 330
Inserting a Translatable String into an Existing Jade Method 331
Jade Method Syntax Coloring 332
Compiling Methods 333
Recompiling All Methods in the Database 334
Using the Compiler Output Viewer 335
Compiler Output Messages 338
Renaming Methods 339
Using Constructors and Destructors 339
create Method Constructors 341
Constructor Method Example 341
Finding a Method Source Position 342
Going to a Specified Method Line 342
Highlighting Methods in Lists of Methods 343
Displaying Method References, Implementations, and Messages 344
Displaying all Method References in the Current Schema 345
Displaying all Method Implementations in the Current Schema 347
Displaying Interface Methods to which the Selected Method is Mapped 347
Displaying all Local Method References 347
Displaying all Local Method Implementations 348
Displaying References to Local Implementors of a Method 348
Displaying Implementor References to a Method 348
Displaying Messages Sent by the Current Method 348
Using the Methods Menu of the Messages Window 349
Displaying Message References 349
Displaying Message Implementors 349
Searching for Unused Schema Entities 349
Displaying Unused Schema Entities 353
Displaying Unreferenced Methods 354
Displaying Unused Local Variables and Parameters 358
Removing Unused Local Variables 362
Locating Possible Transient Object Leaks 362
Using a Method in a Workspace or the JadeScript Class 367
Using a Workspace 367
Compiling and Executing Code in a Workspace 368
Saving Your Workspace 369
Using the User Input Dialog 369
Debugging a Workspace Method 370
Using a JadeScript Method 370
Executing and Debugging a JadeScript Method 371
Running a Unit Test 371
Viewing Code Coverage Results to Analyze Methods in Your Applications 372
Refactoring a Jade Method 372
Creating a New Class 373
Declaring an Unknown Local Identifier as a Local Variable 373
Extracting Existing Logic and Creating a New Method 374
Generating a Method Stub from the Editor Pane 377
Promoting a Local Constant to be a Type Constant 378
Promoting a Local Variable to be a Parameter of the Current Method 379
Promoting a Local Variable to be a Property on the Current Class 379
Promoting a Method Value to be a Local Method Constant 380
Renaming or Changing an Entity 380
Extracting a Method Selected in a Methods List 382
Defining Properties 383
Adding an Attribute Property 383
Adding a Reference Property 386
Defining an Implicit Reference 387
Using Inverse Reference Properties 388
Defining an Inverse Reference Property 390

Dynamic Clusters and Properties 393
Design-Time Dynamic Property Use 396
Deciding Between Static and Design-Time Dynamic Properties 397
Defining a Dynamic Attribute Property 397
Defining a Dynamic Reference Property 398
Adding a Method Mapping for a Property 400
Displaying all References to the Selected Property or Constant 401
Displaying all References that Can Update the Selected Property 402
Displaying all References that Can Read the Selected Property 403
Displaying all Local References to the Selected Property 404
Displaying all Relationships to a Class 405
Using the Relationship View Window 405
Specifying Your Relationship View Window Options 406
Zooming in to and out from the Relationship View Window 407
Defining Class, Primitive Type, and Interface Constants 408
Adding a Class, Primitive Type, or Interface Constant 408
Defining a Class, Primitive Type, or Interface Constant 409
Constant Definition Tips 410
Sorting Constants by Value 410
Promoting Class Constants to Global Constants 411
Printing Class, Primitive Type, or Interface Constants 411
Using Class, Primitive Type, or Interface Constants in Your Methods 412
Defining Global Constants 412
Accessing the Global Constants Browser 412
Finding a Global Constant 413
Adding a Global Constant Category 415
Adding a Global Constant 415
Printing Global Constants 417

Chapter 5 Using the Painter and Form Wizard 418

Using the Jade Painter 419
Accessing the Painter 419
Creating a Form 420
Specifying Your Form Preferences 421
Adding a New Form 423
Printer-style Forms 425
Adding Controls to Your Form 426
Adding Container Controls 428
Pushing a Control to the Bottom 428
Pushing all Parents and the Control to the Bottom 429
Bringing a Control to the Top 429
Bringing all Parents and the Control to the Top 429
Selecting all Controls on Your Form 429
Locating a Control on Your Form 429
Obtaining Help for a Control Type 429
Checking for Duplicate Control Accelerators 429
Finding the Available Control Caption Accelerator Characters 430
Changing the Runtime Tab Sequence 431
Control Order on Touchscreens 432
Adding or Maintaining Parameters for WebJavaApplet Controls 432
Displaying a Hierarchical List of Controls on a Form 434
Defining Control Name Prefixes 437
Translating Control Properties 438
Using the Translatable Property Browser 439
Allowing Control Docking 441
Initializing the JadeRichText Control 442
Defining the Layout of Your Form 442
Sizing Your Controls 443
Alignment Controls 443
Spacing Your Controls 444

Examples of Laying Out Controls 445
Using Grids to Position Controls on Forms 446
Using Alignment Hairs to Position Controls on Forms 447
What Happens Next 449
Maintaining Properties for Your Form or Control 449
Using the Name Combo Box 451
Using the Control About Box Icon 451
Using the Property Page Dialog Icon 451
Using the Update all selected controls Icon 451
Maintaining Common Properties 452
Maintaining Specific Control Properties 452
Maintaining Font and Color Properties 453
Maintaining Size and Position Properties 453
Designing Menus 453
Using the Menu Design Command 454
Maintaining Menu Item Properties 456
Maintaining Menu Text Properties 457
Maintaining Menu Security Properties 458
Inserting a Menu Item Using the Insert Button 458
Deleting a Menu Item Using the Delete Button 459
Moving a Menu Item 459
Example of Designing a Menu 459
Creating the Menu Bar 459
Creating the First Submenu Level 460
Adding Further Submenu Levels 460
Adding Standard Help and Window Menus 461
Adding More Menu Design Features 462
Defining Methods for Your Form 462
Testing Your Form 462
Selecting Skins for Painted Forms 463
Saving Your Form 464
Editing an Existing Form 464
Using the Keyboard and Mouse to Edit Your Form 465
Selecting a Control to Edit 465
Changing the Size of a Selected Control 465
Selecting Multiple Controls 465
Changing the Size of Selected Controls 466
Moving Controls about Your Form 466
Copying a Control 466
Displaying the Properties Dialog for a Control 467
Positioning a New Control on Your Form 467
Placing a Control on a Container Control 467
Removing a Control from Your Form 467
Making Small Positional Adjustments to a Control 468
Changing a Control Type 468
Effects of Editing an Existing Form 468
Removing ActiveX Controls from Forms 468
Painter Menus 469
File Menu 470
New Form Command 470
Edit Form Command 470
Form Wizard Command 471
Run Form Command 471
Select Skin Command 471
Save Form Command 471
Save Form As Command 472
Copy Form Into Locale Command 472
Close Form Command 472
Delete Form Command 472
Delete Current Form Command 472
Menu Design Command 473

Browser Command 473
Exit Painter Command 473
Edit Menu 474
Cut Command 474
Copy Command 475
Paste Command 475
Select All Command 476
Find Command 476
Undo Last Layout Command 476
Controls Menu 476
Add Control Command 477
Delete Control Command 477
Change Type Command 478
Select Parent Command 478
Select All Children Command 478
Select All Siblings Command 478
Push to Bottom Command 478
Push all Parents and Control to Bottom Command 479
Bring to Top Command 479
Bring all Parents and Control to Top Command 479
Check For Duplicate Control Accelerators Command 479
Find Available Caption Accelerator Characters Command 480
Tab Ordering Command 480
Layout Menu 480
Alignment Command 480
Left Command 481
Right Command 481
Top Command 482
Bottom Command 482
Centre Horizontally Command 482
Centre Vertically Command 483
Spacing Command 483
Space Evenly Down Command 484
Space Evenly Across Command 484
Spread Down Container Command 484
Spread Across Container Command 485
Vertically Adjacent Command 485
Horizontally Adjacent Command 486
Vertical Standard Spacing Command 486
Horizontal Standard Spacing Command 487
Top/Right Command Buttons Command 487
Bottom/Right Command Buttons Command 487
Size Command 488
Same Width Command 489
Same Height Command 489
Same Height and Width Command 489
Standard Size Command 490
Suspend Parent Alignments Command 491
Options Menu 491
Grid Command 492
Edit Control Palette Command 492
Show Alignment Hairs Command 492
Lock Control Positions Command 492
Control Hierarchy on Top Command 493
Properties on Top Command 493
Border all Controls Command 493
Hide Control Palette Command 493
Hide Alignment/Size Palette Command 494
Hide Tools Palette Command 494
Hide Status Bar Command 494
Window Menu 494

Show Control Hierarchy Dialog Command 495
Show Properties Dialog Command 495
Help Menu 495
Index Command 495
Painter Toolbars 496
Tools Toolbar Button Functions 497
Control Palette Button Functions 498
Editing the Control Palette 500
Alignment and Size Palette Button Functions 502
Using Form Wizard to Create a Form 503
What Form Wizard Produces 503
Form Functions 504
Form Layout 504
Using the Generated Form 505
Find Button 506
Add Button 506
Delete Button 506
Change Button 506
Clear Button 506
Refresh Button 506
Help Button 507
Browse Buttons 507
Types of Controls 507
Methods 508
Accessing the Form Wizard 508
Specifying Your Form Name and Title 509
Specifying the Target Class 509
Specifying the Visible Instances 509
Selecting the Presentation Style for the Instances 510
Selecting the Functionality of Your Form 510
Selecting Browse Functionality for Your Form 510
Selecting Find Functionality for Your Form 511
Selecting the Features Displayed on Your Form 511
Removing Features Selected for Display 512
Changing the Sequence of Displayed Features 512
Selecting Captions and Control Names 512
Incorporating a Toolbar, Menu, and Status Line 513
Initiating the Form Build 514
Monitoring Your Form Build 514
Copying a Form into another Locale 514

Chapter 6 Using the Object Inspector 516

Using the Quick Inspect Toolbar 516
Inspector Security 518
Inspecting Object Instances 519
Inspecting Instances of the Current Class 519
Inspecting All Object Instances 520
Inspecting Transient Instances 521
Using the Inspector Form 521
Using the File Menu 523
Using the Options Menu 523
Using the Call Stack Menu 524
Using the History Menu 525
Using the Help Menu 526
Index Command 526
About Command 526
Inspecting Variable Instances in the Debugger 526

Chapter 7 Using the Jade Debugger 528
Starting an Application in Debug Mode 529
Displaying Code in the Method Source Window 531
Restrictions 532
Using Debugger Menu Commands 532
Debug Menu 532
Breakpoints Menu 533
Variables Menu 533
View Menu 534
Options Menu 534
Window Menu 534
Help Menu 535
Executing Code in the Debugger 535
Continuing Code Execution 535
Setting the Animate Speed and Toggling the Bubble Help Display 535
Stopping Code Execution in Animate Mode 536
Executing the Current Line of Code 537
Controlling Debugger Execution 537
Using Breakpoints in the Debugger 538
Setting or Unsetting Breakpoints 539
Displaying All Breakpoints in the Current Schema 540
Using Conditional Breakpoints and Pass Counts 541
Displaying Source Code for a Breakpoint 543
Removing and Clearing Breakpoints 543
Using Watches in the Debugger 544
Selecting a Method in the Call Stack Window 545
Adding a New Watch 545
Removing a Watch 546
Displaying the Current Exception Handler Stack 546
Viewing Local Variables in the Debugger 547
Viewing Creation or Deletion Call Stack Information 549
Enabling or Disabling the Capture of Object Call Stacks 549
Inspecting and Modifying Values in the Debugger 549
Inspecting an Object Reference 551
Terminating a Debugger Session 551
Ending a Debugger Session 551

Chapter 8 Adding External Methods and External Functions to Classes 553
Adding External Methods to Classes 553
Writing External Methods 556
Defining External Functions 556
Defining and Maintaining External Functions 557
Adding an External Function 557
Modifying the External Function Template 558
Compiling Your External Function 559
Example of Definition of an External Function 559
Example of Usage of an External Function 560
Viewing References to an External Function 560
Extracting an External Function or Library 561
Maintaining Libraries for External Methods and Functions 561
Adding a Library 562

Chapter 9 Defining ODBC Inquiry Relational Views and Ad Hoc Indexes 564
Overview 565
Maintaining Relational Views 565
Defining Your Relational View 566
Using the Relational Menu 567
Adding a Relational View 567

Naming Your Relational View 567
Specifying Relational View Options 567
Selecting Classes and User-Defined Tables for Your Relational View 569
Setting the Root Class 571
Setting the Default Included Object Features 572
Setting the Visibility of Protected Features 574
Setting the Visibility of Derived Tables 575
Refining the Visibility of Class Features 577
Renaming Tables and Columns 579
Building Your Relational View 582
Removing a Relational View 582
Printing a Relational View 582
Changing a Relational View 583
Extracting a Relational View 583
Loading a Relational View 583
Maintaining Ad Hoc Indexes 584
Defining an Ad Hoc Index 585
Using the Ad Hoc Index Menu 587
Adding an Ad Hoc Index 587
Defining Ad Hoc Index Membership 588
Defining Ad Hoc Index Keys 589
Defining Ad Hoc Index Text 590
Changing an Ad Hoc Index 592
Deleting an Ad Hoc Index 592
Building an Ad Hoc Index 592
Canceling an Ad Hoc Index Build 593
Dropping an Ad Hoc Index 593
Loading Ad Hoc Definitions 593
Saving Ad Hoc Definitions 594
Running the Ad Hoc Index Controller Application 594

Chapter 10 Extracting and Loading Schemas 596

Jade Extract and Load Facilities 596
Extracting Your Schema 598
Specifying Your Extract Options 600
Extracting Multiple Schemas 602
Multiple Schema File Syntax 603
Specifying Your Schema Options 604
Specifying Your Parameter File Options 607
Parameter File Syntax 608
Specifying Selective Options 609
Specifying Your Application Options 611
Specifying Your Change Options 612
Extracting a Schema View 614
Specifying Your Schema View Extract Options 614
Specifying Your Schema View Options 615
Extracting a Specific Class, Method, or Primitive Type 615
Extracting a Selected Class 616
Extracting a Selected Method 616
Extracting All User-Defined Methods in a Selected Primitive Type 617
Extracting All Schema-Defined Methods 617
Extracting Schemas as a Non-GUI Client Application 617
Extracting Schemas in Sections 622
Rejoining Sections of an Extracted Schema 623
Extracting Classes as a Non-GUI Client Application 623
Loading Your Schema 625
Merging a Schema in the Load Process 626
Changes Requiring Data Reorganization 627
Loading Partial versus Complete Schema Definitions 628
Confirmation of Deletion 628

Invoking the Load Process 629
Specifying Your Load Options 630
Selecting the File or Files to Load 631
Specifying the Load Order when Loading Multiple Files 633
Specifying Advanced Load Options 635
Encrypting Schema Source Files 637
Extracting Encrypted Schema Source Files 638
Loading Encrypted Schema Source Files 638

Chapter 11 Internationalization 640
Overview 641
Forms Translation Styles 642
Compiling Translatable Strings 643
Jade Painter 643
Loading Locales 643
Maintaining Locales using Jade Command Files 644
Maintaining Forms Management Styles using Jade Command Files 644
Searching for Translatable Strings and Forms at Run Time 644
Subschema Copies of Subclassed Forms 644
Maintaining Locales for Your Schema 645
Selecting Your Locales 646
Adding Locales 646
What Happens when You Add a Locale 647
Removing Locales 649
Cloning Locale Forms and Strings 649
Adding and Maintaining Formats 651
Adding a Short Date Format 652
Adding a Long Date Format 653
Adding a Time Format 655
Adding a Numeric Format 656
Adding a Currency Format 658
Maintaining Formats 659
Viewing References to Formats 660
Removing a Format 660
Translating Strings 661
Using the String Browser 662
Adding Strings 664
Searching for Text in Existing Strings 665
Searching for Strings that Reference the Current String 667
Viewing References to a String 668
Running an Application from the String Browser 669
Handling Translatable Strings on Message Box Button Captions 669
Message Box Translatable String Handling for Button Captions 669
Programmatically Maintaining Translatable Strings 670
Adding a New Translatable String 671
Updating an Existing Translatable String 671
Extract Translatable Strings Method Example 672
Load Translatable Strings Method Example 673
Translating Forms 676
Using the Form Browser 677
Using the Find Form Dialog 678
Using the Jade Translator 679
Jade Translator Utility 679
Jade Translator Menus 681
File Menu 682
Exiting from the Jade Translator Utility 682
Edit Menu 682
Using the Undo Command 682
Using the Cut Command 682
Using the Copy Command 682

Using the Paste Command 683
Options Menu 683
Using the Do Not Show Form Previews Command 683
Using the Show Form Previews Command 683
Window Menu 684
Arrange Icons 684
Cascade 684
New Window 684
Tile Horizontal 684
Tile Vertical 685
Help Menu 685
Index 685
About Jade Translator 686
Translating Messages 686

Chapter 12 Adding and Maintaining HTML Documents 687

Overview 687
Accessing the HTML Document Browser 688
Using the HTML Wizard to Add an HTML Document 689
Naming your HTML Document and its Source 690
Specifying the Class Name and Its Property Names 691
JADE_TAG Tag Notes 694
Changing an HTML Document 695
Editing an HTML Document 695
Extracting and Loading an HTML Document 695
Extracting a Specific HTML Document 696
Saving an HTML Document as a File 696
Reloading an HTML Document 696
Removing an HTML Document 697
Reimplementing Methods to Interrupt the Processing Cycle 698
HTML Document Implementation 698
Method Handling 699
Generating Data for JADE_TAG Tags Only 700

Chapter 13 Using Method Views to Bookmark Workflows 703

Overview 703
Adding a Method View 704
Displaying All Method Views that You Created 704
Adding Methods to a Method View 705
Removing a Method from a Method View 707
Using the Methods View Browser 707
Opening a Method View 708
Changing a Method View 708
Copying a Method View 709
Removing a Method View 709
Extracting a Method View 709
Showing Method Views Created by All Developers 711

Chapter 14 Adding and Maintaining Interfaces 712

Overview 713
Jade Interface Implementation 713
Benefits of Using an Interface 713
Accessing the Interface Browser 714
Toggling the Display of Entities in Imported Interfaces 715
Defining an Interface 716
Maintaining an Interface 717
Adding Interface Methods 717
Compiling an Interface Method 719

Specifying Text for an Interface 720
Viewing the Implementation of an Interface Method 720
Displaying the History of an Interface 721
Viewing Current Interface Mappings 722
Viewing Interface Method Mappings 722
Locating an Interface 724
Displaying Interface Mappings 724
Displaying all References to the Selected Interface 725
Implementing an Interface 726
Updating an Existing Interface Method Implementation 729
Stopping the Implementation of an Interface 731
Using Interfaces in your Applications 731
Code Re-Factoring 731
Frameworks and Packages 732
New System Development 732
Interface-Related Methods and Properties 733
Printing an Interface 734
Extracting an Interface 735
Removing an Interface 736

Chapter 15 Using the Relational Population Service (RPS) Wizard 737

Overview 737
Defining an RPS Mapping 738
Using the RPS Relational Menu 738
Adding an RPS Mapping 739
Setting Up the RPS Options 740
Selecting Classes for RPS 742
Mapping Classes to Tables 744
Mapping Subclasses to a Class Map 745
Creating a New Class Map 746
Adding Additional Tables to a Class Map 746
Removing a Table 746
Removing Tables with no Columns 746
Selecting Columns for Tables 747
Removing Tables 749
Removing Tables with no Properties from the Map 749
Exposing Properties for All Tables 749
Exposing Properties for a Selected Table 749
Clearing Exposed Properties for All Tables 749
Clearing Exposed Properties for a Selected Table 750
Exposing Special Columns for All Tables 750
Exposing Special Columns for a Selected Table 750
Clearing Exposed Special Columns for All Tables 750
Clearing Exposed Special Columns for a Selected Table 750
Toggling the Display of Virtual Properties 751
Toggling the Display of Condition-Safe Virtual Properties 751
Toggling the Display of Methods 751
Toggling the Display of Many-to-Many Properties 752
Displaying Tree Lines in Features and Filters Panes 752
Adding Many-to-Many Relationships to an RPS Mapping 752
Mapping Many-to-Many Relationships 753
Mapping a Subclass to a Selected Class 754
Removing Selected Junction Table Class Maps 755
Maintaining RPS Column Mappings 755
Building the RPS Mapping 758
Removing an RPS Mapping 759
Changing an RPS Mapping 760
Printing an RPS Mapping 761
Extracting an RPS Mapping 761
Specifying Selective RPS Extract Options 763

Loading an RPS Mapping ... 764
Creating an Exclude Command File ... 765

Chapter 16 Importing External Components ... 766

Overview ... 766
Maintaining ActiveX Control and Automation Server Objects ... 768
Importing ActiveX Control Libraries and Automation Libraries ... 768
Selecting Your ActiveX Type Library ... 770
Specifying a Name for Your ActiveX Type Library ... 771
Naming Object Classes in the ActiveX Type Library ... 772
Naming Interfaces in the ActiveX Type Library ... 774
Naming Constants in the ActiveX Type Library ... 776
Changing the Generated ActiveX Schema ... 777
Removing an ActiveX Type Library ... 777
Extracting and Loading ActiveX Schema Definition Data ... 778
Registering an ActiveX Server ... 778
Unregistering an ActiveX Server ... 779
Maintaining .NET Objects ... 780
Importing .NET External Component Libraries ... 780
Selecting Your .NET Assembly ... 782
Specifying the Name of Your .NET Assembly ... 783
Specifying Names for Imported .NET Objects ... 784
Naming Object Classes in the Imported .NET Library ... 787
Naming Members in .NET Classes ... 789
Naming Constants in .NET Classes ... 791
Changing the Generated .NET Schema ... 792
Removing a .NET Library ... 792
Extracting and Loading .NET Schema Definition Data ... 793
Maintaining OpenAPI Objects ... 795
Importing OpenAPI Specifications ... 795
Selecting Your OpenAPI Specification ... 796
Specifying the Name of the REST API ... 798
Renaming Proxy Objects ... 800
Applying a Regex to Proxy Names ... 801
Renaming Properties and Methods of Proxy Classes ... 802
Confirming the Importing of the OpenAPI Specification ... 803
Removing an OpenAPI Specification ... 804
Displaying All Superschema External Components in Your Current External Components Browser ... 806

Chapter 17 Using the C# Exposure Wizard ... 807

Overview ... 807
Accessing the Exposure Browser ... 808
Adding a C# Exposure Definition ... 809
Setting Up Your C# Exposure Options ... 810
Selecting Classes for Inclusion in Your C# Exposure Definition ... 811
Selecting Features for Inclusion in Your C# Exposure Definition ... 812
Removing a Class from the C# Exposure ... 814
Exposing Properties for All Classes ... 815
Exposing Properties for a Selected Class ... 815
Clearing Exposed Properties for All Classes ... 815
Clearing Exposed Properties for a Selected Class ... 815
Exposing Methods for All Classes ... 816
Exposing Methods for a Selected Class ... 816
Clearing Exposed Methods for All Classes ... 816
Clearing Exposed Methods for a Selected Class ... 816
Exposing Constants for All Classes ... 817
Exposing Constants for a Selected Class ... 817
Clearing Exposed Constants for All Classes ... 817
Clearing Exposed Constants for a Selected Class ... 817

Contents xx

Toggling the Display of Methods 818
Toggling the Display of Constants 818
Displaying Tree Lines in the Features Pane 818
Mapping C# Features 818
Saving Your C# Exposure 820
Generating the C# Classes 821
Displaying a Hierarchical Exposure Browser 823
Toggling the Display of a Composite View of the Hierarchy Exposure Browser 825
Changing a C# Exposure Definition 826
Removing a C# Exposure Definition 826
Extracting a C# Exposure 827
Finding an Existing C# Exposure 827
Automating C# Exposure Generation and Extraction 828

Appendix A Glossary 831

Before You Begin

The *Development Environment User's Guide* is intended as the main source of information when you are using the Jade Platform development environment to develop a user application.

Who Should Read this Guide

The main audience for the *Development Environment User's Guide* is expected to be developers who are unfamiliar with the Jade Platform.

What's Included in this Guide

The *Development Environment User's Guide* has seventeen chapters and one appendix.

Chapter 1	Provides an overview of Jade concepts and terminology
Chapter 2	Provides instructions for signing on and off and using the Jade Platform development environment
Chapter 3	Provides instructions for defining applications and classes in your own user-defined schema, and for performing a schema reorganization
Chapter 4	Provides instructions for defining Jade methods, properties, constants, and conditions
Chapter 5	Provides instructions for using the Jade Painter, including the Form Wizard
Chapter 6	Provides instructions for inspecting your Jade database
Chapter 7	Provides instructions for using the Jade debugger
Chapter 8	Provides instructions for defining external methods and external functions
Chapter 9	Provides instructions for defining ODBC inquiry relational views and ad hoc index definitions
Chapter 10	Provides instructions for extracting and loading schemas
Chapter 11	Provides instructions for localizing (translating) Jade to suit your local requirements
Chapter 12	Provides instructions for adding and maintaining HTML documents
Chapter 13	Provides instructions for defining and maintaining method views, which group methods from any class in any schema into a named workspace
Chapter 14	Provides instructions for defining and maintaining interfaces
Chapter 15	Provides instructions for defining a Relational Population Service (RPS) mapping using the Relational Population Service wizard
Chapter 16	Provides instructions for importing and maintaining ActiveX and .NET external component objects
Chapter 17	Provides instructions for generating a set of C# exposure classes that are built into a .NET class library
Appendix A	Glossary

Conventions

The *Development Environment User's Guide* uses consistent typographic conventions throughout.

Convention	Description
Arrow bullet (❯❯)	Step-by-step procedures. You can complete procedural instructions by using either the mouse or the keyboard.
Bold	Items that must be typed exactly as shown. For example, if instructed to type **foreach**, type all the bold characters exactly as they are printed. File, class, primitive type, method, and property names, menu commands, and dialog controls are also shown in bold type, as well as literal values stored, tested for, and sent by Jade instructions.
Italic	Parameter values or placeholders for information that must be provided; for example, if instructed to enter *class-name*, type the actual name of the class instead of the word or words shown in italic type. Italic type also signals a new term. An explanation accompanies the italicized type. Document titles and status and error messages are also shown in italic type.
Blue text	Enables you to click anywhere on the cross-reference text (the cursor symbol changes from an open hand to a hand with the index finger extended) to take you straight to that topic. For example, click on the "Using Function Keys and Shortcut Keys" cross-reference to display that topic.
Bracket symbols ([ ])	Indicate optional items.
Vertical bar (|)	Separates alternative items.
`Monospaced font`	Syntax, code examples, and error and status message text.
ALL CAPITALS	Directory names, commands, and acronyms.
Small font	Keyboard shortcut keys.

Key combinations and key sequences appear as follows.

Convention	Description
Key1+Key2	Press and hold down the first key and then press the second key. For example, "press Shift+F2" means to press and hold down the Shift key and press the F2 key. Then release both keys.
Key1,Key2	Press and release the first key, then press and release the second key. For example, "press Alt+F,X" means to hold down the Alt key, press the F key, and then release both keys before pressing and releasing the X key.

Related Documentation

Other documents that are referred to in this guide, or that may be helpful, are listed in the following table, with an indication of the Jade operation or tasks to which they relate.

Title	Related to…
Database Administration Guide	Administering Jade databases

Title	Related to…
Developer's Reference	Developing or maintaining Jade applications
Development Environment Administration Guide	Administering Jade development environments
Encyclopaedia of Classes	System classes (Volumes 1 and 2), Window classes
Encyclopaedia of Primitive Types	Primitive types and global constants
External interface Developer's Reference	Developing Jade applications using external interfaces
Jade Initialization File Reference	Maintaining Jade initialization file parameter values
Installation and Configuration Guide	Installing and configuring Jade
Object Manager Guide	Jade Object Manager administration
Jade Platform Differences Guide	Platform differences when running Jade applications
Report Writer User's Guide	Using the Jade Report Writer to develop and run reports
Schema Load User's Guide	Loading schemas into a deployed Jade database
Synchronized Database Service (SDS) Administration Guide	Administering Jade Synchronized Database Services (SDS), including Relational Population Services (RPS)
Thin Client Guide	Administering Jade thin client environments
Web Application Guide	Implementing, monitoring, and configuring web applications

Chapter 1 Jade Platform Concepts and Terminology

This chapter covers the following topics.

- Overview
- Types
- Objects and Classes
  - Object Lifetimes
- Features
- Properties
- Methods
  - Conditions
  - Constraints
- Encapsulation
- Inheritance and Polymorphism
- Schemas
  - Accessing and Updating Instances of Classes in Other Schemas
  - Controlling the Use of Elements in Other Schemas
  - Packages
- Applications
- Jade Language
- Exception Handling
- Jade Platform Development Environment
- Jade Platform Runtime Environment
- Jade Object Manager

Overview

Jade is a powerful, robust multi-user integrated object-oriented development environment that enables you to rapidly build enterprise-wide systems of any complexity that are scalable on open platforms.

This chapter contains an overview of the components of Jade. (For definitions of the terms used in Jade, see Appendix A, "Glossary".)

Jade Object Model

Jade enables you to model and construct your information systems in terms of a set of self-contained components called *objects*.

The Jade definition of objects and the ways in which they are organized and interact with each other together constitute the Jade *object model*. The Jade object model:

- Combines data and operations into objects
- Uses messages to communicate between objects
- Groups similar objects into classes
- Maintains a class hierarchy to provide inheritance of data and procedures

The Jade object model consists of a set of abstract concepts and definitions: its concrete implementation is achieved primarily in terms of the Jade Object Manager.

The Jade Object Manager is a central and fundamental component of the overall Jade architecture. It supports all aspects of the Jade object model, including:

- Object creation and deletion
- Message passing
- Determining runtime behavior based on the class of an object
- Managing efficient storage and retrieval of objects and relationships in the database

For more details, see "Jade Object Manager", later in this chapter.

Types

In Jade, types characterize the behavior that can be applied to an instance or value. Types are:

- Classes, which group "similar" objects, including a definition of the data those objects contain (properties), and the actions they can perform (methods).

  Classes encapsulate structure and operations into a cohesive software unit, which hides the implementation details while exposing only the interface to the class; for example, **Employee** would be a class in a human resources system.

- Primitive types, which have a defined null value that can be tested for by using the **null** language identifier.

  Properties defined as primitive types represent a value, and not a reference to an object.

- Interfaces, which enable non-related classes to be grouped to capture their similarities, without the need to artificially force a class relationship.

  This mechanism provides a set of methods that are guaranteed to be available on any implementing class. (For more details, see Chapter 14, "Adding and Maintaining Interfaces".)

Objects and Classes

An *object* is any entity, either real or abstract, that exhibits some well-defined behavior and that has a unique identity.

A *class* groups all objects that share the same set of properties and methods. Every object is an instance of one, and only one, class. A *class* describes the common characteristics of a set of objects. An object is often referred to as an instance of the class that describes it.

For details about defining your own classes, see "Defining Your Own Classes", in Chapter 3.

Object Lifetimes

A *persistent* object is stored in the database and remains accessible after the end of the process that created it. A *transient* object is not stored in the database and is destroyed when it is explicitly deleted or when the process terminates.

One of the primary applications of persistent object storage is that of sharing objects between several applications. This information sharing occurs at a very high level of semantics, because the objects are not merely passive data but carry with them their behavior definition. In this context, the Jade Object Manager can be used as a common repository for shared objects. It enables you to apply the same design methods to the persistent and transient segments of your application.

Features

Features say something about or do something to objects. Features include properties (or data) and methods (or operations).

Properties

Properties (*attributes* and *references*) represent the internal data storage of an object. The state of an object at any time is determined by the values stored in each of its properties. Properties can be:

- Attributes

  Primitive variables such as numbers, strings, or boolean values (for example, age, name, and permanent)

- Single-valued references

  References to other objects

- Multiple-valued references

  References to collections of other objects (for example, a collection of **employees** in a **company** object)

Properties can be public, protected, read-only, virtual, or subschema-hidden. If properties are protected, they are exclusive to that object and cannot be accessed by any other object. There is one copy of a property for each instance of a class.

For details about defining your own properties, see "Defining Properties", in Chapter 4.

Methods

A *method* (procedural code) implements an aspect of the behavior of an object. Methods constitute the procedural interface of an object. An object can invoke methods of another object, by sending that object a *message*.

Objects cannot perform any action by themselves. You use methods to send messages to objects to perform the appropriate actions.

A method may require parameters, and may return a result. The method that is executed is determined at run time, based on the class of the object to which the message is sent.

Methods can also be public or protected. Public (and read-only) features represent the interface of an object or the set of services the object provides to other objects in the system.

For details about defining your own methods, see "Defining and Compiling Jade Methods and Conditions", in Chapter 4.

Conditions

A *condition* is a declarative restricted method that returns a Boolean result. Conditions cannot be reimplemented from a superschema or superclass; that is, they are not polymorphic.

The *boolean-expression* that is returned can contain only references to the properties of the class and calls to other conditional expressions of that class or its superclass.

Any parameters that you specify can be **constant** parameters only. Only **if** and **return** instructions can be used in condition methods.

For details about defining conditions, see "Adding Conditions to Classes or Primitive Types" under "Defining and Compiling Jade Methods and Conditions", in Chapter 4.

Constraints

A *constraint* is a condition used to maintain automatic inverse references when the specified condition is satisfied.

A condition that is used as a constraint cannot have parameters. Only **if** and **return** instructions can be used in a constraint condition.

Only automatic references can have a constraint. When the manual side of the inverse reference is set, the condition used as the constraint is evaluated and the automatic inverse is set only if the value of the condition is **true**. If the automatic reference is a **Collection** type, the condition is applied to the members of the collection.

For details about defining constraint conditions, see "Adding Conditions to Classes or Primitive Types" under "Defining and Compiling Jade Methods and Conditions", in Chapter 4.

Encapsulation

Encapsulation is the clear separation between the external interface of a class and its internal implementation; that is, the packaging together of properties and methods (or data and operations) for an object.

Encapsulation hides the implementation (internal structure) of an object from the surrounding system. Other objects can access only properties and methods that define the external characteristics of the object.

An object can hide some or all of its implementation from other objects, but can provide methods by which other objects can manipulate its internal state.

An object is like the proverbial engineering "black box". You can change the way a method is implemented without changing the object's interface or its relationships with other objects.

You can hide implementation details to:

- Protect an object from arbitrary and unintended use, or from accidental corruption.
- Hide the implementation details of an object from the users of that object, so those users understand the class interface rather than its implementation. This enables you to change the implementation of an object without affecting other users of the object.

Properties are protected by default.

Inheritance and Polymorphism

Much of the power of object-orientation comes from arranging classes in a hierarchy.

Classes that are:

- Higher in the hierarchy represent more-general characteristics
- Lower in the hierarchy represent more-specific characteristics

Inheritance enables new classes to be defined as subclasses (derived classes) of existing superclasses (base or *parent* classes).

The subclass automatically inherits the properties and methods (features) of the superclass. You can also define your own properties and methods for a subclass or reimplement existing superclass methods, to extend or modify the inherited behavior. You cannot reimplement conditions.

In Jade, a message can be sent to any object and the method that is executed depends on the class of the object to which the message was sent.

The ability of an object to respond differently to a specific message, depending on its type, is known as *polymorphism*; that is, it has the ability to assume several forms.

See also "Controlling the Use of Elements in Other Schemas", later in this chapter.

Schemas

A *schema* is the highest-level organizational structure in Jade, and represents the object model for a particular domain.

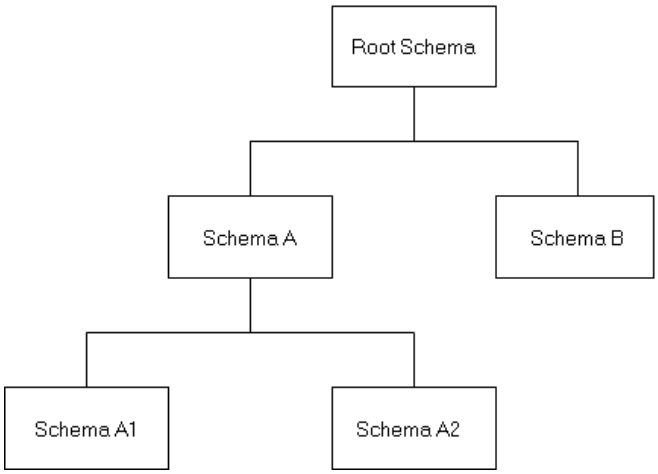
A schema is a logical grouping of classes, together with their associated methods and properties. These effectively define the object model upon which your applications are based.

Jade provides the **RootSchema**, which is always at the top of the schema hierarchy. The **RootSchema** provides essential system classes; for example, the **Object** class (the root of the class hierarchy), **Collection** classes, and the **File**, **Exception**, and **Form** classes. Because these classes are defined in the **RootSchema**, they can be accessed from all subschemas.

You can add subschemas to the **RootSchema**. You can also add new classes to a schema, methods, properties, and constants to a schema class, or methods and constants to an existing class defined in a superschema.

For details about defining user-defined schemas, see "Defining Your Own Schema", in Chapter 3. See also "Controlling the Use of Elements in Other Schemas", later in this chapter.

A sample schema hierarchy is shown in the following diagram.



The schema hierarchy is analogous to a class hierarchy, and similar terminology is used. For example, **A1** and **A2** are *subschemas* of **A**, while **A** is a *superschema* of **A1** and **A2**, in the above example.

Subschemas inherit all of the classes, methods, and properties that are defined in their superschemas. Thus, in the above example, schema **A1** would have access not only to its own classes but also to those defined in **A** and the **RootSchema**. However, schemas on separate branches of the hierarchy (for example, **A1** and **A2**) cannot "see" each other at all unless you use *packages* to expose functionality across the schema hierarchy. If you do not use imported packages, no part of **A1** is directly accessible to **A2**, nor any part of **A2** to **A1**. (See also "Packages", later in this chapter.)

As well as adding entirely new classes to a schema, you can also add methods or constants to an existing class defined in a superschema. For example, schema **A** might add methods to the **File** class (which is defined in the **RootSchema**) to perform functions that are relevant only in that schema. Any such methods would be visible only in schema **A** and its subschemas; they would not be visible in the **RootSchema**. While you can add methods or constants in this way, you cannot add properties to a class defined in a superschema.

When you begin developing a system using Jade, you would normally begin by adding a subschema to the **RootSchema** and you would then define all your classes, properties, and methods within that schema. (For details, see "Defining Your Own Schema", in Chapter 3.) This ensures that your object model (class hierarchy) is clearly packaged and is kept distinct from the system classes. For larger and more complex development efforts, a hierarchy of user schemas may be necessary to adequately represent the object model.

Accessing and Updating Instances of Classes in Other Schemas

At run time, an application can access instances of classes defined in schemas other than the current schema of the application, or one of its superschemas. See also "Controlling the Use of Elements in Other Schemas", later in this chapter.

Note This applies only to applications at run time. In the Jade Platform development environment, methods and properties refer only to classes defined in the currently set schema or one of its superschemas.

Typically, you would use this feature to build common or generic applications in a schema that can operate on subclass instances defined in subschemas. For example, assume that we have the following schemas, classes, and relationships (shown in pseudo schema syntax).

```
Schema TreatmentCentreSchema
    Class TreatmentCentre subclassOf Object
```

```
        allDepartments: DepartmentDict inverseOf Department::myTreatmentCentre
    Class Department subclassOf Object
        myTreatmentCentre: TreatmentCentre inverseOf
                    TreatmentCentre::allDepartments
        allPatients: PatientDict inverseOf Patient::myDepartment
    Class Patient subclassOf Object
        myDepartment: Department inverseOf Department::allPatients
        allDetails: PatientDetailDict inverseOf PatientDetail::myPatient
    Class PatientDetail subclassOf Object
        myPatient: Patient inverseOf Patient::allDetails
    Class DepartmentDict subclassOf MemberKeyDictionary membership Department
    Class PatientDict subclassOf MemberKeyDictionary membership Patient
    Class PatientDetailDict subclassOf MemberKeyDictionary membership
                    PatientDetail

Schema XRaySchema subschemaOf TreatmentCentreSchema
    SubschemaCopyClass Department
        Class XRayDepartment
    SubschemaCopyClass PatientDetail
        Class XRayPatientDetail

Schema OncologySchema subschemaOf TreatmentCentreSchema
    SubschemaCopyClass Department
        Class OncologyDepartment
    SubschemaCopyClass PatientDetail
        Class OncologyPatientDetail
```

The definitions of a *root class* and a *subschema copy class* are as follows.

- A *root class* is the uppermost definition of a class.
- A *subschema copy class* is any copy of a class in a subschema of the schema in which the root class is defined. Subschema copy classes are added to a subschema whenever a superschema class is subclassed in the subschema or has methods added to it in the subschema.

In the above example, **TreatmentCentreSchema** defines the common classes and applications to run a medical treatment centre, which may comprise many departments. Each department can have many patients and each patient can have many **PatientDetail** objects, which are used to record department-specific treatment details and history. **TreatmentCentreSchema** is used to encapsulate common applications that can operate on any department and any patient; for example, billing, producing patient lists and histories, or departmental statistics. There is only ever one instance of **TreatmentCentre**, with each department in the centre included in the **allDepartments** dictionary.

TreatmentCentreSchema has two subschemas (**XRaySchema** and **OncologySchema**), which implement X-Ray and Oncology department applications, respectively. Each of these schemas has a **Department** and **PatientDetail** subclass to which department-specific properties and methods can be added. There will be one instance each of **XRayDepartment** and **OncologyDepartment**, both of which will be included in the **allDepartments** dictionary on the singleton **TreatmentCentre** object.

As inverse references cannot span schemas, all relationships between **TreatmentCentre**, **Department**, **Patient**, and **PatientDetail** are defined in the uppermost schema; that is, in **TreatmentCentreSchema**. You could not, for example, define an inverse from **OncologyDepartment** to **TreatmentCentre**, or the reverse. With this structure, the implementation of each department can be encapsulated in its own schema. This is useful for both development (to aid encapsulation) and deployment (where each schema could be deployed or sold separately).

The common applications that run from **TreatmentCentreSchema** can process any department in the **allDepartments** dictionary of the **TreatmentCentre** object, even if it is an instance of a subschema department. Typically, these applications will access only **Department** properties and methods, where some methods may be reimplemented in subschema departments.

If a method has been reimplemented for a particular subschema department, the appropriate reimplementation will be called (provided that the reimplementation is not on a subschema copy class).

When an application accesses an instance of a class that is not visible to its current schema, it is able to access the instance as though it can see the root class branch for that class (that is, the uppermost definition of the class and its superclasses). This means that all attributes and references are available, as well as all *root class* methods.

The only things that are not available are methods defined on subschema copies of the class or any of its superclasses. In our example, the **XRaySchema Department** class is a subschema copy of the **TreatmentCentreSchema Department** class. Any methods added to the **Department** subschema copy class in **XRaySchema** are not available to applications running from **TreatmentCentreSchema** that access **XRayDepartment** instances.

Tips Any methods on a class (either new methods or reimplementations of superclass methods) that are to be invoked from superschema applications should avoid calling subschema copy class methods entirely.

If you are reimplementing a superclass method that is to be invoked from superschema applications, *do not* reimplement the method on a subschema copy class, as it will not be visible to the superschema. Instead, reimplement the method on a root subclass; that is, a subclass whose uppermost definition is in the current schema.

Controlling the Use of Elements in Other Schemas

Jade enables you to control how classes, methods, properties, and automation event references implemented in your schemas are used by developers in other schemas; for example:

- Developers of packages (exporters) can control the use of schema entities by package users (importers). For details about packages, see "Packages", later in this chapter.
- Developers of framework and re-use schemas can control how classes and methods implemented in their schemas are used by developers in other schemas (for example, by limiting the subclassing and object lifetimes of classes).

For details about method options, see "Method Options" under "Jade Language Notation", in Chapter 1 of your *Developer's Reference*.

If you want to restrict the class or method to this schema and its subschemas, use the **final** option.

If you want to restrict the class or method only in subschemas, use the **subschemaFinal** option. To specify that the class or method is not available in subschemas, use the **subschemaHidden** option. (See also "Class Lifetime Options", later in this section.)

final Option

The **final** class and method option enables you to specify that a class cannot be subclassed or a method cannot be reimplemented in a subclass. Use the **final** option to restrict subclassing and method reimplementation in this schema and its subschemas.

Note This option does not restrict methods and constants being added to subschema copies of the class.

The Define Class dialog provides the **Final (Class cannot be subclassed)** check box and the Jade Method Definition and External Method Definition dialogs provide the **Final** check box. As these check boxes are unchecked by default, classes can be subclassed or a method can be reimplemented in a subclass unless you specify otherwise. (For details, see "Adding Classes to Your Schema" or "Defining and Compiling Jade Methods and Conditions", in Chapter 3 or Chapter 4, respectively.)

Tip You can change this method option (for an existing method) from the editor pane, by adding or removing the **final** method option in the method signature and then recompiling that method.

subschemaFinal Option

The **subschemaFinal** class and method option enables you to specify that a class or method can be extended or reimplemented in its local schema but not in a subschema. When applied to a class, there are no restrictions on the class in its local schema but the class cannot be subclassed in subschemas.

When applied to a method, there are no restrictions on the method in its local schema, but the method cannot be reimplemented anywhere in a subschema.

Note This option does not restrict methods and constants being added to subschema copies of the class.

The Define Class, Jade Method Definition, and External Method Definition dialogs provide the **Subschema Final** check box. As this check box is unchecked by default, classes and methods can be extended or reimplemented in the local schema and in subschemas unless you specify otherwise. (For details, see "Adding Classes to Your Schema", in Chapter 3, or "Defining and Compiling Jade Methods and Conditions", in Chapter 4.)

Tip You can change this method option (for an existing method) from the editor pane, by adding or removing the **subschemaFinal** method option in the method signature and then recompiling that method.

As an example, consider the following situation.

```
Schema A
Class C1 - subschemaFinal
    Class C2 - subclass of C1 – inherits subschemaFinal from C1
        Class C3 - subclass of C2 – inherits subschemaFinal from C2
```

As the developer of schema **A**, you may want to specify that classes **C1**, **C2**, and **C3** cannot be subclassed in a subschema. Making class **C1 subschemaFinal** accomplishes this, as classes **C2** and **C3** inherit this option from class **C1**.

subschemaCopyFinal Option

The **subschemaCopyFinal** method option enables you to specify that a method cannot be reimplemented in a subschema copy class.

The Jade Method Definition and External Method Definition dialogs provide the **Subschema Copy Final** check box. As this check box is unchecked by default, methods can be extended or reimplemented in the local schema, subschemas, and in subschema copy classes unless you specify otherwise. For details, see "Defining and Compiling Jade Methods and Conditions", in Chapter 4.

Tip You can change this method option (for an existing method) from the editor pane, by adding or removing the **subschemaCopyFinal** method option in the method signature and then recompiling that method.

As an example, consider the following situation.

```
Schema A
Class C1 - method m1 - subschemaCopyFinal
```

```
Class C2
    Class C3 - method m2 calls m1
```

As the developer of schema **A**, you may want to guarantee that when method **m2** of class **C3** is called, it *always* uses the schema **A** implementation of **C1::m1**. However, you will allow class **C3** to be subclassed and method **m1** to be reimplemented on subclasses.

The following therefore applies.

```
Schema B - subschema of Schema A
Subschema Copy Class C1 (reimplementation of m1 not allowed on subschema
                         copy)
    Subschema Copy Class C2 (reimplementation of m1 not allowed because
                             subschemaCopyFinal)
        Subschema Copy Class C3 (reimplementation of m1 not allowed because
                                 subschemaCopyFinal)
            Class C4 - reimplemented m1 allowed
```

subschemaHidden Option

The **subschemaHidden** option enables you to mark entities and features as being available only in the local schema; that is, they are not available for use in any subschemas or exported packages.

The Jade compiler enforces that **subschemaHidden** entities are not referenced by subschema code, and subschema browsers do not display entities and features that are **subschemaHidden** in a superschema.

The Define Class, Jade Method Definition, External Method Definition, Add Condition, Define Attribute, and Define Reference dialogs provide the **Subschema Hidden** check box. As this is unchecked by default, classes, methods, properties, and automation event references are available in subschemas unless you specify otherwise. (For details, see "Adding Classes to Your Schema", in Chapter 3, or "Defining and Compiling Jade Methods and Conditions", in Chapter 4.)

Tip You can change this method option (for an existing method) from the editor pane, by adding or removing the **subschemaHidden** method option in the method signature and then recompiling that method.

Class Lifetime Options

Class options enable you to restrict the lifetime of instances of a class and its subclasses. These options differ from the **real** and **abstract** class options.

Specifying that only transient instances of a class are allowed is particularly useful for package and framework development when you want to enforce that no persistent instances of specific classes will ever exist, so that reorganization issues on subsequent deployments do not arise.

Note You cannot define a class when all three class or subclass options are unchecked (that is, at least one of the class lifetime check boxes and one of the subclass lifetime check boxes must be checked).

If you create an abstract superclass (for which no instances can be created), you may still want to set the lifetime options described in the following subsections so that you restrict the lifetime of **real** subclasses.

persistentAllowed Class Option

The **persistentAllowed** class option enables you to specify that a class can have persistent instances. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

If you uncheck the **Allow Persistent Instances** check box on the **Lifetime** sheet of the Define Class dialog, an exception is raised at every attempt to create a persistent instance of the class. An error is raised if you attempt to construct a persistent object from within a method (that is, you call **create <*object-name*> persistent**).

Persistent instances of subclasses are restricted according to the **subclassPersistentAllowed** class option, described later in this section.

When you create a new class, the default setting for the **persistentAllowed** class option is determined as follows.

- The default value is the same as the **subclassPersistentAllowed** option value of the immediate superclass.
- If the **subclassPersistentAllowed** option of the immediate superclass is set to **false**, the **Allow Persistent Instances** check box on the **Lifetime** sheet of the Define Class dialog is disabled.

You cannot set the **persistentAllowed** option to **false** for a class where persistent instances of the class exist. You must delete these instances before the change is made.

transientAllowed Class Option

The **transientAllowed** class option enables you to specify that a class can have non-shared transient instances. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

If you uncheck the **Allow Transient Instances** check box on the **Lifetime** sheet of the Define Class dialog, an exception is raised at every attempt to create a non-shared transient instance of the class. An error is raised if you attempt to construct a transient object from within a method (that is, you call **create <*object-name*> transient**). Non-shared transient instances of subclasses are restricted according to the **subclassTransientAllowed** class option, described later in this section.

When you create a new class, the default setting for the **transientAllowed** class option is determined as follows.

- The default value is the same as the **subclassTransientAllowed** option value of the immediate superclass.
- If the **subclassTransientAllowed** option of the immediate superclass is set to **false**, the **Allow Transient Instances** check box on the **Lifetime** sheet of the Define Class dialog is disabled.

sharedTransientAllowed Class Option

The **sharedTransientAllowed** class option enables you to specify that a class can have shared transient instances. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

If you uncheck the **Allow Shared Transient Instances** check box on the **Lifetime** sheet of the Define Class dialog, an exception is raised at every attempt to create a shared transient instance of the class. An error is raised if you attempt to construct a shared transient object from within a method (that is, you call **create <*object-name*> sharedTransient**).

Subclass shared transient instances are restricted according to the **subclassSharedTransientAllowed** class option, described later in this section.

When you create a new class, the default setting for the **sharedTransientAllowed** class option is determined as follows.

- The default value is the same as the **subclassSharedTransientAllowed** option value of the immediate superclass.
- If the **subclassSharedTransientAllowed** option of the immediate superclass is set to **false**, the **Allow Shared Transient Instances** check box on the **Lifetime** sheet of the Define Class dialog is disabled.

subclassPersistentAllowed Class Option

The **subclassPersistentAllowed** class option enables you to set the **persistentAllowed** and **subclassPersistentAllowed** class option values for all subclasses of this class.

When the **subclassPersistentAllowed** option is set to **false** on a superclass by unchecking the **Allow Persistent Subclass Instances** check box on the **Lifetime** sheet of the Define Class dialog, the **Allow Persistent Instances** and **Allow Persistent Subclass Instances** check boxes are disabled on the **Lifetime** sheet of the Define Class dialog for all subclasses of the current class so that you cannot change these values. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

An exception is raised at every attempt to create a persistent instance of a subclass. If persistent instances of subclasses exist, you must delete these instances before making the change.

subclassTransientAllowed Class Option

The **subclassTransientAllowed** class option enables you to set the **transientAllowed** and **subclassTransientAllowed** class option values for all subclasses of this class.

When the **subclassTransientAllowed** option is set to **false** on a superclass by unchecking the **Allow Transient Subclass Instances** check box on the **Lifetime** sheet of the Define Class dialog, the **Allow Transient Instances** and **Allow Transient Subclass Instances** check boxes are disabled on the **Lifetime** sheet of the Define Class dialog for all subclasses of the current class so that you cannot change these values. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

subclassSharedTransientAllowed Class Option

The **subclassSharedTransientAllowed** class option enables you to set the **sharedTransientAllowed** and **subclassSharedTransientAllowed** class option values for all subclasses of this class.

When the **subclassSharedTransientAllowed** option is set to **false** on a superclass by unchecking the **Allow Shared Transient Subclass Instances** check box on the **Lifetime** sheet of the Define Class dialog, the **Allow Shared Transient Instances** and **Allow Shared Transient Subclass Instances** check boxes are disabled on the **Lifetime** sheet of the Define Class dialog for all subclasses of the current class so that you cannot change these values. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

Packages

Packages allow one schema (the *importing* schema) to import and make use of classes from another schema (the *exporting* schema). The importing and exporting schemas do not have to be related (that is, they can exist in completely independent schema branches). For details about building and reusing packages, see Chapter 8 of the *Developer's Reference*.

The exporting schema defines the classes that it exports in packages. A single schema can export multiple packages. You can include only classes with public access in a package. When you export a class, you can select the properties, methods, and constants that are to be made available to importing schemas.

The import name of a package is defined at the time the schema is imported. (By default, the imported name is the same as the exported name.) An imported package does not have to have the same name as the exported package that it imports. However, you cannot change package definitions and export characteristics at import time, nor can you subclass imported classes. Imported classes implicitly inherit from the nearest **RootSchema** subschema copy class of the importing schema.

Applications

An *application* is an end-user runtime system, defining a collection of forms and other runtime information such as the start-up form and the location of the help file.

For details about the types of application, see the **Application** class **applicationType** property in Volume 1 of the *Encyclopaedia of Classes*.

Relationship between Schemas and Applications

Within a single schema, there can be one or many application instances, each providing a different "view" of the object model, with different forms allowing the data in that model to be displayed and possibly updated.

While the various applications in a schema can differ in appearance and functionality, all share the same underlying object model; that is, the object model defined by the schema.

Jade Language

The Jade language is the purpose-built programming language for the Jade environment. The Jade language combines control structures and arithmetic expressions with an object-oriented message-passing syntax. In the Jade language, programs are organized as cooperative collections of objects.

The Jade language is a strongly typed language; that is, the programmer enforces the class of an object by specifying the type of each variable. This strong typing of the Jade language provides the following benefits.

- Reduces the risk of runtime errors due to invoking invalid methods
- Early detection of errors for the programmer

You can write external methods (or routines) for Jade classes in any language that can create a library. External methods are called from Jade as if they were system-provided methods. Jade methods can invoke methods written in other languages and other languages can invoke Jade methods.

For details about using the Jade language, see "Jade Language Reference", in Chapter 1 of the *Developer's Reference*.

Exception Handling

Jade exception handling provides a mechanism that enables you to construct Jade applications that deal properly with exceptions that occur because of abnormal conditions.

For details about exception handling, see "Handling Exceptions", in Chapter 3 of the *Developer's Reference*. See also the following subsections.

System Exceptions

System exceptions are error conditions that are handled automatically by Jade; for example, a lock exception or an error that occurs when opening a file.

User Exceptions

You can define exceptions other than those automatically captured by the system. This enables you to add new attributes and methods specific to your own exception protocol or to override system methods.

Jade Platform Development Environment

The Jade Platform development environment is written in the Jade language. The Jade Platform provides you with a predefined set of classes that comprise a class hierarchy, or framework.

The classes provide the basic functionality required by the:

- Jade Object Manager
- Jade Platform development environment
- Jade dynamic runtime environment

The Jade Platform development environment enables you to define classes, properties, and Jade methods. (For details about defining classes, see Chapter 3. For details about defining methods, properties, constants, and conditions, see Chapter 4.)

The Jade Platform also provides a graphical painter (form builder) which enables you to define the user interface for your application by creating screen forms, controls, and reports (printed forms). *Forms* are windows that contain *controls*, the graphical objects drawn on forms; for example, a list box or an option (radio) button. (For details about using the Jade Painter, see Chapter 5.)

Jade Platform Runtime Environment

The Jade Platform uses the form definitions created in the development environment and stored in the database to build windows at run time. The Jade Platform runtime environment:

- Handles dynamic window creation
- Controls properties and events
- Handles "window" events

For details about runtime applications, see the *Runtime Application Guide*.

Jade Object Manager

The Jade Object Manager supports complex objects in an efficient form that is easy to use. A complex object consists of data and operations to manipulate that data. (A *process* is the activation of a single thread of control.)

A *node* is an installation of Jade that dynamically supports the client role and the server role on the same workstation. A client node is a node initiating a request and a server node is the node processing the request. Each node can take on client or server roles.

A node processing a request received from a server node can invoke a client role for a callback to the initiating client. Nodes handle the multiple threading of processes.

Note In the development architecture, the client and server can be nodes on the Local Area Network (LAN), and not dedicated workstation functions. The Jade Object Manager provides a more-flexible distributed processing environment, as you are not restricted to the rigid client/server partitions enforced by other models.

For a detailed description of Jade processes, see "Jade Object Manager Distributed Processing", in Chapter 1 of the *Object Manager Guide*.

For details, see "Jade Object Manager Distributed Processing", in Chapter 1 of the *Object Manager Guide*.

The principal Jade Object Manager characteristics are:

- Object model functionality
- Seamless interface to the object model
- Extensibility of schema and storage media
- High performance
- Automatic referential integrity support
- Recovery
- Configures, initiates, and controls multiple threads

The Jade Object Manager:

- Encapsulates data and has class independence
- Can store data and methods
- Holds data that can be used only by the methods of the classes; that is, data is designed for specific methods only
- Contains active objects, whose methods are executed in response to requests
- Contains classes that can be reused
- Can reorganize classes without affecting the use of those classes (see also "**Reorganization Restrictions**", in Chapter 14 of the *Developer's Reference*)
- Can contain complex data structures whose complexity need not be known by the end-users

What the Jade Object Manager Handles

The Jade Object Manager is central to the Jade Platform structure, and handles:

- Storage
- Transaction management (ACID-compliant database; that is, one that has atomicity, consistency, isolation, and durability)
- Cache management (more than one process shares a common object cache for persistent objects in a single node)
- Concurrency control
- Dynamic binding

The Jade Object Manager arbitrates between multiple processes making requests, coordinates multiple requests for locks, and maintains details of locked objects, based on lock requests by users.

The Jade Object Manager also handles notifications, and maintains statistical information that can be accessed by the Jade Monitor program, including:

- Licence information
- Details of locked objects and queued locks
- Lodged notification requests
- Users attached to the database
- Operational statistics

The object-oriented languages that can interface with the Jade Object Manager are:

- The Jade language
- Smalltalk
- C++
- Visual Basic

The C or Visual Basic non-object-oriented languages can also use the Jade Object Manager, but as the Jade Platform provides an object-oriented view of the data, maximum benefits accrue when using the Jade Object Manager from an object-oriented language.

See also the:

- *.NET Developer's Reference*
- *External Interface Developer's Reference*
- "Jade Application Programming Interface (API)" and "C-Level Application Programming Interface (API)" chapters, in the *Object Manager Guide*

Chapter 2 Getting Started

This chapter covers the following topics.

- Signing On and Off
  - Initiating a Jade Session
  - Signing On to the Jade Platform Development Environment
  - Signing Off from the Jade Platform Development Environment
  - Signing Off from a Runtime Jade Application
- Using the Jade Platform Development Environment
  - Development Source Control
  - Navigating around the Jade Platform Development Environment
  - Displaying a Light or Dark Development Environment Theme
  - Using Function Keys and Shortcut Keys
  - Filtering Entries in Combo Box and List Box Text
  - Using Jade AutoComplete Functionality
  - Customizing the Layout of Hierarchy Browser Forms
  - Using Concurrent Windows
  - Setting User Preferences
  - Sending Messages to Other Developers
  - Maintaining Debugger Breakpoints from the Editor Pane
  - Performing Edit Actions
  - Recording and Replaying a Series of Keystrokes
- Jade Platform Online Help
  - HTML5 Online Help
  - Product Information Library in Portable Document Format
  - Obtaining Help in a Window or Dialog
  - Obtaining Help in the Editor Pane
  - Obtaining General Help

- Obtaining Jade Version Information
- Creating Context Links to Your Own Application Help File

Signing On and Off

Before you initiate Jade, ensure that your options are specified in the target (command line) for the Jade program, as shown in the following example that runs Jade in single user mode.

```
c:\jade\bin\jade.exe path=c:\jade\system app=Jade server=singleUser
```

For details, see "Jade Configurations", in Chapter 1 of the *Installation and Configuration Guide*.

Initiating a Jade Session

To initiate the Jade Platform development environment

1. Select **Start** from the Taskbar.
2. Optionally, find and open **Jade** from **All apps**.
3. Click the **Jade** icon.

Alternatively, double-click the **Jade** program shortcut on your desktop, if you have created one.

What Happens Next

When you have started Jade, the start-up form displays to whom Jade is licensed, the Jade version, and the name of the:

- Application
- Server
- Schema
- Database path

This information is also displayed when you select the **About Jade** command from the Help menu in the development environment.

The current status is also displayed as the database is opened, each initialization process is actioned, and the Jade application is started.

When the Jade application has started, the Jade sign-on dialog is then displayed.

The user identifier (user id) of the workstation user is displayed in the **User Id** text box. The **Patch No** combo box is displayed only when patch versioning is enabled and one of the following applies.

- A user-defined security library (DLL) is specified in the **DevelopmentSecurityLibrary** parameter in the [JadeSecurity] section of the Jade initialization file and the **JadePatchControlSecurity** parameter in the [JadeSecurity] section is set to **true**.
- If the **DevelopmentSecurityLibrary** parameter is set to the default value of **None**, the **EnablePatchControlExtensions** parameter in the [JadePatchControlExtensions] section is set to **true**.

For details about:

- Patch versioning, see Chapter 3, "Patch Versioning" of the *Development Environment Administration Guide*
- Running a Jade application with a license type of restricted, see Chapter 1, "Running a Jade Application", of the *Runtime Application Guide*

Signing On to the Jade Platform Development Environment

» To sign on to Jade

1. Specify another user id in the **Username** text box, if required.
2. Select the appropriate option in the Select Options group box.
   - The **Browse Classes** option button is selected by default; that is, the Schema Browser is displayed by default when you sign on to Jade.
   - Select the **Form Painter** option button if you want the Jade Painter to be displayed over the Schema Browser background window when you have signed on, to enable you to define or maintain your Jade screen and printed forms.
   - Select the **Administration** option button if you want the Jade Installation Preferences to be displayed when you have signed on, to enable you to specify default preferences for the applications in your Jade environment, or if you want to back up your files in a multiuser environment or to administer patch version control.

   Note It is the responsibility of your system administrator to administer your Jade environment and specify the environmental options when Jade is installed.

   However, you can use the **Preferences** command from a browser window Options menu to specify your own preferences for your user id. For details, see "Setting User Preferences", later in this chapter.
3. If the **Patch No** combo box is displayed, specify or select the patch number with which you want to work. To display patch number details as bubble help, move the mouse over the combo box.

   The patch numbers that are listed in the combo box are those that were assigned to you (based on your user id). Parameters in the [JadePatchControlExtensions] section of the Jade initialization file determine:
   - That the default patch number is used when you specify zero (**0**)
   - That entry of a patch number is required
   - The specified patch number must be unique (that is, it cannot be assigned to another user)
   - The patch number must already exist

   Closed patch numbers are not displayed in the list box area of the **Patch No** combo box. To reopen a closed patch number, you must use the Patches Browser window.

   If you enter a closed patch number and the **PatchNumberCanBeReopened** parameter in the [JadePatchControlExtensions] section of the Jade initialization file is set to **true**, you are warned of this and prompted to confirm that you want to reopen the closed patch. If the parameter is set to **false** or it is not defined, an exception is raised.
4. Click the **Start** button. Alternatively, click the **Cancel** button to exit from Jade.

When you have specified a user id and optional password, the Schema Browser is then displayed by default. For details about the Schema Browser, see Chapter 3. Alternatively, the Jade Painter or the Jade Installation Preferences dialog is then displayed, depending on your selection of the **Form Painter** or **Administration** option button, respectively.

If you specified or selected a patch number, the patch number is validated. If it is valid, a call is then made to your user-defined security library (DLL), passing the user name and patch number as parameters.

When you sign on to the Jade Platform development environment with the default **Browse Classes** option, the Tips dialog is displayed by default.

When you have viewed the current Jade tip, you can:

- Click the **Previous Tip** or the **Next Tip** button to view an earlier or later tip, respectively.
- Click the **Close** button to close the Tips dialog and perform the Browser action that you require.

  Alternatively, you can click the close icon at the upper right corner of the window or select the **Close** command from the Control-Menu to close the dialog.
- Uncheck the **Show tips at start-up** check box if you do not want the Tips dialog displayed every time you sign on to the Jade Platform development environment. The **Show Tips At Start-up** check box on the **Windows** sheet of the Preferences dialog is then unchecked.

  If you have previously unchecked this control to hide the display of the Tips dialog and you now want it displayed when you sign on:

  - Check the **Show Tips At Start-up** check box on the **Windows** sheet of the Preferences dialog.

  The Tips dialog will then be displayed the next time you sign on to the Jade Platform development environment with the default **Browse Classes** option from the Jade sign-on dialog.

Signing Off from the Jade Platform Development Environment

Use the **Exit** command from the File menu to exit from your Jade development work session.

Alternatively, select the **Logoff** command from the File menu to exit from your Jade development work session, in preparation for starting a new work session.

The Jade sign-on dialog is then displayed again, to enable you to sign on as a different user or to invoke the **Administration** or **Form Painter** facility.

When you exit or log off from the Jade Platform development environment:

- Any current Painter work session is closed.
- All locks are released.
- All objects are cleared from memory.

Note Any current runtime Jade application is not terminated when you exit from a Jade development work session.

» To sign off from Jade, perform one of the following actions

- Select the **Exit** command or the **Logoff** command from the File menu.
- Click the close icon at the top right corner of the window or select the **Close** command from the Control-Menu.

- Press Alt+F4.

You are prompted to save any method sources or Workspace windows that have not been saved.

When all required method sources or Workspace windows have been saved, a dialog is then displayed, asking you to confirm that you are sure that you want to exit.

By default, the Exiting Jade dialog is then displayed, to enable you to confirm that you want to exit from Jade. (To suppress the requirement to confirm that you want to exit from Jade, uncheck the **Exit Confirmation** check box in the **Exit** sheet of the Preferences dialog, accessed from the Options menu **Preferences** command.)

» To confirm that you want to exit

- Click the **Yes** button.

  Alternatively, click the **No** button if you want to continue with your Jade session.

After a momentary delay, your Jade development work session is then closed down.

Signing Off from a Runtime Jade Application

Use the application-specific means of exiting from your Jade runtime application (for example, your system may have an Exit menu or a File menu that contains an **Exit** command).

When you exit from a runtime Jade application:

- Any current form is closed.
- All locks are released.
- All objects are cleared from memory.

Note Any current Jade development work session is not terminated when you exit from a runtime Jade application.

» To sign off from a Jade application, perform one of the following actions

- Select the application-specific means of exiting from your Jade application; for example, an **Exit** command from a File menu.
- Click the close icon at the top right corner of the window or select the **Close** command from the Control-Menu.
- Press Alt+F4.

After a momentary delay, your runtime Jade application is then closed down.

Using the Jade Platform Development Environment

This section covers the following topics.

- Development Source Control
- Navigating around the Jade Platform Development Environment
  - Using the Navigation Bar in Hierarchy Browsers
  - Using Hierarchy Nodes to Navigate around a Browser Window

- Using Browser Toolbar Buttons
- Using the Editor Clipboard Toolbar
- Using Custom Toolbar Buttons
- Using Jade Platform Development Environment Menus
- Using Popup Menus
- Displaying or Clearing a History of Accessed Entities
- Using the Quick Navigation Facility

- Displaying a Light or Dark Development Environment Theme
- Using Function Keys and Shortcut Keys
- Filtering Entries in Combo Box and List Box Text
- Using Jade AutoComplete Functionality
- Customizing the Layout of Hierarchy Browser Forms
- Using Concurrent Windows
- Setting User Preferences
- Sending Messages to Other Developers
- Maintaining Debugger Breakpoints from the Editor Pane
- Performing Edit Actions
- Recording and Replaying a Series of Keystrokes

When the Jade Platform development environment outputs a message advising you that an unavailable resource (for example, a form, class, application, lock, and so on) is held by another user, the message includes ***user-name*** **{*process-identifier*}**. This enables you to determine which process to force off, and it minimizes errors when doing so. The following is an example of the additional information output with the message.

```
The Form ViewSchema::UpdateCustomer is in use by Wilbur {6}
```

Development Source Control

A Jade environment and its development environment sit in a larger development ecosystem, which can consist of other development tools, testing frameworks, test systems, and so on. These are often tied together with a version control system (VCS), which uses and supports one or more different workflows, even if the overall development goal is the same.

Jade incorporates a Git client, so that you can use an on-premises or a cloud-based Git provider to move your source and source changes into (push) and out of (pull) your team's source repository, to more-easily incorporate your Jade source into a modern development workflow.

Notes The Jade Git client functionality is fully compatible with other Git clients.

The Jade product information library documents only the Jade implementation of the Git source control; it does not cover the Git technology itself. (If you are not familiar with Git, see https://git-scm.com/book/en/v2.)

A Git client runs on either a presentation client or a standard client. The working directory can be anywhere on the network; it is not restricted to your local file system.

To use the Git client functionality, your Git administrator or team leader would normally create a bare Git repository on a server for the team. For details, see https://git-scm.com/docs/git-init.

Jade forms definition files are optionally encoded in the legacy DDB format as .**ddb** files.

The Source Control feature comprises mainly an integrated Git client working on files extracted to or loaded from the Git working directory, using the Browse menu **Git Source Control Client** command and its submenu commands.

To commit one or more changes in Git, you do not have to be online or connected to the remote repository, because you have a full repository on your local file system. Commits are therefore recorded only in your local repository and they are not transferred to the remote repository until you explicitly decide to share them.

When you modify a file, it is not automatically included in the next commit. You must explicitly mark the changes you want included in the next commit, by adding them to the *staging area*. You can stage complete files only. A hashed string (a string directly generated based on the information it represents) is used instead of an ascending revision number, so that commits in Git are uniquely identified.

As Git is a distributed version control system (DVCS), rather than relying on a central server to store development history, the complete version history is recorded in a local copy, or *clone*, of a remote repository. This allows you to:

- Maintain your own history of changes, independent of any other copy, extract your changes, and publish them to a more-central repository only when you are ready to do so
- Develop your source without needing to access a central server at all times during the development process

A Git clone returns a full-fledged repository; not just a working copy. You then have your own repository on your local machine, including the complete history of the project. You can do everything on your local machine; for example, commit, inspect history, restore older revisions, and so on. Only when you want to share your work with a wider group do you need to connect to a remote server.

The Jade Platform development environment provides source control security hooks. The Git client repository requires authentication and authorization. For details, see "Source Control Functions", in Chapter 2 of the *Object Manager Guide*.

The **Source Management** sheet of the Preferences dialog enables you to define your own Git client options; that is:

- Your Git user name, which is used to identify you as the author for a Git commit operation.
- Your email address, which also identifies you as the author for a Git commit operation.
- Your working (local) directory.

For details, see "Maintaining Source Management Options", later in this chapter.

A typical development cycle iterates over the following steps on your workstation.

1. Add or maintain your Jade changes (for example, classes, methods, or properties).
2. Extract each change (singly, selectively, or as a schema) to your local Git folder.
3. Stage and commit your change or changes to your local repository.

4. Merge your changes.
5. Push your changes to the remote repository.

The **Git Source Control Client** submenu in the Browse menu provides commands that enable you to configure your source control client, to clone a remote repository, to check out a source control branch or tag, to commit changes to your local repository, and to push and pull source changes to and from the remote repository. For details, see:

- Configuring Team Options
- Cloning the Remote Repository
- Committing Source Changes to the Local Repository
  - Source Control Staging Categories and Statuses
- Pushing Source Changes to the Remote Repository
- Pulling Commits from the Remote Repository
- Checking Out a Source Control Branch or Tag
  - Checking Out a Remote Branch
  - Checking Out a Tag

Applies to Version: 2018.0.01 and higher

For details about source control authentication, see "Source Control Authentication", in the following section.

Source Control Authentication

When communicating with a remote Git repository server, the Jade Platform development environment may be required to authenticate with the remote repository service.

The Jade Platform development environment uses Git Credential Manager for Windows (GCM) and extensions to provide authentication and secure Git credential storage.

Note The Git configuration has a helper setting, which when set to the value manager, instructs Git to use the Windows Credential Manager. The Jade Git client uses the GCM, regardless of the value.

The Jade Git client uses the GCM to provide the following functionality.

- Secure credential storage in the Windows Credential Store
- Two-factor authentication support for GitHub and Bitbucket
- Multi-factor authentication support for Azure DevOps (formerly known as Visual Studio Team Services (VSTS) or Visual Studio Online (VSO))
- Personal Access Token generation and usage for Azure DevOps, GitHub, and Bitbucket
- Non-interactive mode support for Azure DevOps backed by Azure Directory
- NTLM/Kerberos authentication for Team Foundation Server(TFS)
- Optional settings for build agent optimization

Applies to Version: 2018.0.01 and higher

Authentication Experience

When you access a Git repository from a Jade Platform development environment for the first time, you are prompted to enter your credentials.

If your account has multi-factor authentication (MFA) configured, you are presented with a provider-specific experience to complete the authentication process. Following completion of the authentication process, your credentials are saved in the Windows credential store and used to process the current and subsequent Git requests.

Note Jade uses a prefix of **jade-git** in the credential key name, which means that keys are easily recognizable and that saved credentials are not shared with other Git client tools.

If you ever need to clear credentials from storage, you can use the Credential Manager in the Windows Control Panel to remove it or use the **cmdkey.exe** tool from the command line. See the Windows **cmdkey** documentation, for details.

Applies to Version: 2018.0.01 and higher

Personal Access Tokens

When the repository hosting service supports the use of Personal Access Tokens (PATs), a PAT is created with read and write access to the repository for each repository that is accessed. The PAT is saved in the credential store and used to process the current and subsequent Git commands.

When a PAT has been created and saved, Git commands that are sent to the server do not prompt for user credentials until:

- The token expires
- The token is explicitly revoked on the server
- Cached credentials are removed

Note When a PAT is created, you may receive an e-mail message from the repository server, advising you that a personal access token has been added to your account.

Applies to Version: 2018.0.01 and higher

Authentication Methods

The method used to authenticate with a Git repository server can vary with the service and configuration.

The Jade Git client supports the following authentication methods.

- Basic authentication
- Windows Integrated Security (NTLM) authentication
- GitHub and Bitbucket two-factor authentication
- Azure Active Directory (AAD) authentication
- Microsoft Live Account (MSA) authentication

Each authentication method has a different authentication experience that uses provider-specific dialogs. For details, see the following subsections.

Applies to Version: 2018.0.01 and higher

Authentication Workflow

The following flows, triggered before a Git command is sent to the remote Git server, explain how authentication works for different authentication methods.

- Basic Authentication
- Microsoft Live Account (MSA) or Azure Active Directory (AAD) Authentication
- GitHub Authentication
- Bitbucket Authentication

Applies to Version: 2018.0.01 and higher

Basic Authentication

1. Check to see if cached credentials exist in the Windows Credential Store under the **jade-git/repository** key.
2. If cached credentials are not found:
    a. A generic Windows security dialog is displayed for the entry of the user name and password of the user.
    b. The specified credentials (user name and password) are stored in the credential store under the **jade-git/repository** key.
3. The acquired credentials are used to perform the **requests** Git command.

Note Credentials are not validated with the server before use, which means invalid credentials could be saved in the credential store and sent to the server with the request.

Applies to Version: 2018.0.01 and higher

Microsoft or Azure Authentication

1. Check to see if cached credentials exist in the Windows Credential Store under the **jade-git/repository** key.
2. If cached credentials are not found:
    a. A dialog is displayed to get the Microsoft Live Account or Azure Active Directory credentials of the user. The dialog shows a multi-factor experience if it is configured for the user's identity.
    b. Acquired credentials are validated with the server.
    c. If the validation succeeds:
        i. A personal access token is created with read and write permissions for the repository.
        ii. The personal access token is stored in the credential store under the **jade-git/repository** key.
3. If cached credential are found, they are validated with the server.
4. If authentication succeeds, the acquired credentials are used to perform the **requests** Git command.

Applies to Version: 2018.0.01 and higher

GitHub Authentication

1. Check to see if cached credentials exist in the Windows Credential Store under the **jade-git/repository** key.
2. If cached credentials are not found:
   a. A GitHub-branded dialog is displayed, prompting for the entry of credential information.

      If two-factor authentication is configured, an additional dialog is displayed so that the two-factor authentication code can be obtained.

   b. Acquired credentials are validated with the server.
   c. If the validation succeeds:
      i. A personal access token is created with read and write permissions for the repository.
      ii. The personal access token is stored in the credential store under the **jade-git/repository** key.
3. If cached credential are found, they are validated with the server.
4. If authentication succeeds, the acquired credentials are used to perform the **requests** Git command.

Applies to Version: 2018.0.01 and higher

Bitbucket Authentication

The Bitbucket authentication workflow is similar to that of GitHub. (For details, see "GitHub Authentication", in the previous section.)

The main differences are the use of Atlassian-branded dialogs and a different two-factor authentication experience.

Applies to Version: 2018.0.01 and higher

Credential Configuration

Configuration settings are available to customize credential management behavior.

The credential settings are configured using the Git configuration files with the same scope or precedence rules as those of Git; that is, global settings override system settings and local (per repository) settings override global settings.

You can use the Git **git config** utility to set, unset, and alter the setting values. For details, see the following Git documentation topics.

- git-config (https://git-scm.com/docs/git-config), for details about getting and setting Git configuration settings
- Customizing Git - Git Configuration (https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration), for more details about Git configuration in general

Note The one Git credential setting that we *do* recommend changing is the **authority** setting.

Applies to Version: 2018.0.01 and higher

Credential Authority Setting

The authentication method is determined by the **authority** setting in the **git credential** namespace. The configuration key is one of:

```
credential.authority

credential.repository-host-domain-name.authority
```

The supported values are **Auto**, **Basic**, **AAD**, **MSA**, **GitHub**, **Bitbucket**, and **NTLM**. (Note that deprecated values and alternative names have been omitted from this list of supported values.)

Use:

- **GitHub** if the host is **github.com** or an on-premises GitHub enterprise server.
- **Bitbucket** if the host is **bitbucket.org** or an on-premises Bitbucket server.
- **AAD** for Azure Active Directory authentication.
- **MSA** for Microsoft Live Account authentication.
- **NTLM** if the host is a Team Foundation Server or another NTLM authentication-based server.

The default value for the credential authority setting is **Auto**, which causes the authentication subsystem to attempt to detect the authority. Auto-detection is based on using the domain name part of the repository URL and a trial-and-error strategy. If the authentication authority cannot be detected, the "Basic Authentication" method is used.

Note The authority auto-detection mechanism depends on recognition of the hosting service contained in the domain part of the repository URL. While auto-detection *does* work for cloud-hosted services such as **github.com** or **bitbucket.org**, it does *not* work for on-premises servers such as GitHub enterprise.

It is strongly recommended that you explicitly set the authority in the Git credential namespace, because the **Basic** authentication method:

- Does not support multi-factor authentication
- Does not support use of personal access tokens
- May not work correctly when the host is a Team Foundation Server requiring **NTLM** authentication

You can set the authority setting by using the Git command line or by editing the relevant configuration file and adding the key. To add a global setting, which applies to all users on the machine, use one of:

```
git config --global credential.repository-service-domain-name.authority

git config --global credential.authority
```

Note The first of the above formats that contains the repository service domain name qualifier takes precedence over the second non-qualified format.

In the following example, Jade sources are maintained in an on-premises GitHub enterprise server.

```
git config --global credential.authority GitHub
```

In the following example, Jade sources are maintained in an Azure DevOps-hosted repository and the developer also accesses other repository-hosting services. In this case, it is important to include the domain name qualifier for each repository host.

```
git config --global credential.microsoft.visualstudio.com.authority AAD
```

The default values for other credential management settings should be suitable for most users. If you need to know more about the available settings, see "Configuration Options" under "Git Credential Manager for Windows"; that is, http://microsoft.github.io/Git-Credential-Manager-for-Windows/Docs/Configuration.html.

The credential subsystem can also be configured using environment variables, which may be useful for build environments. Environment variables take precedence over configuration settings.

For the complete list of environment variables that the GCM understands, see "Environment Variables" (https://github.com/Microsoft/Git-Credential-Manager-for-Windows/blob/master/Docs/Environment.md).

Tip See https://github.com/Microsoft/Git-Credential-Manager-for-Windows/blob/master/Docs/Faq.md (the Git Credential Manager for Windows FAQs) if you have issues when using the Windows Credential Manager.

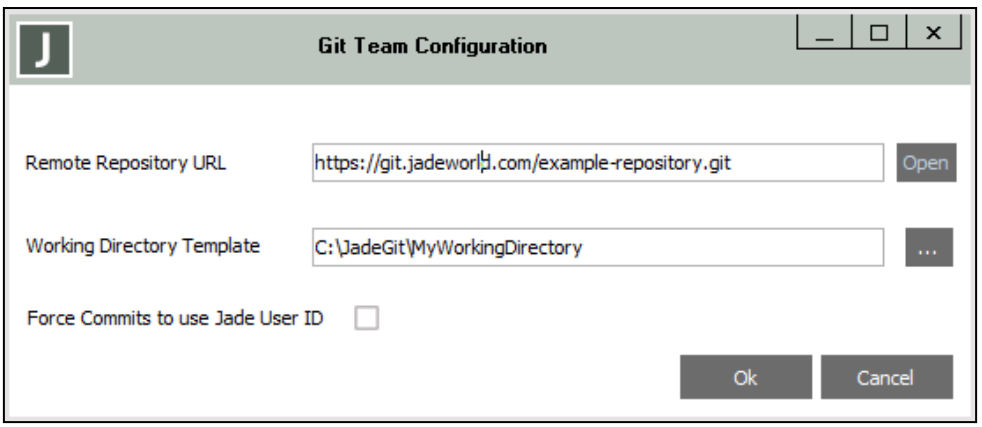
Applies to Version: 2018.0.01 and higher

Configuring Team Options

The Git Team Configuration dialog enables your team leader to configure options that will be used by your entire team for this Jade environment.

To configure your team options

1. To configure the Git team options, select the **Team Configure** command from the **Git Source Control Client** submenu in the Browse menu.
2. The Git Team Configuration dialog, shown in the following image, is then displayed.



3. In the **Remote Repository URL** text box, specify the URL or directory path of your remote repository that was set up by your source control administrator or your team leader.

   Specify a URL or a valid path in this text box if you want to clone the remote repository to a local repository on your workstation. If you attempt to clone with an invalid URL or path, an exception is raised.

   Note As this URL or path is used for the initial clone operation, it is no longer used after the repository is cloned.

   Click the **Open** button if you want to attempt to open the URL in the default browser if it is a website or the path in Windows Explorer if it is a directory .
4. In the **Working Directory Template** text box, optionally browse to and select the working directory template for your team. This is useful in highly standardized environments.

   Note The working directory template is a suggested directory for standardized development setups.
5. If you want to enforce commit operations to use your Jade user identifier, check the **Force Commits to use Jade User ID** check box. This is unchecked, by default, so that your user name can be any value.

When you check this check box, the **Committer Name** text box on the **Source Management** sheet of the Preferences dialog is disabled and set to your Jade user profile name (for example, **jadewilbur1**).

When you commit changes to a repository, both the Git author and Git committer values for the commit operation use the same name and e-mail value.

6. Click the **OK** button. (Alternatively, click the **Cancel** button to abandon your specified values.)

Applies to Version: 2018.0.01 and higher

Cloning the Remote Repository

When you have configured the Git team values, the Git source control client enables you to clone the remote repository to a local repository on your workstation.

» To clone the remote repository

1. To clone the remote repository, select the **Clone** command from the Git Source Control Client submenu in the Browse menu.
2. The Git Clone dialog, shown in the following image, is then displayed.

The **Repository URL** and **Local Path** text boxes are pre-filled with the configured URL of the remote repository that was set up by your source control administrator or your team leader and the path of your working directory, respectively.

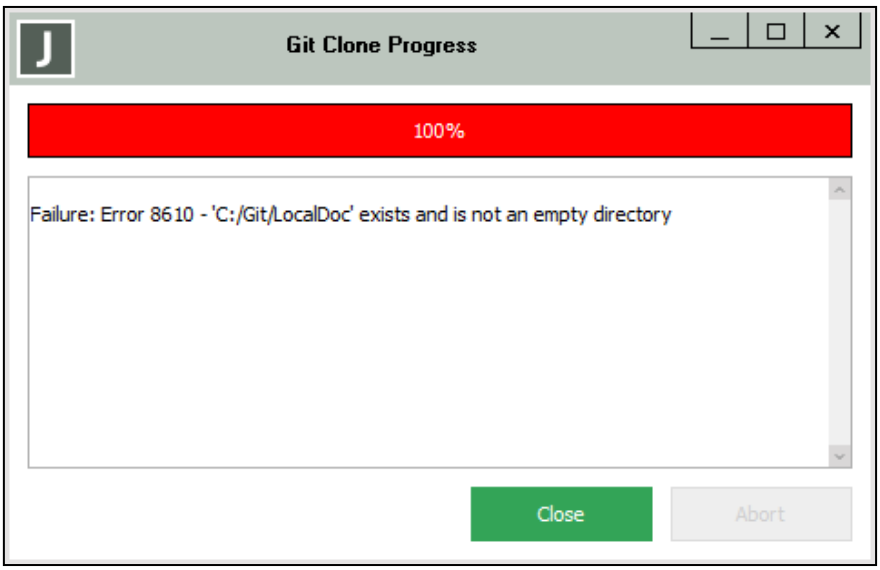
3. If you want to specify a different remote repository, in the **Repository URL** text box, specify the URL the remote repository that you want to clone.
4. If you want to specify a different local path in which the cloned repository is located, in the **Local Path** text box, specify the path or click the **Browse** button to locate the folder on your workstation.
5. Click the **Clone** button. (Alternatively, click the **Cancel** button to abandon your specified values.)

If the path does not exist on your workstation, an error is raised.

You can now iterate through a typical development cycle, as follows.

1. Extract files (singly, selectively, or as a schema) to your local Git folder.
2. Stage and commit your changes to your local repository.
3. Pull the changes of other people in your team down to your local repository.
4. Push your changes to the remote repository.

If an error is detected (for example, the specified local path already contains data), a message box like that shown in the following image is displayed.



Applies to Version: 2018.0.01 and higher

Committing Source Changes to the Local Repository

Because you have a Git repository on your local file system, you do not have to be online or connected to a central (remote) repository. Commits are therefore recorded only in your local repository. They are not transferred to a remote repository until you explicitly decide to share them, by pushing your source changes to the remote repository.

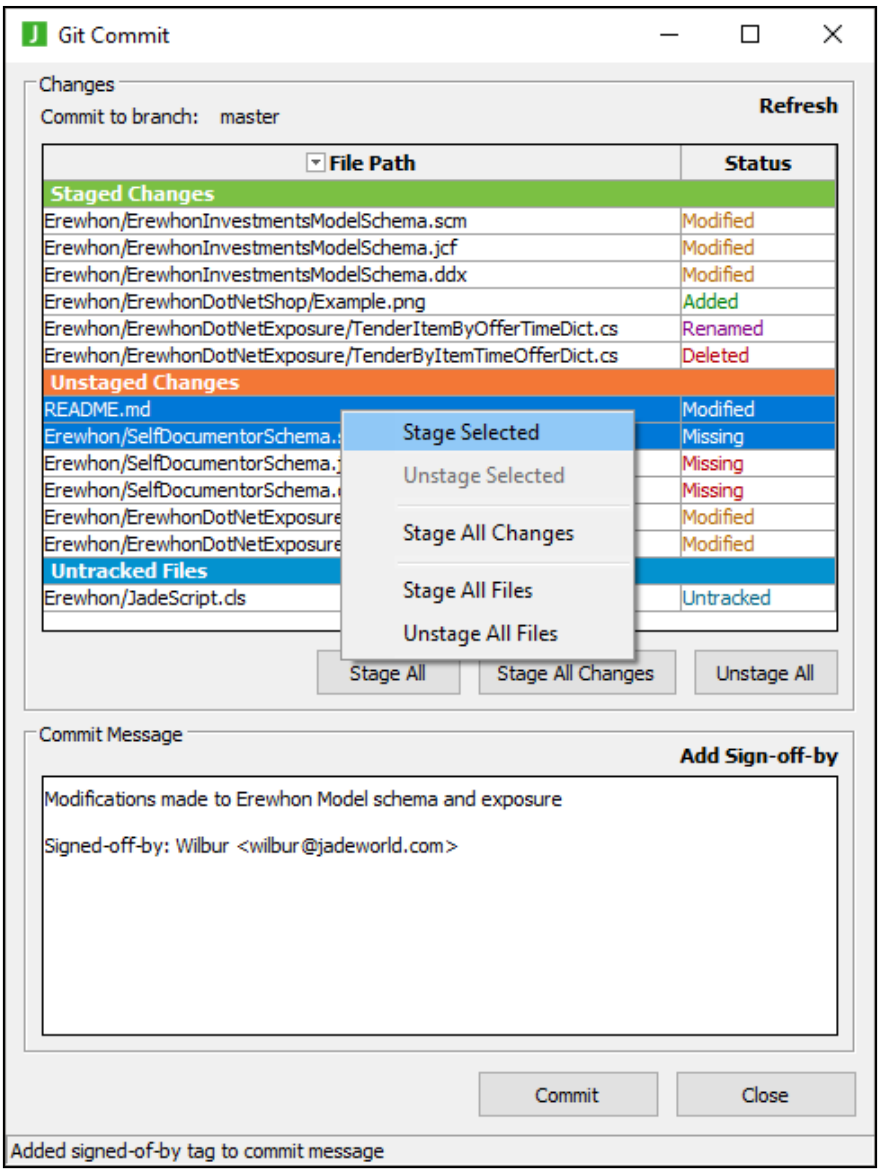
Note You must first have specified your committer name and email address in the Commit Details group box on the **Source Management** sheet of the Preferences dialog.

When you modify a file (for example, extracting a schema or method) it is not automatically included in the next commit. You must explicitly mark the changes that you want in the next commit operation, by adding them to the *staging area* in your local repository. You can commit a complete file only. If a file (for example, a method) has been modified, all changes are staged. However, if you have made more changes *after* the file has been staged, you must stage the file again, to include all of your changes.

To commit changes you have made to the working directory on your local workstation

1. Select the **Commit** command from the **Git Source Control Client** submenu in the Browse menu.

   The Git Commit dialog, shown in the following image, is then displayed.



   For details about the types of changes that can be displayed in the **Changes** table and the statuses that can be displayed, see "Source Control Staging Categories and Statuses", in the following subsection.

2. If you want to:

   - Stage specific changes:

     i. In the Unstaged Changes area of the table, select the rows of the changes you want to stage.

     ii. Right-click, and then select the **Stage Selected** command from the popup menu.

- Stage all unstaged changes, perform one of the following actions.
  - Click the **Stage All Changes** button.
  - Right-click on a row in the table, and then select the **Stage All Changes** command from the popup menu.
  - To stage *all* changes, including untracked files that have never been staged or committed, click the **Stage All** button or right-click on a row in the table and then select the **Stage All Files** command from the popup menu.
- Remove the staged status of specific changes (for example, you want to exclude a change), in the Staged Changes area of the table, select the rows of the changes you want to unstage, right-click, and then select the **Unstage Selected** command from the popup menu.

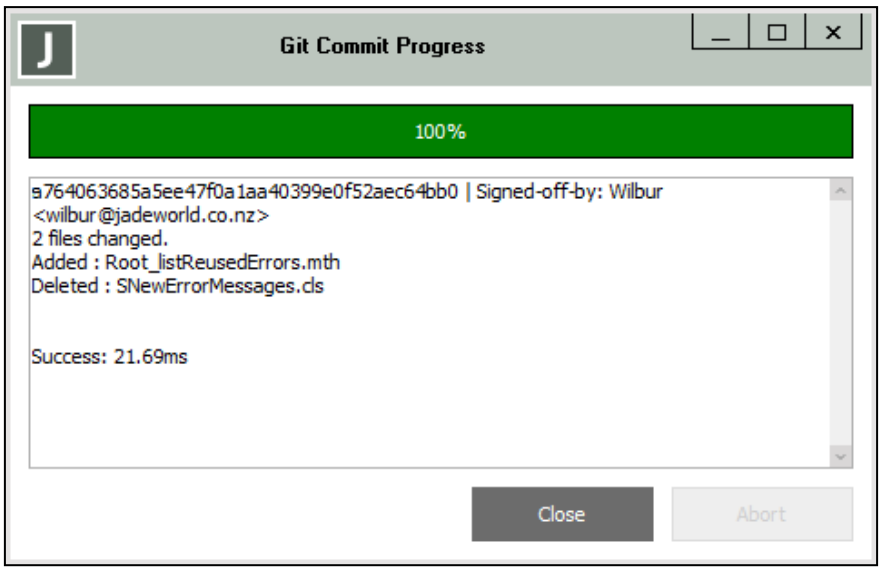
  Alternatively, to unstage *all* staged changes:
  - Click the **Unstage All** button.
  - Right-click on a row in the table and then select the **Unstage All Files** command from the popup menu.

3. In the **Commit Message** text box, click the optional **Add Sign-off-by** link if you want your configured Git name and e-mail address recorded in the text of the commit message.
4. In the **Commit Message** text box, enter the message that describes the changes or additions you are committing. You must specify a commit message.

   If a commit template is specified within any of the Git configuration files, the text of the template is automatically added to the **Commit Message** text box.
5. To commit your staged changes to your local repository HEAD, click the **Commit** button.

   Alternatively, click the **Close** button on the Git Commit dialog to abandon your commit actions. (Note that any staging or unstaging changes that you make while the dialog is open persist, even if you click the **Close** button.)

   An example of the Git Commit Progress form is shown in the following image. (The first value is the unique hash (SHA) for the commit.)



Applies to Version: 2018.0.01 and higher

Source Control Staging Categories and Statuses

The file change categories that can be displayed in the **Changes** table and the status that can be displayed are as follows.

- **Staged Changes**

  A staged file has been modified since the last commit and this change has been recorded in the repository index (that is, it has been staged). All staged changes become part of the Git history after the next commit and they are no longer displayed in the Git Commit dialog. Staged files have one of the following statuses, depending on the change that has taken place.

  - Added - a new file has been added to the repository index. It is unknown to the local repository HEAD; that is, it has not previously been committed.
  - Modified - a new version of a file has been added to the repository index. A previous version of the file exists in the local repository HEAD.
  - Deleted - the deletion of a file has been promoted from the working directory to the local repository index. A previous version of the file exists in the repository HEAD.
  - Renamed - the renaming of a file has been promoted from the working directory to the local repository index. A previous version of the file exists in the repository HEAD.
  - Type changed - a change in type for a file has been promoted from the working directory to the local repository index. A previous version of the file exists in the repository HEAD.

- **Unstaged Changes**

  An unstaged file has previously been recorded in the Git repository but has changes that have not yet been staged to the local repository index. Staging an unstaged file records the change into the local repository index and promotes the file to the **Staged Changes** type. Unstaged files have one of the following statuses, depending on the change that has taken place.

  - Modified - the file has been modified in the working directory. A previous version of the file exists in the repository index.
  - Missing - the file is missing from the working directory. It could have been moved or deleted. A previous version of the file exists in the index.
  - Renamed - the file has been renamed in the working directory. A previous version of the file exists in the repository index.
  - Type changed - the file type has been changed in the working directory. A previous version of the file exists in the repository index.

- **Untracked Files**

  An untracked file exists locally but is not a part of the Git local repository, as it has never been staged. Unless an untracked file has been added to a Git ignore list, it is displayed in the **Untracked Files** type in the table in the Git Commit dialog. Staging an untracked change records the change in the local repository index and promotes the file to the **Staged Changes** type. Untracked files have the following status.

  - Untracked - a new file in the working directory that is unknown to the local repository index or HEAD.

The following files are *not* displayed in the Git Commit dialog.

- Unaltered - the file has not been modified.
- Ignored - the file is untracked but its name or path matches an exclude pattern in the repository .**gitignore** file.

- Conflicted - the file has a conflict resulting from a merge action. These cannot be committed while conflicts exist.

Applies to Version: 2018.0.01 and higher

Pushing Source Changes to the Remote Repository

When you have staged and subsequently committed all of your changes to your *local* repository, you can then push the changes to the *remote* repository. Changes are pushed from the current checked out branch to the corresponding tracked remote branch, or if there is no corresponding tracked branch, you can select a remote repository and specify a branch name that will be created on the selected remote repository. Generally, the remote branch name should match the local branch name.

Any Git user can create a repository on a server, create branches, and then make any branch the *default*; for example, a development team may make **develop** the default branch. *HEAD* refers to the currently checked-out branch.

Note Jade supports only pushing to a bare repository; otherwise exception 8604 (*Operation requiring a working directory is performed against a bare repository*) is raised.

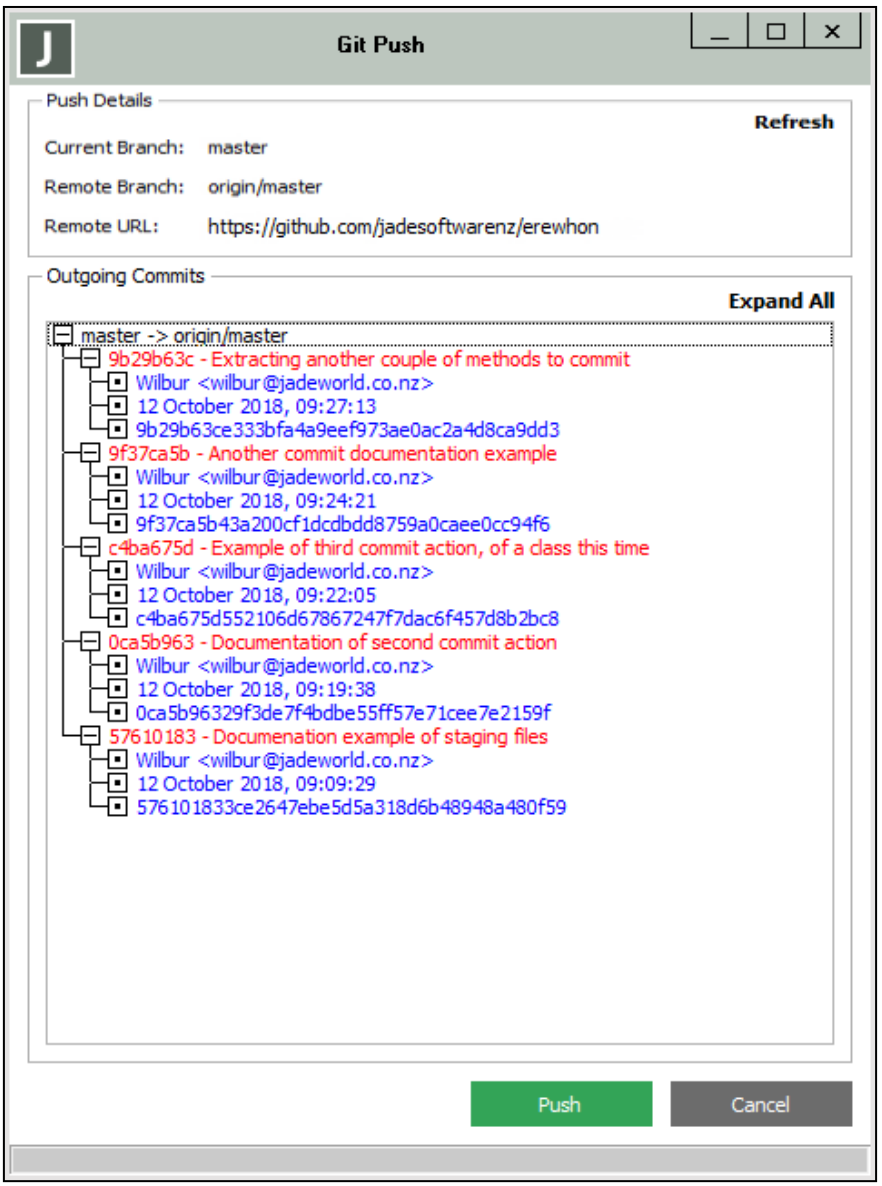
Although Jade does not currently provide the functionality to reverse a commit after it has been pushed, Git has a **revert** command, which you call on a specific commit or all commits, to reverse that change or those changes. You could then push this new commit so that all other members of your team pick up the reverted changes. (For details, see https://git-scm.com/docs/git-revert.)

Applies to Version: 2018.0.01 and higher

To push committed changes from the local HEAD to the corresponding remote repository branch

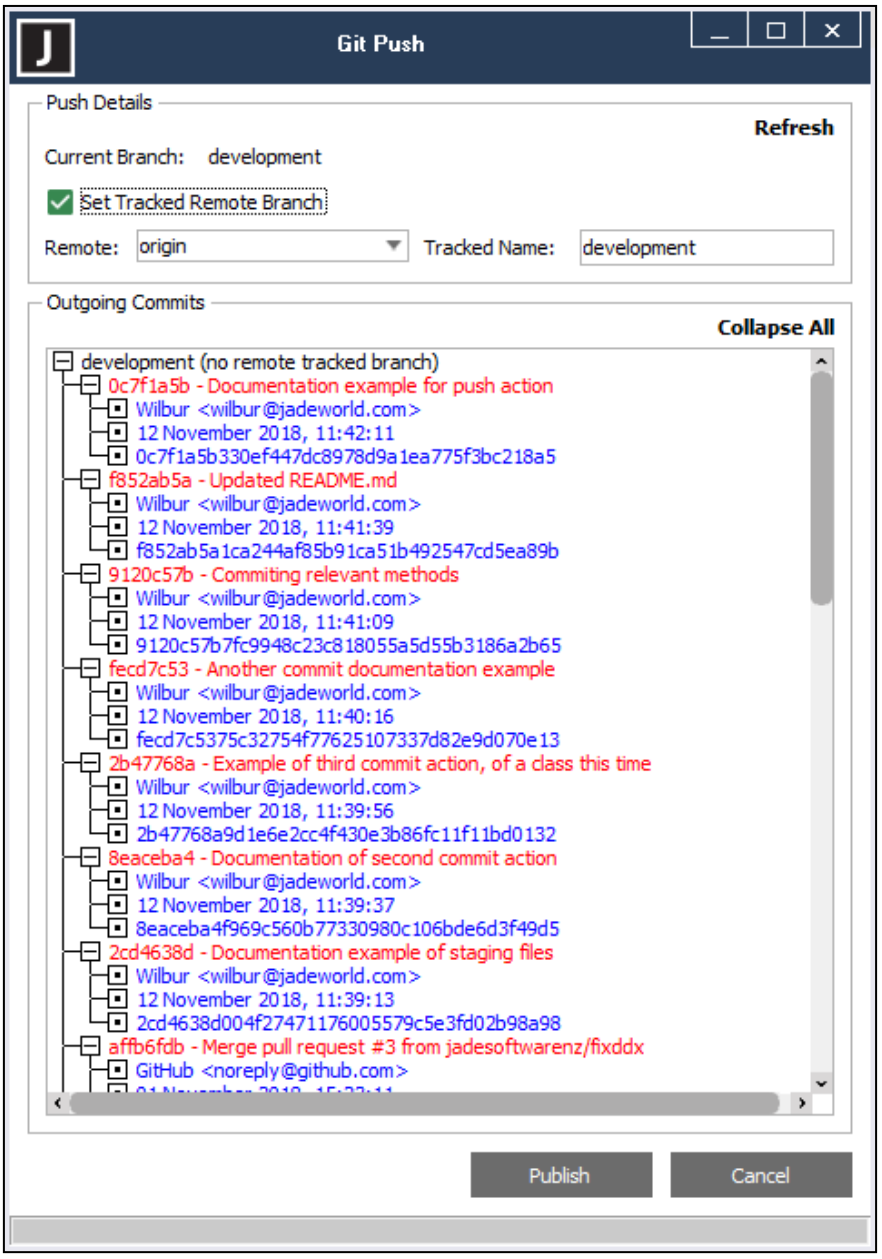
1. Select the **Push** command from the **Git Source Control Client** submenu in the Browse menu.

   The Git Push dialog, shown in the following image, is then displayed.



   If you have actioned other Git operations, click **Refresh** if you want to update the display of your committed changes.

The Git Push dialog, shown in the following image, is displayed if there is no corresponding tracked branch.



To publish your committed changes:

a. Check the **Set Tracked Remote Branch** check box, which is unchecked by default.

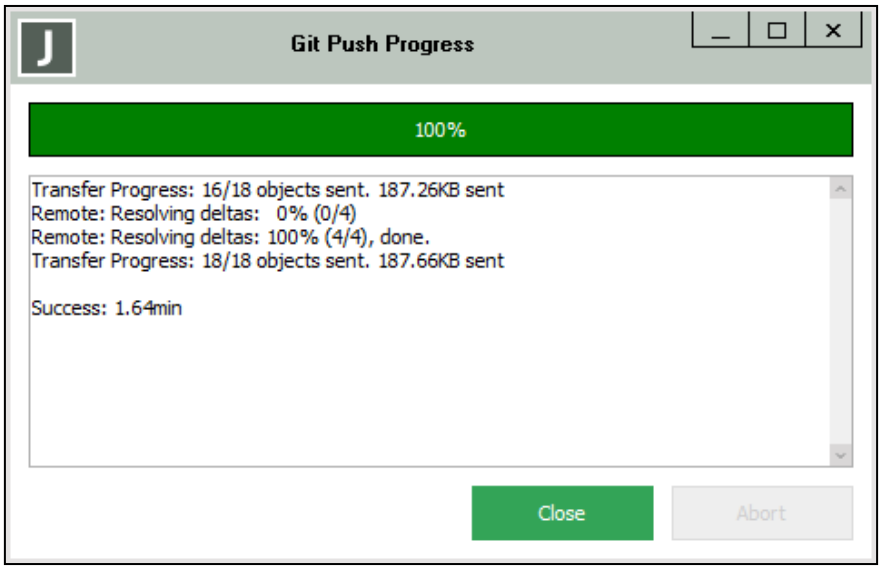
   The **Remote** combo box and **Tracked Name** text boxes are then displayed, and the **Publish** button replaces the **Push** button.

b. In the **Remote** combo box. select or specify the remote repository.

c. In the **Tracked Name** text box, specify the name of the tracked branch that will be created on the selected remote repository. (The remote branch name generally matches the local branch name.)

2. To push your committed changes from your current local repository to the corresponding remote repository tracked branch, click the **Push** button, or the **Publish** button if the local HEAD does not have a tracked branch.

   An example of the Git Push Progress form is shown in the following image. The progress form displays various status messages during the push operation, and incrementally updates the progress bar.



   Alternatively, click the **Abort** button to abandon the pushing of your committed changes to the remote repository. Click the **Close** button when the push operation is complete.

Pulling Commits from the Remote Repository

The **Pull** command incorporates changes from a remote repository into your working directory; that is, for your current branch.

A pull operation is a fetch followed by a merge. Fetching and merging enables you to view the commits that are to be merged before they are merged. For details, see https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests and https://git-scm.com/docs/git-pull.

If the merge or pull operation does *not* result in a merge conflict, the merge or pull operation is complete. If the operation *does* result in merge conflicts, the related progress dialog displays a merge conflict error.

Note If a merge conflict is detected, you should close the Git Pull dialog, resolve the conflict, and commit the result before attempting the merge or pull operation again, because when a repository is in a merge conflict state, attempting to pull or merge again results in an error stating that you should fix your files first.

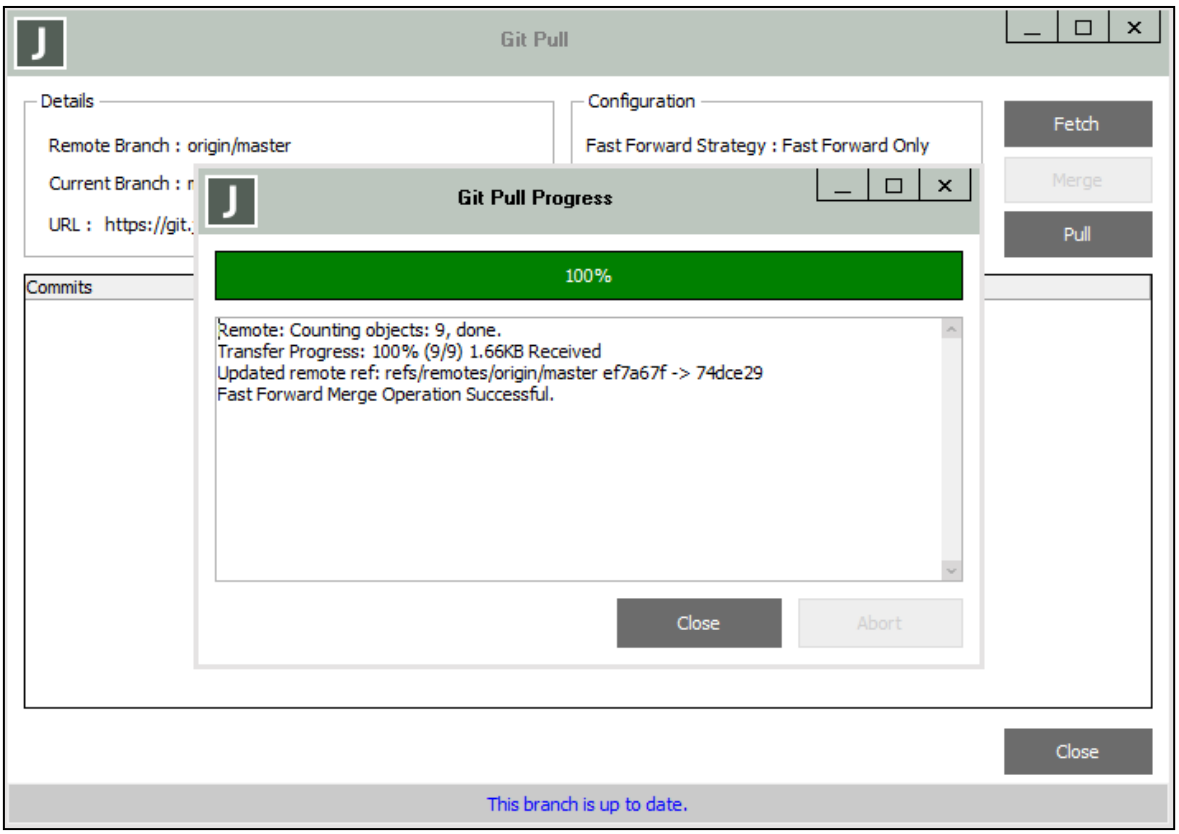
It is your responsibility to resolve any merge conflict.

If you decide that you do not want to incorporate the changes from the remote branch into your local branch, you can close the Git Pull dialog without merging.

To pull committed changes from the remote repository into your current branch

1. Select the **Pull** command from the **Git Source Control Client** submenu in the Browse menu.

   The Git Pull dialog is then displayed, as shown in the following image.



2. If you want a merge operation to abort as soon as any conflicts are detected, check the **Abort merge on conflict** check box in the Configuration group box, which is unchecked by default. You can then continue working and resolve the conflicted merge at a time of your choosing.

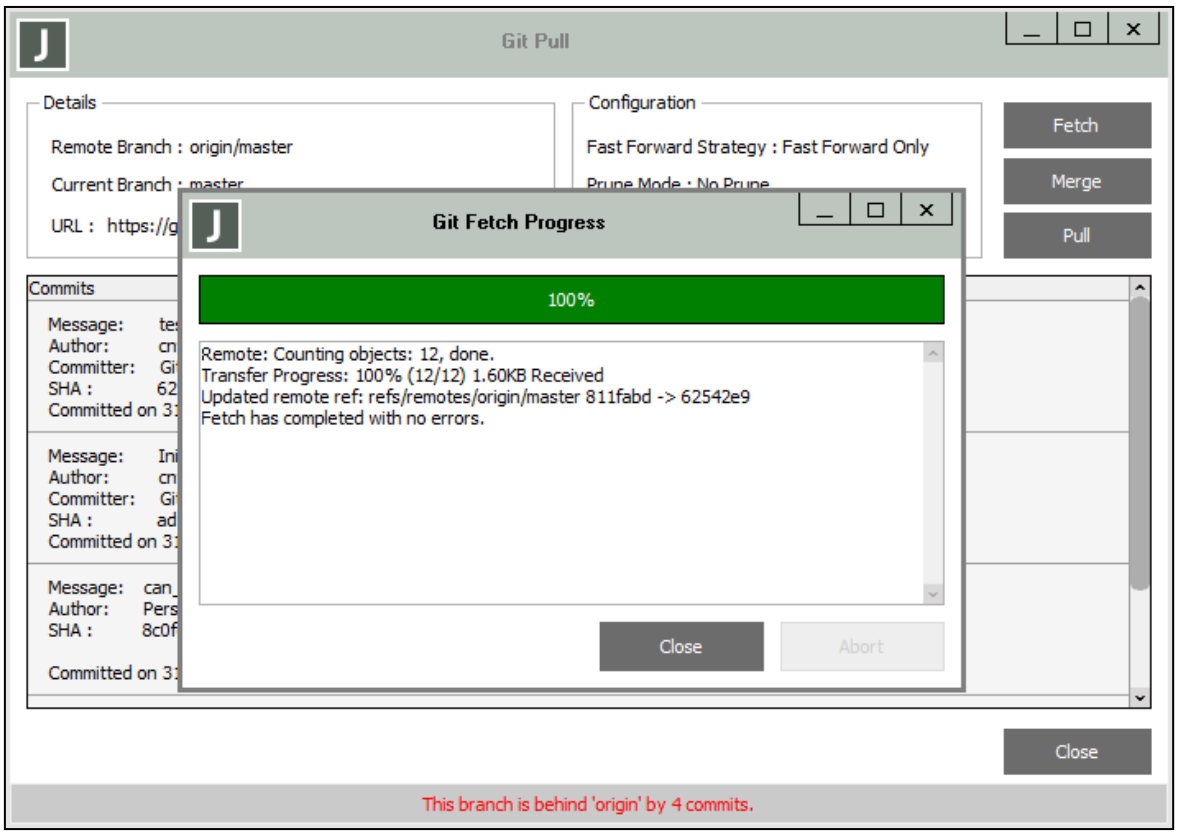
   When this check box is unchecked, conflict markers are added to the merged files for each conflict detected, and the fetch and merge operations are completed.

   Note Merge conflicts are not handled by Jade but have a conflict marker applied. The Git Pull Progress dialog advises you if there are conflicts, so that you can make the appropriate changes in your Jade source in your local repository or use the Git merge command line functionality to do so.

   For details, see https://git-scm.com/docs/git-merge.

3. To pull the changes that other team members have committed from the remote repository to your local repository, click the **Fetch** button and then merge the fetched commits, or click the **Pull** button to do a fetch and a merge operation. During a merge operation, conflicts may be detected that will require you to manually resolve the issue.

An example of the fetch process on the Git Pull Progress form is shown in the following image. The progress form displays various status messages during the transfer operation, and incrementally updates the progress bar.



Alternatively, click the **Abort** button to abandon the transferring of committed changes from the configured tracked branch to your local repository HEAD.

Click the **Close** button when the transfer is complete.

Checking Out a Source Control Branch or Tag

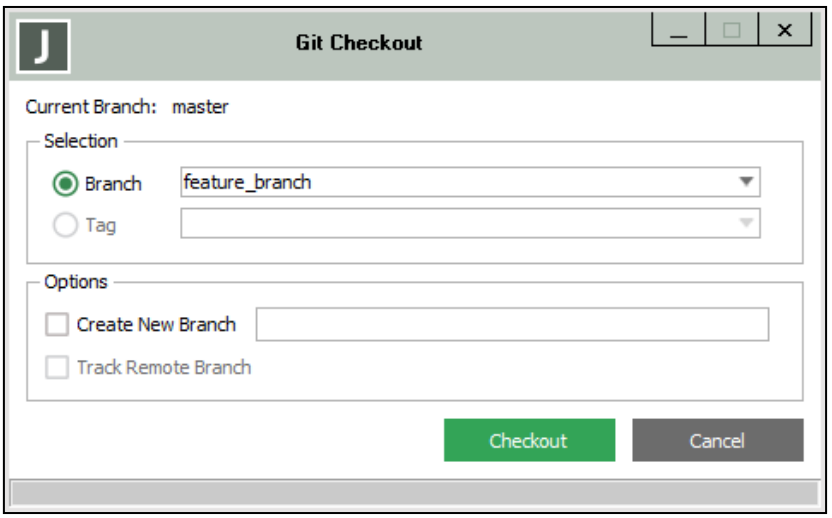
The checkout operation is used to switch between different revisions on your local repository. This process updates the files in the working tree and repository index to match the version of the specified branch or tag.

If a local branch is checked out, the checkout operation also updates HEAD, to set the specified branch as the current branch. Local modifications to the files in the working tree are kept, so that they can be committed to the checkout branch.

To check out a local branch

1. Select the **Checkout** command from the **Git Source Control Client** submenu in the Browse menu.

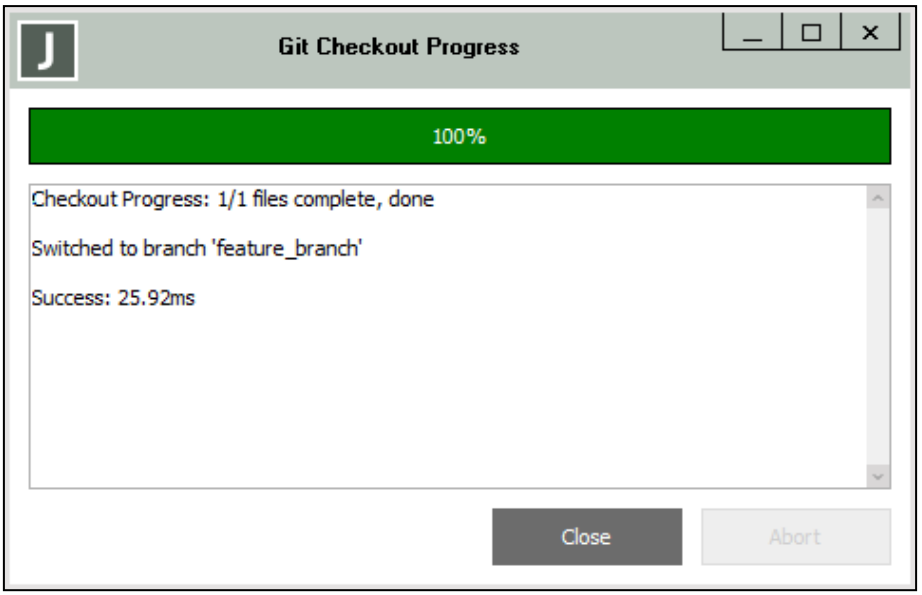
   The Git Checkout dialog, shown in the following image, is then displayed.



2. Select the required local branch in the **Branch** combo box. This combo box includes both local and remote branches. You can identify remote branches by the remote name prefix that they include.

3. To perform the checkout operation, click the **Checkout** button. (Alternatively, click the **Cancel** button to abandon the checkout operation.)

4. An example of the Git Checkout Progress form is shown in the following image. The progress form displays various status messages during the checkout operation, and incrementally updates the progress bar.



   Alternatively, click the **Abort** button to abandon the checkout operation.

5. Click the **Close** button when the checkout operation is complete.

Applies to Version: 2018.0.01 and higher

For details about checking out a remote branch or a tag, see the following subsections.

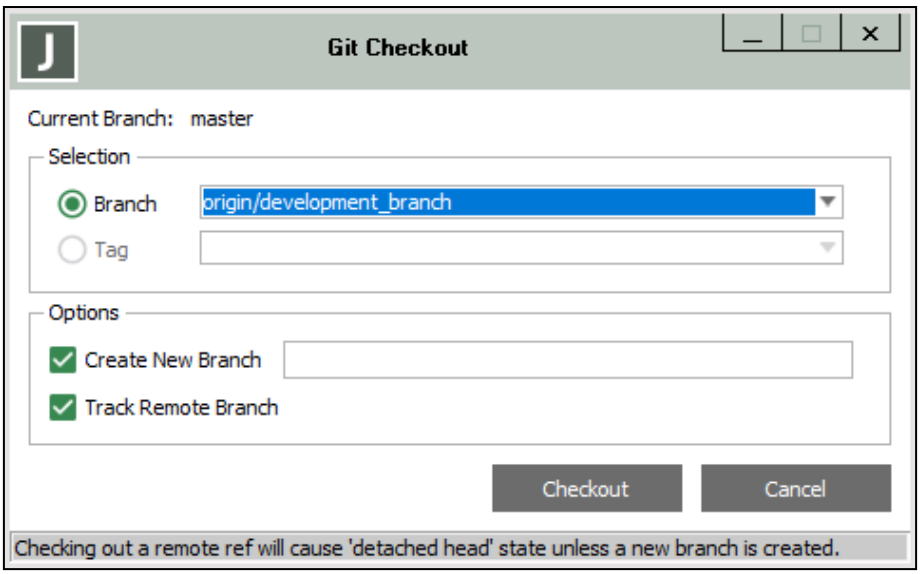
- Checking Out a Remote Branch
- Checking Out a Tag

Checking Out a Remote Branch

To check out a remote branch

1. Select the **Checkout** command from the **Git Source Control Client** submenu in the Browse menu.
2. Select the required remote branch in the **Branch** combo box. Remote branches are identified by the remote name prefix followed by their branch name.

   When you have selected a remote branch in the **Branch** combo box, the **Create New Branch** and **Track Remote Branch** check boxes are both automatically checked, as shown in the following image.



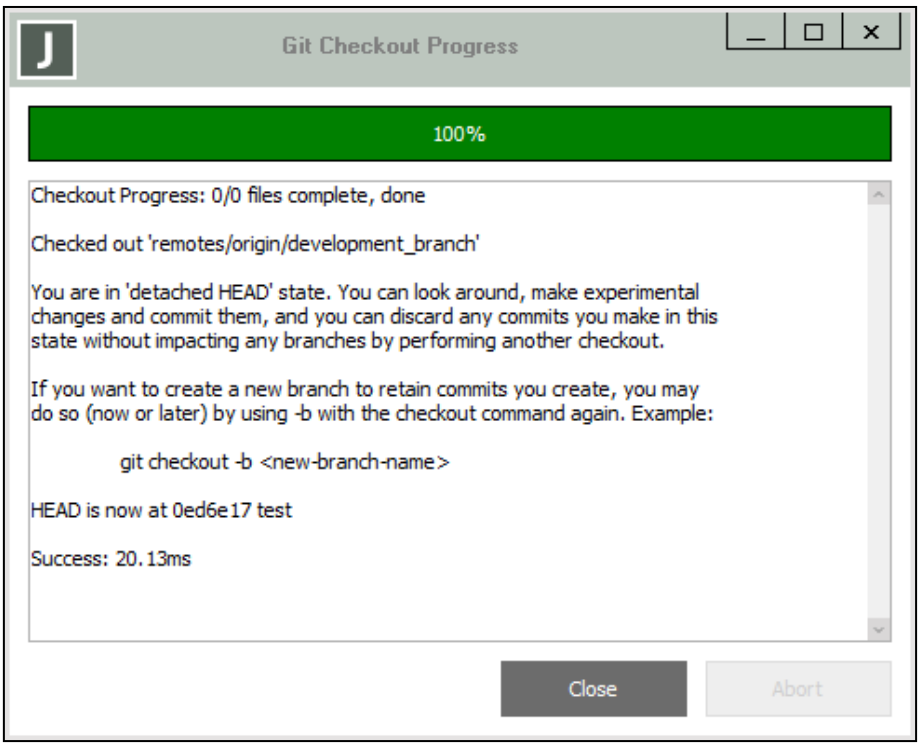
Note These check boxes are automatically checked because when checking out a remote branch, you should create a new local branch. If a new local branch is *not* created, a detached HEAD state occurs.

When in detached HEAD state, there is no current branch; that is, you are working at *no branch*. (For details, see https://git-scm.com/docs/git-checkout#_detached_head.)

3. If you are creating a new local branch, you must specify a branch name in the text box at the right of the **Create New Branch** check box. Generally, this name should match that of your selected remote branch.
4. When the **Track Remote Branch** check box is checked in conjunction with the **Create New Branch** check box, the new branch created during the checkout operation has its tracked remote branch set to the remote branch selected in the **Branch** combo box.
5. To perform the checkout operation, click the **Checkout** button. (Alternatively, click the **Cancel** button on the Git Checkout dialog, to abandon the checkout operation.)

If the **Create New Branch** check box is checked and a new local branch specified, a new local branch is created with the specified name and its tip matches that of the selected remote branch. This new branch is then checked out into the index and working directory.

If a remote branch is checked out without creating a new branch, a message box like that shown in the following image is displayed.



Applies to Version: 2018.0.01 and higher

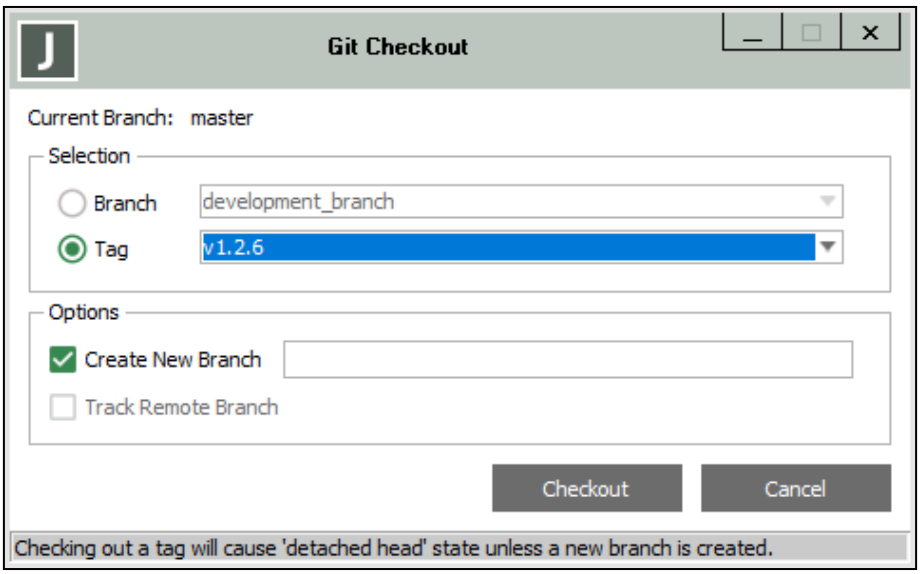
Checking Out a Tag

A *tag* marks a specific point in Git history such as a release; for example, **v1.0**.

To checkout a tag

1. Select the **Checkout** command from the **Git Source Control Client** submenu in the Browse menu.
2. Select the **Tag** option button and the required tag from the combo box at the right of the **Tag** option button.

When you have selected a tag, the **Create New Branch** check box is automatically checked, as shown in the following image.



Note This check box is automatically checked because when checking out a tag, you should create a new local branch. If a new local branch is *not* created, a detached HEAD state occurs.

When in detached HEAD state, there is no current branch; that is, you are working at *no branch*. (For details, see https://git-scm.com/docs/git-checkout#_detached_head.)

3. If you are creating a new local branch, you must specify a branch name in the text box at the right of the **Create New Branch** check box.

4. To perform the checkout operation, click the **Checkout** button. (Alternatively, click the **Cancel** button on the Git Checkout dialog, to abandon the checkout operation.)

If the **Create New Branch** check box is checked and a new local branch specified, a new local branch is created with the specified name and its tip matches that of the selected tag. This new branch is then checked out into the index and working directory.

Applies to Version: 2018.0.01 and higher

Navigating around the Jade Platform Development Environment

Jade provides the hierarchy nodes, toolbar buttons, and menus described in the following subsections, to enable you to navigate around the Jade Platform development environment. The Jade Platform development environment contains browser windows that provide a hierarchical structure of the browser elements.

The Schema Browser is always opened on start-up.

You can access the browser windows listed in the following table from Browse menu commands.

Command	Views and Maintains...
Ad Hoc Index	ODBC query indexes suitable for optimizing ad hoc queries without requiring database reorganization (for details, see "Maintaining Ad Hoc Indexes", in Chapter 9)
Applications	Applications in your schema (including running an application)
Breakpoints	All breakpoints that have been set in all methods in the current schema
Changed Methods	All methods that have changed within specified criteria
Checked Out Methods	All methods that have been checked out within specified criteria
Classes	Classes and their associated methods, properties, and constants
Classes In Use	All classes in use by all processes, nodes, and applications in non-production development environment
Deltas	Deltas enabling developers working in different parts of Jade to lock methods
Display Version Info	Entities that are versioned in all schemas (for details, see "Displaying Versioned Entities in All Schemas", in Chapter 3 of the *Developer's Reference*)
Exposures (C# or web services)	Jade objects exported through a C# or web service exposure library (for details, see Chapter 17 in this document, or "Defining a Web Service Provider Application", in Chapter 11 of the *Developer's Reference*, respectively)
External Component Libraries	.NET, ActiveX control, and ActiveX automation objects (for details, see Chapter 16, "Importing External Components")
External Databases	External database schema definitions
External Functions	External functions exported from a Dynamic Link Library (DLL)
Git Source Control Client	Source control (for details, see "Development Source Control", earlier in this chapter)
Global Constants	Global constants and their categories
HTML Documents	HTML documents
Interfaces	Jade interfaces and their associated methods and constants
Libraries	Libraries of external user-written Application Programming Interface (API) calls
Maps	Class map files
Methods Viewer	Method views (for details, see "Using Method Views to Bookmark Workflows", in Chapter 13)

Command	Views and Maintains...
Packages	Export packages of classes, properties, methods, and constants that are available to other schemas and import packages that have been imported from other schemas (for details, see "Using Packages", in Chapter 8 of the *Developer's Reference*)
Patches	Patch control information (for details, see "Maintaining Patch Numbers", in Chapter 3 of the *Development Environment Administration Guide*)
Primitive Types	Primitive types and their associated methods
Relational Views	Relational view of the current schema
RPS Mappings	Relational Population Service (RPS) mappings of the current schema
Schema Views	Customized Class Browser displaying only the required classes
Status List	All methods and class constants that match the specified status criteria
Status List for Current User	All methods and class constants that match the specified status criteria for the current user
Unreferenced Methods	All methods that are not referenced by other methods
Versioned Methods	All versioned methods in the selected object
Web Service Consumer	Web service consumers (for details, see "Defining a Web Service Consumer Application", in Chapter 11 of the *Developer's Reference*)

You can also access some browsers from toolbar buttons or by using shortcut keys. For more details, see "Using Browser Toolbar Buttons" or "Using Function Keys and Shortcut Keys", later in this chapter.

The Schema Browser highlights an incomplete schema with a default background color of red. If you select an incomplete schema in the Schema Browser, the Schema, Browse, and Jade menus are disabled. If you have selected an incomplete schema and you then attempt to open a Class, Primitive Type, Map, or Application Browser by clicking on the relevant toolbar button, a message is displayed, advising you that the schema is incomplete.

Tips To activate an open window, select the appropriate window from the list in the Window menu. Alternatively, you can use the Ctrl+F6 shortcut keys to cycle through all open MDI child windows.

When a delta is set, the delta text display on the right of the Jade Platform development environment toolbar is drawn in red. When you double-click on this delta text description, the Delta Browser is displayed.

Right-click on an item in a browser to quickly display a popup menu that provides choices for that item. (A popup menu contains the same commands as the appropriate menu in the menu bar. For example, a popup Methods menu is displayed at the caret position when you right-click on a method in the Methods List of the Class Browser.) You can use the Esc key to cancel the display of popup menus.

Left-clicking on a menu item that has a submenu immediately opens the submenu instead of waiting for the menu hover timer to expire before the submenu is displayed.

If you do not want forms to be displayed in the default standard Multiple-Document Interface (MDI) style, select the required option button in the Mdi group box on the **Browser** sheet of the Preferences or Jade Installation Preferences dialog. When you select:

- **Use Mdi**, standard MDI forms are displayed (the default value).
- **Use Mdi With Tabs**, a line of tabs is displayed above the MDI client window. Each tab is associated with a displayed MDI child form and displays the form's caption. Clicking on that tab brings the associated form to the front. The MDI child forms can still be maximized, restored, and minimized within the MDI client area.

- **Use Tabs Only**, a line of tabs is displayed above the MDI client window. Each tab is associated with a displayed form and displays the form's caption. Clicking on that tab brings the associated form to the front. The forms are always maximized and cannot be restored or minimized. The parent of these forms is the associated tab sheet; *not* the MDI client window.

When the MDI option on the Preferences dialog **Browser** sheet is set to **Use Mdi With Tabs** or to **Use Tabs Only**, you can close a tab by clicking the middle mouse button (that is, pressing the mouse wheel) on the tab. (The middle mouse button does not close pinned tabs.)

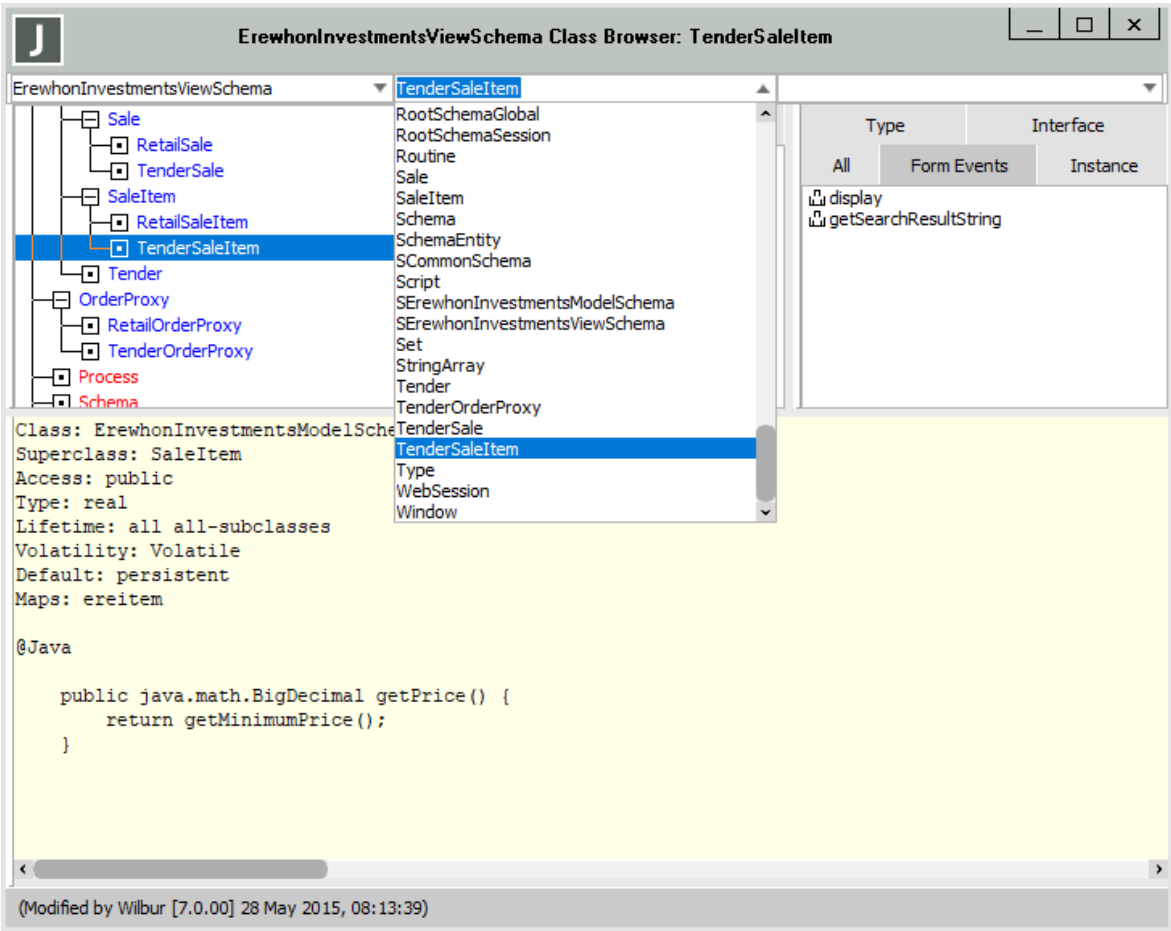
Notes This affects only the main development environment tabs.

You can still close a tab by clicking the close icon ☒ at the top right corner of the tab, including pinned tabs.

For details about specifying your browser preferences and how Jade handles the MDI with tabs and the tabs-only styles, see "Maintaining Browser Options", later in this chapter.

Using the Navigation Bar in Hierarchy Browsers

Each hierarchy browser (for example, the Class Browser) can display a navigation bar (shown in the following image), with drop-down combo boxes from which you can select a schema, a class, and a method.



Selecting a schema, class, or method refreshes the current browser with the selected entity; for example, selecting a schema refreshes the display to that schema (as though a new browser was opened) and selecting a class selects the class (as though a class was selected using F4).

Tip For details about filtering the entries displayed in list boxes associated with the navigation combo boxes, see "Filtering Entries in Combo Box and List Box Text", elsewhere in this chapter.

The display is updated when the combo box is closed and a list entry has been selected. This enables you to use the arrow keys to scroll through the entries and enter text to refine the list before clicking the required entry or pressing Enter.

The navigation bar is hidden, by default. Toggle the display of the navigation bar by checking or unchecking the **Show Navigation Bar** check box on the **Browser** sheet of the Preferences dialog. The setting of this check box is saved in your user profile and can be extracted to a preferences initialization file by using the existing Export Preferences functionality provided on the **Miscellaneous** sheet of the Preferences dialog. You can also toggle the navigation bar display by using the View menu **Show Navigation Bar** command.

The navigation bar combo boxes, from left to right, are as follows.

- When the combo box on the left of the browser is dropped down, it lists the name of every schema defined in your system, including versioned schemas (which are displayed with the versioned background color of your user preference).

  From version 2020.0.01 and higher, changing the schema using the navigation bar combo box in the Class List of a Class Browser to navigate to the parent class in a superschema does not select the **Object** class in the superschema by default. The selected class is the first class previously selected in the class hierarchy that is common to both schemas; that is, if a:

  - Superschema is selected, the class selected will be the superclass of the previously selected class.
  - Subschema is selected, the previously selected class will again be selected in that schema.

  The text box portion of the combo box usually displays the name of currently selected schema.

- When the middle combo box is dropped down, it lists in alphabetical order the name of every class defined in the selected schema.

  Selecting a class in the list is equivalent to clicking on the class in the Class List of the browser.

  The text box portion of the combo box usually displays the name of the currently selected class.

  The list does not include any superschema classes not marked as copy classes in the currently selected schema. To access such classes, press F4.

- When the combo box at the right is dropped down, it lists in alphabetical order the name of every method in the currently selected class.

  When the class is a form, the list includes all event methods defined for the form, menus, and all controls. The event methods display the full name (for example, **cmbCustomers_click**).

  This list does not include the names of any superclasses, regardless of the setting of the View menu **Show Inherited** command.

  Selecting a method in the list displays the selected method. If the method is in the currently displayed method list, that method entry is selected. If the method is *not* in the currently displayed list, the **All** folder of the Methods List is displayed and the method is selected in the methods list of the combo box.

  If the method is an event method, the associated form, control, or menu item in the Properties List is selected (and the **All** folder in the Properties List is displayed).

  If the method is not an event method, the properties selection is unaffected (that is, the **<methods>** entry is no longer displayed in the Properties List).

If one of the combo box entries is selected and the current method display has been changed but not saved, you are prompted to save or discard the changes, or to cancel the action.

Each combo box enables you to enter text. Changing the combo box text causes the display of only entries that contain the specified text somewhere in the entity name.

The search process handles uppercase searches if all text entered after the first character is uppercase. The list then includes all entries in which the first character of the entity name matches the first entered character and the entity name has uppercase characters that match the order of the remaining entered uppercase text. For example, entering **sNT** in the method combo box at the right of the navigation bar include method names **searchNameTextOnly** and **someNumberedTest**.

Note Because the same form can be used to access different schemas, the browser history in the displayed History menu includes the schema name if the entry is not for the currently selected schema.

If the accessed schema was versioned, any history menu list entry for the versioned schema (for example, the schema or a class name) is preceded by an asterisk (*) character.

Applies to Version: 2018.0.01 and higher

Using Hierarchy Nodes to Navigate around a Browser Window

Browser windows provide:

- A graphical picture of the selected schema element hierarchy
- A collection of hierarchy nodes representing selected schema elements; for example, a class

Notes In this section, the term *node* refers only to the hierarchy node that represents a schema element. It does *not* refer to the client/server term node, which is a physical workstation participating in a particular application.

For details about dynamically customizing the layout of Hierarchy Browser forms, see "Customizing the Layout of Hierarchy Browser Forms", later in this chapter.

Nodes can be expanded to show all subitems or they can be collapsed to show a condensed version of the element. An expanded node is displayed with a minus sign (**-**). A collapsed node that has subitems displays a plus sign (**+**). A node that has no subitems has no sign.

To collapse an expanded node, perform one of the following actions

- Click on the node.
- Select the **Expand Branch** command from the View menu. For details, see "Expanding a Node Branch", in the following subsection.

To expand a collapsed node

- Click on the node.

To navigate using nodes

1. Click on the schema element or its subitem in the browser window.
2. Select the required function for the selected element; for example, select the **New Jade Method** command from the Methods menu when you select a class in the Class Browser.

The Class Browser enables you to use the right arrow (→) key to expand the current class (display subclasses in the next hierarchy level of the selected class) or the left arrow (←) key to collapse (hide all levels of subclasses in the selected class) the current class.

In addition, you can perform these actions for any class that has subclasses, by using the plus key (**+**) or minus key (**-**) on the numeric keypad. Alternatively, use the asterisk key (*****) to expand the current class to display all levels of subclasses in the selected class.

Expanding a Node Branch

Use the View menu **Expand Branch** command from the Class Browser or Schema Browser to expand the class or schema node that is currently selected. By default, the **Object** class is selected when a Class Browser is opened. The **Object** node is expanded (that is, its subclasses are displayed), while all other nodes (that is, subclasses of **Object** class subclasses) are collapsed.

» To expand the selected node

- Select the **Expand Branch** command from the View menu.

The selected node is then expanded, and all levels of subclasses in the selected class or subschemas in the selected schema are displayed. Each node that has children has a minus character (-) displayed in its node, indicating that the branch is expanded.

» To collapse an expanded node

- Click the expanded node icon for the class or schema that you want to collapse.

The selected node is then collapsed, and none of its subclasses or subschemas is displayed. The node then displays a plus character (+), indicating that the branch is collapsed.

Using Browser Toolbar Buttons

A toolbar is displayed for all browse windows in the Jade Platform development environment. In addition, the right of the Browser toolbar contains the Editor Clipboard toolbar, to enhance your use of the internal Jade editor clipboards. (For details, see "Using the Editor Clipboard Toolbar", later in this chapter.)

For details about the Jade Painter toolbars that provide painter functions, see "Painter Toolbars", in Chapter 5.

Use Browser toolbar buttons to:

- Quickly access commonly used functions

  Tip When you hover the mouse over the icon on a toolbar button, the bubble help that displays the icon function also displays the equivalent menu shortcut in the displayed bubble help text as *bubble-help-description* [*shortcut-keys*]; for example, **Browse Classes [Ctrl+B]**. (The shortcut key that is displayed for an action is set on the **Short Cut Keys** sheet of the Preferences dialog.)

  The development environment **Reorg**, **Context Help**, and **General Help** toolbar icons do not have an equivalent menu shortcut, so there is no shortcut displayed in the bubble help.

- Perform general administrative functions
- Navigate around the Jade Platform development environment

» To use a toolbar button

- Click the appropriate button

Toolbar buttons that are not available for selection are disabled (dimmed). For example, if you select a class in the Class Browser window, the **Save** toolbar button is disabled and cannot be selected.

Note The toolbar is displayed by default. To hide the toolbar display, uncheck the **Show Tool Bar** check box in the **Window** sheet of the Preferences dialog, accessed from the **Preferences** command in the Options menu of browser windows.

You can reposition the toolbar vertically or horizontally, by clicking on the grip bar and dragging the toolbar to the required position on the background form or causing it to float in an independent window.

The View menu in a Hierarchy Browser enables you to change the size of browser toolbar icons. For details, see "Changing the Size of Toolbar Icons", in the following subsection.

You can drag one or more files from the Windows Explorer onto the:

- **New Workspace** or **Open Workspace** toolbar button to open a new workspace for each file that is dropped. (Any folders that are dropped are ignored.)
- **Load** toolbar button to open the Load Options dialog and populate the **Schema File Name** text box with the file or files. If you drop:
  - A single file with a .**mul** file type, the **Load Multiple Schemas** check box is checked.
  - More than one file, the **Load Multiple Files** check box is checked and all of the dropped file names are displayed in the **Schema File Name** text box, so that you can add more files, if required.

  Dropping a folder is rejected.

The functions of Browser toolbar buttons are listed in the following table.

Button	Description	For details, see…
	Opens a new Workspace window	New Workspace Toolbar Button
	Opens an existing Workspace window	Open Workspace Toolbar Button
	Saves the text in a Workspace or the editor pane	Save Toolbar Button
	Cuts (logically deletes) the selected portion of method	Cut Toolbar Button
	Copies the selected portion of a method or the contents of the current list window to the clipboard	Copy Toolbar Button
	Pastes a portion of a method from the clipboard to the current position	Paste Toolbar Button
	Prints selected objects	Print Selected Toolbar Button
	Opens a Class Browser window	Browse Classes Toolbar Button
	Opens a Primitive Types Browser window	Browse Primitive Types Toolbar Button
	Opens an Interfaces Browser window	Browse Interfaces Toolbar Button

Button	Description	For details, see…
	Opens a Class Maps Browser window	Browse Maps Toolbar Button
	Opens an Application Browser window	Browse Applications Toolbar Button
	Opens a Schema Browser window	Browse Schema Toolbar Button
	Runs a Jade runtime application	Run Application Toolbar Button
	Displays the Load Options dialog	Load Toolbar Button
	Displays the Extract dialog	Extract Toolbar Button
	Accesses the Painter	Painter Toolbar Button
	Displays the Jade Monitor Start-up dialog	Monitor Toolbar Button
	Sets or unsets a breakpoint on the current line of logic	Toggle Breakpoint Toolbar Button
	Creates and toggles bookmarks for current positions	Bookmark Toolbar Button
	Displays whether the current schema requires reorganization	Reorg Toolbar Button
	Views the previously accessed method, property, or type	History Back Toolbar Button
	Views the method, property, or type that was next accessed	History Next Toolbar Button
	Accesses context-sensitive online help for a specific element	Context Help Toolbar Button
	Displays the first page of the online help	General Help Toolbar Button

Changing the Size of Toolbar Icons

Use the View menu **Toolbar Icon Size** command from a Hierarchy Browser to change the size of icons on Jade Platform development environment toolbars. The size options are **Small** (16x16 pixels), **Medium** (32x32 pixels), and **Large** (48x48 pixels).

To change the size of toolbar icons

1. Select the **Toolbar Icon Size** command from the View menu. The icon sizes submenu is then displayed.
2. From the **Small Icons**, **Medium Icons**, and **Large Icons** commands, select the icon size that you require. A checkmark indicates the size that is currently selected.

The new icon size is immediately applied to all open relevant Hierarchy Browsers and open Painter forms.

Tip You can also change the toolbar icon size on the **Browser** sheet of the Preferences dialog. For details, see "Maintaining Browser Options", later in this chapter.

New Workspace Toolbar Button

Use the **New Workspace** button from the Browser toolbar to open a new Workspace window. The Workspace window consists of an editor pane, to enable you to write a Jade method that is not to form part of your application. For example, you can use the Workspace window to write and test a method and output results to the Jade Interpreter Output Viewer window or to test that the common File Open dialog to select a file is displayed.

Open Workspace Toolbar Button

Use the **Open Workspace** button from the Browser toolbar to open a disk file on a Workspace window. For more details, see "New Workspace Toolbar Button", in the previous subsection.

Save Toolbar Button

Use the **Save** button from the Browser toolbar to save the text in a Workspace window or a method editor pane.

Note Logic is not saved as you enter it.

Your logic is automatically saved when it is compiled, even when it contains errors. You can save logic that has not been compiled, or that has been compiled and contains errors. If you save a method that has not been compiled, the following message is displayed in the status line:

```
Method saved (not compiled)
```

Cut Toolbar Button

Use the **Cut** button from the Browser toolbar to logically delete a selected portion of a method in the current editor pane.

You can physically delete whole methods only by using the **Remove** command from the Methods menu. (For details, see "Removing a Schema Element", in Chapter 3.)

This button is disabled when there is no selected text to be cut.

Copy Toolbar Button

Use the **Copy** button from the Browser toolbar to copy a selected portion of a method or the contents of the current list window to the clipboard.

Note If focus is in a list window that has a hierarchical structure of nodes, only those entities that have been displayed since the browser window was opened are copied. For example, if focus is in a Class List and no collapsed nodes have been expanded, only the highest level classes are copied. Conversely, if one or more nodes has been expanded and subsequently collapsed while the browser window had focus, all lower-level entities that have been displayed are copied to the clipboard.

This button is disabled when no text is selected in the editor pane or a list window does not have focus.

Paste Toolbar Button

Use the **Paste** button from the Browser toolbar to paste a selected portion of a method from the clipboard into the current method. The pasted portion can be a selection cut or copied from the current method, or from another method. This button is disabled when there is no selected portion of a method in the clipboard.

Note You can paste a portion of a method to the same method or a different method, or a complete method to a different method.

Print Selected Toolbar Button

Use the **Print Selected** button from the Browser toolbar to print selected elements in the Jade Platform development environment. The **Print Selected** toolbar button is enabled only from the Class Browser, Primitive Types Browser, Interface Browser, Global Constants Browser, and the Summary of Patches window; that is, you cannot select this button from the Schema Browser, Class Maps window, or Application Browser.

The current element determines the selective print options. For example, the Print Options dialog accessed from the Class Browser provides different options from that accessed from the Primitive Types Browser or Interface Browser.

Using the Print Options Dialog

Access the Print Options dialog by clicking the **Print Selected** icon in the Browser toolbar or selecting the **Print Selected** command from the File menu. The options provided by this dialog are determined by the element that is selected when you access the dialog.

You can select operations documentation (technical objects such as subclasses, methods, or references) for development purposes, and user documentation (design-related objects such as forms) for such things as end-user sign-off. For details about printing a selected schema element, see "Printing a Selected Schema Element", in Chapter 3.

Browse Classes Toolbar Button

Use the **Browse Classes** button from the Browser toolbar to open a Class Browser window.

Note More than one Class Browser for a schema can be open at any time.

You can also have concurrent open Class Browsers for different schemas in the current development environment session. (For details, see "Using Concurrent Windows", later in this chapter.)

Tip If you have set a Schema View, right-click on the **Browse Classes** toolbar button to display a Class Browser for that Schema View.

Browse Primitive Types Toolbar Button

Use the **Browse Primitive Types** button from the Browser toolbar to open a Primitive Types Browser window.

Note More than one Primitive Types Browser for a schema can be open at any time. You can also have concurrent open Primitive Types Browsers for different schemas in the current development environment session. (For details, see "Using Concurrent Windows", later in this chapter.)

Browse Interfaces Toolbar Button

Use the **Browse Interfaces** button from the Browser toolbar to open an Interface Browser window. (For details about this browser, see "Accessing the Interface Browser", in Chapter 14.)

Note More than one Interface Browser for a schema can be open at any time. You can also have concurrent open Interface Browsers for different schemas in the current development environment session. (For details, see "Using Concurrent Windows", later in this chapter.)

Applies to Version: 2018.0.01 and higher

Browse Maps Toolbar Button

Use the **Browse Maps** button from the Browser toolbar to open a Class Maps Browser window.

Notes Only one Class Maps Browser for the current schema can be open at any time. If a Class Maps Browser is already open for that schema, it is brought to the top when you click the **Browse Maps** button.

You can have concurrent open Class Maps Browsers for different schemas in the current development environment session. (For details, see "Using Concurrent Windows", later in this chapter.)

The map file for your application classes is specified when you define a new application.

Browse Applications Toolbar Button

Use the **Browse Applications** button from the Browser toolbar to open an Application Browser window. A Jade application is an end-user runtime system, defining a collection of forms and the location of the help file.

You can have one or many applications in each schema, each providing a different "view" of the object model, with different forms allowing data in that model to be displayed and possibly updated.

Notes Only one Application Browser for the current schema can be open at any time. If an Application Browser is already open, it is brought to the top when you click the **Browse Applications** button.

You can have concurrent open Application Browsers for different schemas in the current development environment session. (For details, see "Using Concurrent Windows", later in this chapter.)

Browse Schema Toolbar Button

Use the **Browse Schema** button from the Browser toolbar to access the Schema Browser. For example, you can access the Schema Browser to select and set a different schema or subschema to be the current schema.

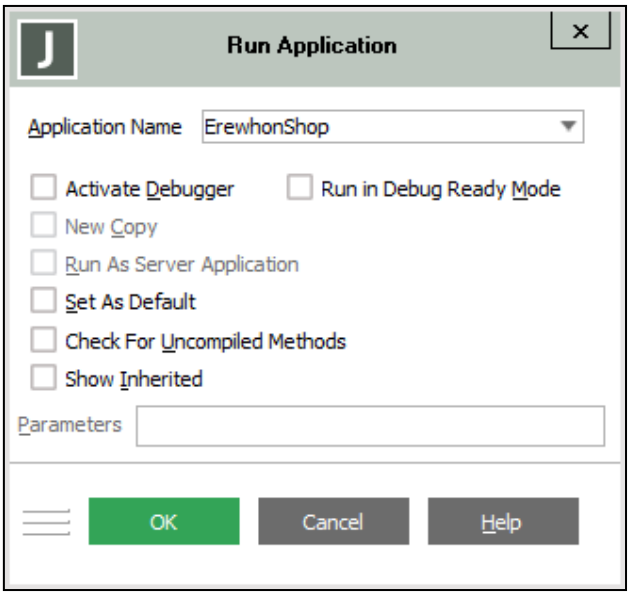
The appearance and functionality of applications in a schema can differ, but they all share the underlying object model defined by the schema.

Note The Schema Browser is the default background Jade browser, and is always opened on start-up.

Run Application Toolbar Button

Use the **Run Application** button from the Browser toolbar to run a Jade application.

The Run Application dialog, shown in the following image, is then displayed.



The **Run Application** toolbar button is disabled in browser windows of the latest schema context.

Tip To bypass the Run Application dialog, right-click on the **Run Application** toolbar button.

To run your Jade application

1. In the **Application Name** combo box, select the application that you want to run. The current application is displayed by default.

   If you want to run another application in the current schema, select the appropriate application in the **Application Name** list box. If no application is set, the first application is displayed.

   For details about running a superschema application from the current schema, see step 8 of this instruction.

2. Check the **Activate Debugger** check box if you want to activate the debugger for the application. By default, the debugger is not activated when an application is run.

3. Check the **Run in Debug Ready Mode** check box if you want to run the application in debug mode but without the debugger being initiated when this is checked. Attaching to that application then allows debugging of any method already on the execution stack. (For details, see "Attaching the Debugger to a Running Application", in Chapter 1 of the *Runtime Application Guide*.)

   Note If the application is idle and no modal dialogs are displayed, attaching the debugger is equivalent to running in debug-ready mode.

4. Check the **New Copy** check box if you want to run a new copy of the selected application when that application is currently running. The **New Copy** check box is disabled if the application is not already running.

5. Check the **Run As Server Application** check box if you are running a non-GUI application (defined as application type **Non-GUI** or **Web-Enabled Non-GUI** in the Define Application dialog) in multiuser mode and you want to run the application on the server node instead of your client node. For example, you can initiate a non-GUI application to run on the server node so that it continues running after you have shut down your client node.

This check box is enabled only when the selected application is of type **Non-GUI** or **Web-Enabled Non-GUI** and you are running Jade in multiuser mode. When you check this check box, the **Activate Debugger and New Copy** check boxes are disabled.

6. Check the **Set As Default** check box if you want to set an application in your schema as the current default application. (This check box enables you to set the default application without having to access the Application Browser to change the current application.)

   This check box is enabled only when the selected application is not the current application.

7. Check the **Check For Uncompiled Methods** check box if you want to be warned before the application is invoked about any methods in the application that are uncompiled or are being edited. This check box is unchecked by default; that is, methods are not checked before the application runs.

8. Check the **Show Inherited** check box if you want to run a superschema application from a subschema. The **Application Name** combo box is then populated with the applications from the current schema *and* its superschemas.

   Note If the default application is set to a superschema application on the Run Application dialog, that superschema application is the default application.

9. If the initialize method for the application has a single parameter of type **HugeStringArray**, the **Parameters** text box is enabled so that you can specify parameters for command line arguments when running an application from within Jade.

   Specify each entry in the array as a space-delimited token in the parameters string; for example:

   ```
   "schema=RootSchema app=JadeSchemaLoader ini=MyJade.ini
   schemaFile=d:\jade\scm\test.scm dontSaveSources=true
   loadStyle=onlyStructuralVersioning"
   ```

   Note It is up to the application being initiated to delete the shared transient **HugeStringArray** instance.

10. Click the **OK** button to run the selected application. (Alternatively, click the **Cancel** button to abandon your selections.)

    Your selected application is then initiated and the following actions are performed.

    - If the **New Copy** check box is unchecked and the application is currently active, focus is set to the active application.
    - If the **Set As Default** check box is checked, the Application Browser is updated to reflect the new current (default) application.
    - If the **Check for uncompiled methods** check box is checked and the application contains any methods that have not yet been compiled or that are locked for editing, a message box is then displayed.

      If you want to compile or save the specified method before running the application, click the **No** button and compile or save the method before you retry running the application. Alternatively, click the **Yes** button if you want to run the application with an uncompiled method or one that is locked for editing.

    When you have started your Jade application, the start-up form for that application is then displayed (for example, a log-in screen or a menu).

Load Toolbar Button

Use the **Load** button from the Browser toolbar to install (load) a full or partial schema from an extract file into your current schema; for example, to restore or reconstruct your Jade database.

When you want to restore or reconstruct your Jade database, for example, you can load (install) the file that you extracted. The schema (.**scm**) file is loaded before you load the form and data definition (.**ddb** or .**ddx**) file.

The **Load** toolbar button also enables you to:

- Restore a full or partial schema
- Obtain code from another user
- Fully reconstruct a schema in a compatible Jade release

For more details, see "Loading Your Schema", in Chapter 10.

Extract Toolbar Button

Use the **Extract** button from the Browser toolbar to extract (or save) the current schema from your current release to a file (for example, to back up your existing database).

Before you reorganize your database or you install a new release of Jade, you may want to first extract to a file any schema and forms. (Jade upgrades your schema and form definitions as part of the upgrade process, but you may want to take a back-up copy of your database before you upgrade to a new Jade release). You can extract a complete application, or you can extract forms from all applications in a schema.

Note Repeat this process for each schema under the **RootSchema** that you want to save to a file. Alternatively, you can use the multiple schema extract facility.

The **Extract** toolbar button also enables you to:

- Back up a full or partial schema
- Pass code to another user
- Fully reconstruct a schema in a compatible Jade release

For details, see "Extracting Your Schema", in Chapter 10.

Painter Toolbar Button

Use the **Painter** button from the Browser toolbar to create and maintain screen and printer forms for runtime Jade applications.

The Painter is a graphical user interface (GUI) painter that supports bitmaps, list boxes, button controls, and so on. For more details, see Chapter 5.

Note Only one Painter window for the current work session can be open at any time. However, you can have concurrent open forms for different schemas in the Painter window, if required.

A *form* is a window that acts as a container for controls that display information and that permit user input and interaction. Forms have properties that determine aspects of their appearance (for example, position, size, or color) and aspects of their behavior (for example, whether they can be resized).

Tip To quickly access a specific form in Painter from the Class Browser, simply select the appropriate form in the Class List and then right-click on the **Painter** toolbar button.

Monitor Toolbar Button

Use the **Monitor** button from the Browser toolbar to monitor your Jade environment. You can monitor the Jade environment from any workstation.

The Jade Monitor enables you to view:

- Users
- Licence information
- Locks in place
- Notification
- Online statistics
- Queued locks

The Jade Monitor start-up dialog is then displayed, to enable you to select the parts of the Jade environment that you want to monitor. For more details, see the *Monitor User's Guide*.

Toggle Breakpoint Toolbar Button

Use the **Toggle Breakpoint** button from the Browser toolbar to set or unset a breakpoint on the line of logic where the caret is currently positioned in any Jade method. When a breakpoint is set, the selected line of logic is then highlighted in yellow (or the selected color of your choice). When a breakpoint is unset, the current line is no longer highlighted.

Note You can also set breakpoints from the Debugger window.

Breakpoints assist you in analyzing your logic, by interrupting execution of your logic. When you run a Jade application and the debugger is activated, the debugger stops before executing a line of logic that contains a breakpoint.

Include breakpoints at important points in your logic so that you can observe the flow of logic and determine the values of items. Breakpoints are invalid for blank lines, lines that contain only comments, and **end** instruction lines.

Note By default, the lines on which breakpoints are set (debug lines) are highlighted in yellow. You can change these display colors by using the **Editor** sheet of the Preferences dialog, accessed from the **Preferences** command in the Options menu of browse windows. A breakpoint always shows an icon in the breakpoint column of the user method in the editor pane. An enabled breakpoint set on a line indicated by a red circle is displayed in the breakpoint margin at the left of the line and a disabled breakpoint by a white circle.

The Preferences dialog **Editor Options** sheet provides the **Highlight entire source line for breakpoints** check box that controls whether the display of a breakpoint highlights the whole source line for a breakpoint. (This check box is checked by default.)

For more details, see Chapter 7.

Bookmark Toolbar Button

Use the **Bookmark** button from the Browser toolbar to create a bookmark for the current position in a Class, Primitive Types, or Interface Browser window. (You can also set a bookmark by selecting the **Set Bookmark** command from the View menu in the Class Browser, Primitive Types, or Interface Browser.)

You can set a bookmark to return to a class, primitive type, interface, property, or method.

A *bookmark* marks a specific place in a class, primitive type, or interface, to enable you to quickly move to the marked location from another part of that browser. For example, if you select the **myFaults** property and **display** method in the **Customer** class of the Class Browser and you set a bookmark, you can perform any other activity in the Jade Platform development environment such as maintaining forms, and then return to the position of the bookmark at any time by selecting the **Bookmark** toolbar button when the Class Browser is the current window.

Notes You can set only one bookmark in Class Browser, Primitive Types Browser, and Interface Browser windows. If you have a bookmark set in a method in the Class Browser and the Primitive Types Browser or Interface Browser is the current window, you cannot go to the class bookmark until the Class Browser has focus.

If you have created a bookmark and then you subsequently go to that bookmark, the bookmark is no longer retained; that is, the **Bookmark** toolbar button toggles the setting or unsetting of a bookmark.

Reorg Toolbar Button

Use the **Schema Needs Reorg** toolbar button to reorganize your current schema when it is marked for reorganization (indicated by a red light on the toolbar button).

When you click this button when it indicates that reorganization is required, the Classes Needing Reorg dialog is then displayed.

When the schema is not marked for reorganization, the toolbar button changes to a green light and bubble help for the button displays **Reorg Is Not Required**.

For details about reorganizing your database, see "Performing a Reorganization", in Chapter 3.

History Back Toolbar Button

Use the **History Back** button from the Browser toolbar to access the previous method, property, or type (for example, class or primitive type) that was accessed in the current schema from that browser. This button is disabled when no entity was previously accessed in the current schema from that browser.

You can view a history of the last 25 entities accessed in the current schema from that browser, in the order in which they were accessed. (For details, see "Displaying or Clearing a History of Accessed Entities", later in this chapter.)

History Next Toolbar Button

Use the **History Next** button from the Browser toolbar to access the method, property, or class (for example, class or primitive type) that was next accessed in the current schema from that browser. This button is disabled when the current entity is the last of the 25 entities in the history list for that schema or you did not access any other entity after the one that is currently displayed.

For details, see "Displaying or Clearing a History of Accessed Entities", later in this chapter.

Context Help Toolbar Button

Use the **Context Help** button from the Browser toolbar to access context-sensitive help for a specific element; for example, a window or a dialog control.

To access information about a window, menu, or a specific part of a dialog

1. Select the **Context Help** toolbar button.
2. When the mouse pointer (cursor) changes to an arrow and a question mark, point at the screen, window, or toolbar button about which you want information.
3. Click the left mouse button.

The help window is then displayed, showing information about the selected element. For example, if you click the context help pointer in the Methods List window of the Class Browser, a window of help information about that window is then displayed.

General Help Toolbar Button

Use the **General Help** button from the Browser toolbar to open the online help.

Depending on the value of the **UseJadeWebHelp** parameter in the [JadeHelp] section of the Jade initialization file, the default page of the product information (the first page of the Jade online help directory document) is then displayed on a web page in your browser or the directory document (**Jade.pdf**) is then displayed in Adobe Reader, providing a summary of and hyperlinks to all documents in the Jade Platform product information library.

Use the functions available in Jade online help to find the required topics. For details, see "Jade HTML5 Online Help" or "Jade Platform Product Information Library in Portable Document Format", later in this chapter.

Using the Editor Clipboard Toolbar

The Jade Platform development environment provides an editor clipboard toolbar at the right of the Browser toolbar in the main window, to enhance your use of the internal Jade editor clipboards and the Windows clipboard. (For details about using multiple Jade internal clipboards to copy the same code fragments (snippets) into many different unrelated classes where the code cannot be inherited from a common superclass and you have more than one code fragment, see "Using Multiple Clipboards when Refactoring Code" under "Editor Pane Shortcut Keys", later in this chapter.)

The Editor Clipboard toolbar has 11 buttons labeled **C** and numbers in the range **1** through **9** followed by zero (**0**), which represent the Windows clipboard status (the **C** button) and whether the internal clipboard of that number contains copied text. (The numbered buttons are in the sequence of the numeric keys on a keyboard.) The button of a clipboard that contains no text is disabled and the letter or number is displayed in light gray, as shown in the following image.

If you copy content into a Jade internal clipboard buffer using the Ctrl+Alt+C key combinations, releasing the pressed keys, and then pressing one of the numeric keys that indicates the buffer (that is, **1** through **9**, and **0**), the button number is shown in blue font and in bold. The **C** buffer contains any text copied to the Windows clipboard.

Note Clipboard buffers **C**, **1** through **9**, or zero (**0**) support copying and pasting selected rectangular content.

To view the clipboard text in bubble help, move the mouse over the clipboard buffer.

Clicking a clipboard buffer button pastes the content into the current editor pane at the location of the caret (or carets) or replaces text that is currently selected (that is, it performs the same action as the Ctrl+Alt+V key combinations, releasing the pressed keys, and then pressing the numeric key that indicates the number of the buffer).

Note The clipboard buffer buttons are disabled if the current schema is not a user schema (that is, it is the RootSchema) or if the editor pane does not have focus.

When you exit from the Jade Platform development environment, the contents of the internal Jade text editor clipboard buffers and any non-empty additional text entries defined in a floating toolbar that you have added (the **x<*n*>** entries) are saved in your user profile. Text in the Windows clipboard is *not* saved.

When you next initiate the Jade Platform development environment, the Jade internal clipboard buffers and additional text entries are restored with the saved values.

You can dock the editor clipboard toolbar separately at the left, top, right, or bottom of the Jade Platform development environment background form. For details, see "Floating the Editor Clipboard", in the following subsection.

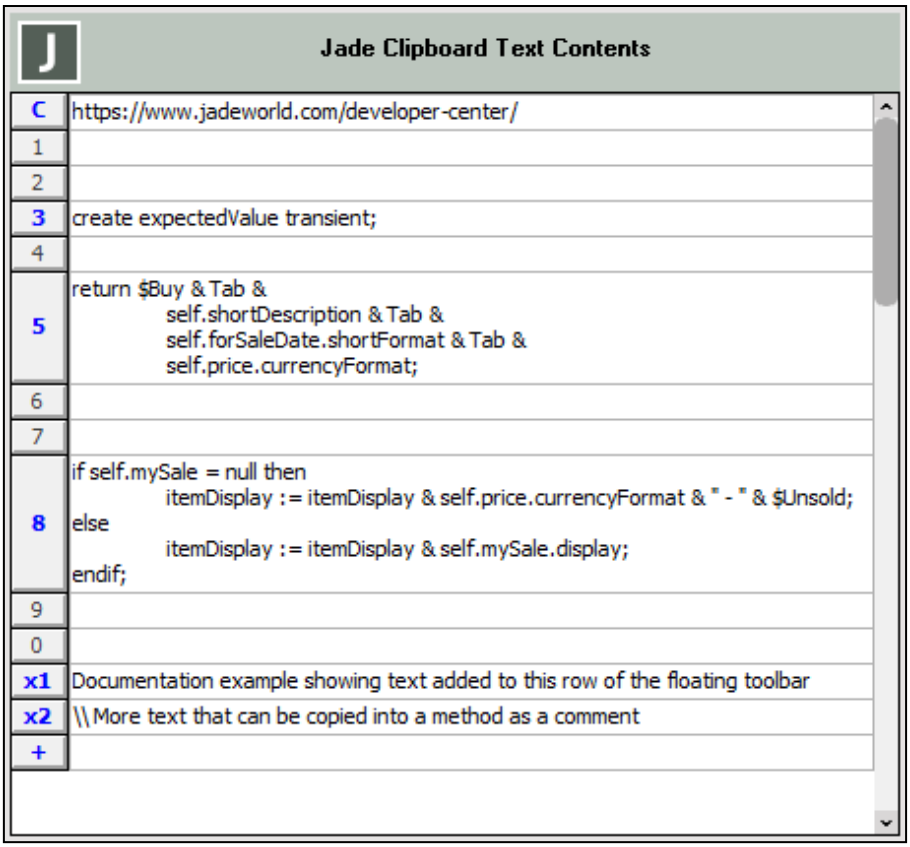
For details about toggling the display of the editor clipboard toolbar or the floated Jade Clipboard Text Contents form, see "Toggling the Display of the Editor Clipboard", later in this chapter.

Applies to Version: 2016.0.01 and higher

Floating the Editor Clipboard

You can dock the editor clipboard toolbar separately at the left, top, right, or bottom of the Jade Platform development environment background form. For details about the editor clipboard toolbar, see "Using the Editor Clipboard Toolbar", in the previous section. See also "Toggling the Display of the Editor Clipboard", in the following section.

When the editor clipboard toolbar is first floated, each text box is sized vertically to match the clipboard contents up to a maximum of 150 pixels.



The dock or float status and the floating size and position of the editor clipboard are saved if you have set your **Save Windows** user preference to **true** (that is, you have checked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog). When you next initiate the Jade Platform development environment, the editor clipboard toolbar status is restored.

If you float the editor clipboard toolbar by dragging the toolbar off the development environment main toolbar, the Jade Clipboard Text Contents form is displayed, containing the following in a table format.

- A button labeled **C** (that is, Windows clipboard), followed by a text box that contains any text that is in the Windows clipboard.

- Buttons in the range **1** through **9** followed by zero (**0**), each followed by a text box that contains the text from each internal Jade text editor clipboard buffer, if applicable.
- A button labeled **+**, which enables you to add extra lines of text to the table view by typing the required text or code in the second column of the **+** row.

  There is no key sequence available to paste text selected in the editor pane into the **+** row. You must click in the second column of that row and specify the required text. Each additional row becomes an **x<*n*>** row when the cell loses focus.

A button in the table is disabled if there is no text in that clipboard buffer.

If you change the contents of the Windows clipboard or an internal Jade editor clipboard, the appropriate text box is updated with that change.

If a Jade text editor pane has focus, clicking a button pastes that text into the editor pane, replacing any text that is currently selected.

You can manually edit the text box contents, so that when the text box loses focus, the appropriate clipboard buffer is updated.

Note As the text boxes accept tabs, the standard tab action to leave a text box is not available. To change focus, you must use the mouse to click another window.

If the text box content is a rectangular selection, when you edit the content the rectangular state is retained as long as at least one carriage return / line feed character (**CrLf**) remains. Reducing the content of a rectangular selection to one line removes its rectangular state and the content is pasted in simple (or stream) format.

Note You cannot add the rectangular state to content in stream format in a clipboard buffer.

After text is entered into the field labeled **+** and the text box loses focus, the button is labeled **x<*n*>**, indicating that there is extra text in the buffer, with the **<*n*>** value being the extra sequence number. Another button labeled **+** with an empty text box is then created, allowing additional text to be available for pasting beyond the available clipboard buffers. As there is no key sequence available to paste the text, you must click the associated button.

You can resize the table button entries vertically, which sizes the associated text box accordingly.

Applies to Version: 2016.0.01 and higher

Toggling the Display of the Editor Clipboard

To toggle the display of the editor clipboard toolbar or the floated Jade Clipboard Text Contents form

- Check or uncheck the **Show Clip Board Toolbar** check box on the **Window** sheet of the Preferences dialog
- Select the **Show Clipboard Toolbar** command in the View menu
- Select the **Show Clipboard Toolbar** command from the toolbar context (popup) menu

The editor clipboard toolbar or the floated Jade Clipboard Text Contents form is then displayed or hidden until you repeat the action.

If the editor clipboard toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the editor clipboard toolbar.

Applies to Version: 2016.0.01 and higher

Using Custom Toolbar Buttons

You can add one or more custom buttons (icons) to the Jade Platform development environment toolbar to run a specific application, **JadeScript** method, or Jade test cases in standard run or debug mode, regardless of the current schema. You can then click these buttons to initiate the required action without having to switch the Jade Platform development environment view to the required environment.

Note Custom toolbar buttons apply to *your* view of the Jade Platform development environment; not that of other users.

A button with the caption **+** is positioned to the right of the clipboard toolbar. This button initiates a dialog from which you define the functionality to be assigned to a new button that will be added to the toolbar area. You must select a schema and application from the relevant combo boxes on the dialog. For details, see "Adding a Toolbar Button to Run a Script or Application", elsewhere in this document.

When you click an added custom button with the specified caption, a new copy of the selected application is started and it runs under the selected schema. The following image is an example of custom buttons in the Jade Platform development environment toolbar.

Hovering the mouse over a custom toolbar button displays the specified functionality; for example:

```
Start application 'ErewhonShop' using schema 'ErewhonInvestmentsViewSchema'
```

The bubble help display also includes **Use mouse right click to edit the button definition**.

To edit or delete an existing custom button

1. Right-click the button. The custom button definition dialog (Update existing run application / script button) is then displayed with the options and combo box selections defined for the button.
2. Click the **Delete** button to remove the button from the toolbar, or edit your specified values and then click the **Update** button to change the actions assigned to the button.

Notes When the Jade Platform development environment is closed, any defined custom button descriptions are stored in your user preferences file so that your custom buttons are restored when you next start the Jade Platform development environment.

All combo boxes filter the displayed list according to the content of any text typed into the text box portion of the combo box.

Applies to Version: 2022.0.01 and higher

Adding a Toolbar Button to Run a Script or Application

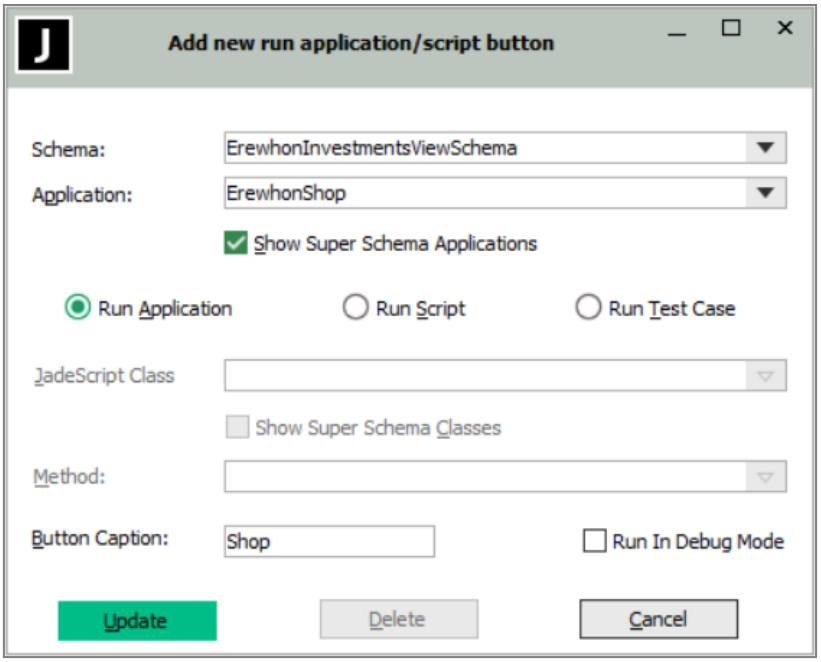
You can add one or more custom buttons to the Jade Platform development environment toolbar to run a specific application, **JadeScript** method, or Jade test cases in standard run or debug mode, regardless of the current schema. These buttons initiate the required action without having to switch the Jade Platform development environment view to the required environment.

For details about custom toolbar buttons, see "Using Custom Toolbar Buttons", elsewhere in this document.

To add a new custom button

1. Click the button with the caption **+** positioned to the right of the clipboard toolbar.

   The Add new run application / script button dialog, shown in the following image, is then displayed to enable you to define the functionality to be assigned to a new button that will be added to your view of the toolbar area.



2. Select both a schema and an application from the relevant **Schema** and **Application** combo boxes.

   By default, the **Application** combo box contains only the applications defined for the selected schema.

3. Check the **Show Super Schema Applications** check box if you want to include all superschema applications in the application list of the **Application** combo box.

4. Select the option button of the type of action that you require; that is:

   - **Run Application**, if you want the new button to run the specified application in the specified schema.

     This action is the same as selecting the required schema in the Jade Platform development environment, initiating the Run Application dialog, and then selecting the required application.

   - **Run Script**, if you want to add all **JadeScript** classes for the selected schema to the **JadeScript Class** combo box.

     Check the **Show Super Schema Classes** check box if you want to include all superschema **JadeScript** classes in the **JadeScript Class** combo box.

     Only classes that have suitable **JadeScript** class methods are included (that is, those with no type methods and only those methods that have no parameters).

     When you select a **JadeScript** class, the **Method** combo box is populated with the available methods that can be executed. Select the method that is to be run when the new custom button is clicked.

The selected application will be started under the selected schema and will run the selected method when the new custom button in the Jade Platform development environment toolbar is clicked. This action is the same as selecting the specified schema, the **JadeScript** class, and the required method in the Class Browser and then selecting the File menu **Run** command.

- **Run Test Case**, if you want to add all **JadeTestCase** classes for the selected schema to the **Test Case Class** combo box.

  Check the **Show Super Schema Classes** check box if you want to include all superschema **JadeTestCase** classes in the **Test Case Class** combo box.

  Only classes that have suitable **JadeTestCase** class methods are included (that is, only test case methods).

  By default, the **Test Case Class** combo box displays **- All Test Cases for Schema -**. When this entry is selected, clicking the **+** toolbar button in the Jade Platform development environment is equivalent to selecting the **Unit Test** command or **Unit Test Debug**) command from the Jade menu in the Schema Browser.

  The Unit Test Runner form that is then displayed lists all test cases for the schema.

  If you select a specific test case class in the **Test Case Class** combo box of the Add new run application / script button dialog, the **Method** combo box is populated with the suitable test case methods. By default, the **Method** combo box displays **- Run all Test Case methods for class -**, which causes the execution of all test cases for the selected class when you click the **+** toolbar button in the Jade Platform development environment. Alternatively, select a specific method if you want to execute that test case method only.

  The test case execution is run by starting the selected application using the selected schema.

5. To complete the set up of your new custom button, specify the caption for your new button in the **Button Caption** text box. The caption has a maximum length of 20 characters.
6. If you want the code executed in debug mode, check the **Run in Debug Mode** check box to cause the execution to run in the Jade Debugger.
7. When you have specified all of your required values, click the **Update** button to add a new button with the specified caption to the right of the **+** button in the toolbar or to the right of any previously defined custom buttons.

When you close the Jade Platform development environment, any defined button descriptions are stored in your user profile and they are restored when you restart the Jade Platform development environment.

Note All combo boxes filter the displayed list according to the content of any text typed into the text box portion of the combo box.

Applies to Version: 2022.0.01 and higher

Using Jade Platform Development Environment Menus

The menus (and the commands in menus) that are displayed at any time depend on the type of browser window that is currently displayed. For example, the **Status List** command is available from the Browse menu in the Class, Primitive Types, Interface, and Schema Browsers.

Notes Menu commands that are not available for selection are disabled (dimmed).

There are faster ways of accessing functions; for example, popup menus or toolbar buttons for frequently used functions. (You can use the Esc key to cancel the display of popup menus.)

To access a menu

- Click the required menu.

A drop-down list is then displayed, showing the commands that are available in the selected menu.

The Jade Platform development environment menus are listed in the following table.

Menu	Description
Ad Hoc Indexes	ODBC query indexes suitable for optimizing ad hoc queries without requiring database reorganization (for details, see "Maintaining Ad Hoc Indexes", in Chapter 9)
Application	Browses and maintains your schema applications
Breakpoints	Maintains debugger breakpoints
Browse	Browsing functions; for example, viewing breakpoints or changed methods
Categories	Browses and maintains global constant categories
Classes	Browses and maintains your schema classes
Components	Browses and maintains ActiveX control, ActiveX automation, and .NET external object libraries
Conditions	Browses and maintains your class conditions
Constants	Browses and maintains your class, primitive type, interface, and global constants
Consumer	Maintains your web service consumers (for details, see "Defining a Web Service Consumer Application", in Chapter 11 of the *Developer's Reference*)
Databases	Administers external database schemas
Deltas	Browses and maintains change control deltas
Edit	Editing functions; for example, copy or search functions
Exposure	Maintains Jade objects exported through a C# class library or a web service (for details, see Chapter 16 or Chapter 17 in this document, or "Defining a Web Service Provider Application", in Chapter 11 of the *Developer's Reference*, respectively)
File	Administers a Jade development database
Functions	Browses and maintains external functions
Help	Accesses the standard Common User Access (CUA) help options
History	Displays or clears a history of the last method, property, and type entities accessed from the browser
HTML	Browses and maintains your HTML documents
Interfaces	Browses and maintains interfaces
Jade	Executes Jade methods or tests code units, debugs all of or a selected portion of a Workspace window or a **JadeScript** method, or accesses the Schema Browser
Library	Browses and maintains your Jade libraries
MapFiles	Browses and maintains your class map files
Methods	Browses and maintains your class, primitive type, and interface methods
Methods Viewer	Browses and maintains customized methods browsers displaying only the required methods from any class in any schema (for details, see "Using Method Views to Bookmark Workflows", in Chapter 13)

Menu	Description
Options	Maintains your global options for the schema and all subschemas
Packages	Browses and maintains your export and import packages (for details, see Chapter 8, "Using Packages", in the *Developer's Reference*)
Properties	Browses and maintains your class properties
Relational	Browses and maintains your relational views and RPS mappings
Schema	Browses and maintains your schemas
Server	Browses and maintains Jade Object Manager servers, both local and remote
Types	Browses and maintains primitive types
View	Provides different ways of looking at your Browser windows
Views	Browses and maintains your schema views
Window	Arranges and manipulates child windows in an MDI Jade application

For details about Painter menus, see "Painter Menus", in Chapter 5.

Using Popup Menus

Use popup, or context, menus to quickly display the choices for your currently selected item.

To access a popup menu

- Right-click on the required item.

A popup menu contains the same commands as the menu in the menu bar. For example, a popup Methods menu is displayed at the caret position when you right-click on a method in the Methods List of the Class Browser. (You can use the Esc key to cancel the display of popup menus.)

Using the Refresh Facility

In multiuser mode, use the **Refresh** command from the View menu in a browser window to refresh (update) your current window. This enables you to view changes made to that window by other users. (See also "Refreshing the Browser", in Chapter 3.)

Displaying or Clearing a History of Accessed Entities

When you have accessed one or more method, property, and type (for example, class or primitive type) entities in the current schema in your work session, the History menu enables you to view a history of the last 25 entities accessed from that browser, in the order in which you accessed them. The type in which a method or property is defined precedes the method or property name in the history list.

The behavior of the Jade hierarchy browser history list differs between the three methods of selection, as follows.

- Clicking a history item in the History menu moves the item to the top of the list.
- Clicking the **History Back** and **History Next** buttons does not move the history item to the top of the list, and the item remains in its current position.
- Using the Ctrl+Alt+left or right arrow keys shortcut does not move the item to the top of the list.

Unless you use the History menu, entries will be eligible to be removed earlier, because the last entry in the list is removed when a new entry is added and the list size exceeds 25.

To display a specific method, property, or type entity that you have accessed in the current work session, click on the entity in the History menu list.

The entity that is then selected in the Class, Primitive Type, Interface, Properties, or Methods List and displayed in the editor pane is indicated by a check mark to the left of the entity name in the history list.

Tips Press Ctrl+Alt+B to view the previous history entry in the list or Ctrl+Alt+N to view the next history entry, or use the respective **History Back** and **History Next** Browser toolbar buttons.

The **Browser** sheet of the Preferences or Jade Installation Preferences dialog enables you to specify that you want to list only the most-recently accessed methods in the history list. For details, see "Maintaining Browser Options", later in this chapter.

Clearing the History of Accessed Entities

» To clear the history of the most-recently accessed entities in the current work session

- Select the **Clear History** command from the History menu.

  This command is disabled if you have not yet accessed an entity from the current browser in your work session or you have already cleared the history but not yet accessed another entity from the browser.

The list of all entities accessed from that browser is then removed from the list. (The history list is also cleared when you access another browser.)

Using the Quick Navigation Facility

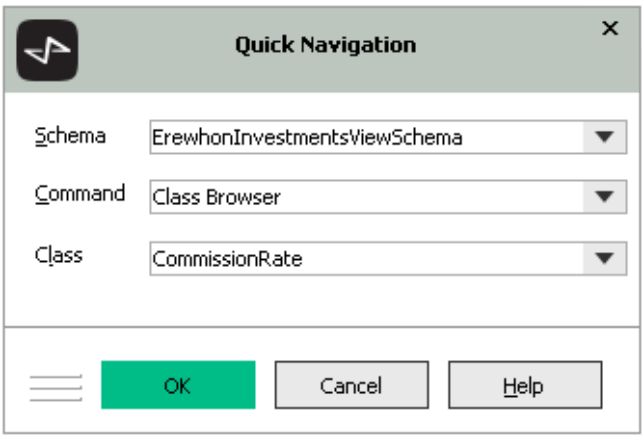
Use the **Quick Navigation** command from the File menu to quickly access:

- A specified class in the Class List of the Class Browser
- An interface selected in the Interface List of the Interface Browser
- A method for a class selected in the Class List of the Class Browser

» To specify the class to access, perform one of the following actions

- Select the **Quick Navigation** command from the File menu.
- Press Ctrl+Q.

The Quick Navigation dialog, shown in the following image, is then displayed.



The **Schema** combo box displays the name of the current schema selected in the Schema Browser. The **Command** combo box is set to **Class Browser** by default.

Tip You can use this dialog to navigate to a class, interface, or method in a schema other than the current schema, if required. For details, see "Navigating to a Specified Class in the Class List", "Navigating to a Specified Interface in the Interface Browser", or "Navigating to the Methods for a Specified Class", in the following subsections.

Navigating to a Specified Class in the Class List

To access a specified class

1. If you want to navigate to a class in another schema, select the appropriate schema in the drop-down list or specify the name in the text box of the **Schema** combo box.
2. Ensure that **Class Browser** is displayed in the **Command** combo box. (This is the default setting.)
3. In the **Class** combo box, select the class to which you want to navigate in the drop-down list or specify part or the entire name in the text area so that your required class is selected.

   Tip For details about filtering the entries displayed in the list box associated with the **Class** combo box, see "Filtering Entries in Combo Box and List Box Text", elsewhere in this chapter.

4. Click the **OK** button.

The Class Browser is then displayed, showing only the specified class and any subclasses. (For details about using this browser, see "Using the Class, Primitive Types, or Interface Browser", in Chapter 3.)

Navigating to a Specified Interface in the Interface Browser

To access a specified interface

1. If you want to navigate to an interface in another schema, select the appropriate schema in the drop-down list or specify the name in the text box of the **Schema** combo box.
2. Ensure that **Interface Browser** is displayed in the **Command** combo box. (The caption of the next control then changes to **Interface**.)

3. In the **Interface** combo box, select the interface to which you want to navigate in the drop-down list or specify part or the entire name in the text area so that your required interface is selected.

   Tip For details about filtering the entries displayed in the list box associated with the **Interface** combo box, see "Filtering Entries in Combo Box and List Box Text", elsewhere in this chapter.

4. Click the **OK** button.

The Interface Browser is then displayed, with the specified interface selected in the Interface List. (For details about using this browser, see "Accessing the Interface Browser", in Chapter 14.)

Navigating to the Methods for a Specified Class

To access the methods for a specified class

1. If you want to navigate to a method in another schema, select the appropriate schema in the drop-down list or specify the name in the text box of the **Schema** combo box.
2. In the **Command** combo box, select **Methods Browser** from the drop-down list or specify **Methods Browser** in the text area.
3. In the **Class** combo box, select the class to whose methods you want to navigate or specify part or the entire name in the text area so that your required class is selected.

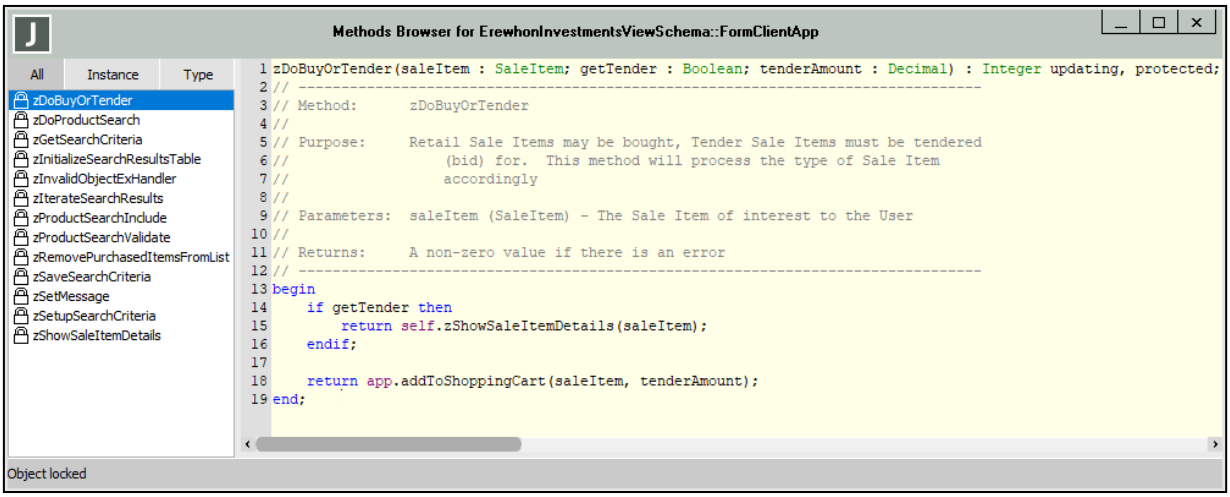
   Tip For details about filtering the entries displayed in the list box associated with the **Class** combo box, see "Filtering Entries in Combo Box and List Box Text", elsewhere in this chapter.

4. Click the **OK** button.

Tips You can also access the Methods Browser for a class selected in the Class List of the Class Browser, a primitive type selected in the Primitive Type List of the Primitive Types Browser, or an interface selected in the Interface List of the Interface Browser, by selecting the **Methods Browser** command from the Classes menu, the Types menu, or the Interfaces menu. A standalone browser containing only the methods defined in the selected class, primitive type, or interface is then displayed.

You can group methods from any class in any schema into a named workspace that assists you in book-marking workflows (for example, all methods and events that relate to a specific feature, with the method view description advising other users about the functionality provided by the methods). For details, see "Using Method Views to Bookmark Workflows", in Chapter 13.

The Methods Browser for the specified class is then displayed, as shown in the following image.



The methods provided by instances of the class are listed on the left of the browser.

To view the logic for a method

- Click on the appropriate method in the list.

The logic for the selected method is then displayed in the pane on the right of the Methods Browser. You can select the **Remove** command from the Methods menu to remove a selected method from the class or you can select the **References** command or the **Implementors** command from the Methods menu to view the references or implementors of the selected method. (See also "Removing a Schema Element", in Chapter 3.)

Note You can modify a method that is displayed in the editor pane. If you modify a method and then close the Methods Browser before you have saved or compiled the method, a dialog prompts you to save the method before the window is closed.

To compile a displayed method, perform one of the following actions

- Press F8
- Select the **Compile Method** command from the Methods menu

To execute a displayed JadeScript method or Workspace code

1. In a Workspace only, drag the mouse over the logic to select the body of the method.
2. The Jade menu displays the **Execute It** command if the **JadeTestCase** class or subclass is not selected. Perform one of the following actions.
   - Press F9 to execute the selected logic of a displayed method or all code in a Workspace.
   - Select the **Execute It** command from the Jade menu.

Notes If the **JadeScript** method or Workspace code is not compiled, the F9 key first compiles the method before executing it.

Running a **JadeScript** method always starts a new application copy.

You can execute a **JadeScript** method or Workspace code only from the current schema context (version).

To test JadeTestCase class methods

- The Jade menu displays the **Unit Test** command if a user-defined schema with the **JadeTestCase** subclass, a **JadeTestCase** class or subclass, or a method in a **JadeTestCase** subclass is selected.

  Perform one of the following actions.

  - Press F9.
  - Select the **Unit Test** command from the Jade menu.

  The Unit Test Runner form is then displayed. For details, see "Running Unit Tests", in Chapter 17 of the *Developer's Reference*.

You can press F9 or select the **Unit Test** command from the Jade menu to test a **JadeTestCase** entity from a schema, class, or method selected in the applicable browser.

In the Schema Browser, if the selected schema has user subclasses of **JadeTestCase**, pressing F9 or selecting the **Unit Test** command from the Jade menu (which is enabled only in this situation) runs the tests for all **JadeTestCase** classes in the selected schema and in all subschemas that also contain such tests.

In the Class Browser, pressing F9 or selecting the **Unit Test** command from the Jade menu with a **JadeTestCase** or subclass selected runs the test for this class and for all subclasses.

If a test method is selected in the Methods List of the Class Browser, the test is run for that method only.

Notes If a **JadeTestCase** class or subclass method is selected and it is not compiled, the F9 key first compiles the method before testing it.

You can test a **JadeTestCase** class or subclass method only from the current schema context (version).

To debug the execution of a Jade unit test

The Jade menu displays the **Unit Test Debug** command if a user-defined schema with the **JadeTestCase** subclass, a **JadeTestCase** class or subclass, or a method in a **JadeTestCase** subclass is selected.

- Select the **Unit Test Debug** command from the Jade menu.

The Jade unit test framework is then initiated in Jade debug mode for the selected **JadeTestCase** class or selected method of the class if a method is selected. For details, see "Running Unit Tests", in Chapter 17 of the *Developer's Reference*.

Displaying a Light or Dark Development Environment Theme

You can set a pre-defined light or dark color mode in the Jade Platform development environment so that you can select a dark theme if you prefer the dark color set that reduces excessive brightness. The Jade development environment also provides a default Jade theme and dark and light skins.

A Jade theme consists of:

- A name
- The Window colors
- The Editor colors
- The Relationship colors

The **Window** sheet of the Preferences dialog in the Jade Platform development environment enables you to:

- Change between a light theme or a dark theme. (The **Editor** sheet and the sample list box in the **Window** sheet reflect the selected theme.)
- Select from dark and light versions of skins; for example. **Jade Lava Lamp Dark** and **Jade Lava Lamp**.
- Add, edit, reset, or remove a user-defined theme.

When you click the **OK** button on the Preferences dialog, the entire development environment is updated to reflect the selected theme.

The **Miscellaneous** sheet enables you to import or export a user-defined theme. (These options are not available on the administrative Jade Installation Preferences dialog, as they are user-specific.)

For details about maintaining your user preferences, see "Setting User Preferences", later in this chapter.

» To set a theme

1. Select the **Preferences** command from the Options menu to open the Preferences dialog, and then click the **Window** tab of the **Window** sheet.
2. Select the **Light Mode** or **Dark Mode** option button in the Color Mode group box.

   The **Select Jade Theme** combo box is then populated with all available themes of the selected light or dark mode.
3. Select the light or dark theme that you require.

   The colors listed on the **Window**, **Editor**, and **Relationship** sheets and the sample list at the right of the **Editor** sheet are updated to reflect the selected theme.
4. Click the **OK** button if you want to apply the selected theme and update your user profile. Alternatively press Esc or click the **Cancel** button so that the selected theme is not applied and the current theme remains selected in the combo box and the colors remain as they were.

Applies to Version: 2020.0.01 and higher

Using Function Keys and Shortcut Keys

You can use the keyboard to perform functions in the Jade Platform development environment.

- The keyboard shortcut commands in the editor pane follow standard conventions.
- Move a class to a new superclass or copy or move a method to another class by holding down the Ctrl or Shift key and dragging the class or method to the required class.

- Press Ctrl+Home to move a list box or table with an associated collection to the first entry in the collection. Conversely, press Ctrl+End to move the display to the last entry in the collection.

  Note Pressing the Home key on a list box or table moves the display to the first entry that has been loaded in the control. Pressing the End key moves the display to the last entry currently loaded in the control. Using the Ctrl key for a list box or table that does not have a collection attached has the same result as pressing the Home or End key without the Ctrl key.

- To display the Painter form of the **Form** subclass selected in the Class List of the Class Browser, right-click on the **Painter** toolbar button.
- If you have set a Schema View, right-click on the **Browse Classes** toolbar button to display a Class Browser for that Schema View.

You can change most of the development environment and editor shortcut keys to values of your choice; for example, changing the editor Ctrl+S key binding (**Insert Syntax**) to the **Save** function. The exceptions are editor pane accelerator key bindings, standard Windows functionality shortcuts (for example, for copy, paste, and undo actions), and Windows form shortcuts. In addition, you cannot:

- Add a shortcut to a menu item that does not have a shortcut defined in the Jade Painter
- Change the key accelerator in the caption of a menu item; for example, **&File** to **F&ile**

Notes Visual Studio uses the F9 key to set breakpoints, by default, and uses the F5 key to run and continue debug execution. Jade uses the F5 key to set breakpoints and the F9 key to run and continue debug execution. Switching between the two development environments can lead to confusion and frustration when using the F5 and F9 keys. You can swap the F9 and F5 accelerator key bindings, by checking the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) accelerators** check box on the **Short Cut Keys** sheet on the Preferences dialog. This check box is unchecked by default (that is, false).

You can swap the F11 and F12 accelerator key bindings (for example, if you want the F12 key in the Jade editor pane to have a similar meaning to F12 in Visual Studio, where the definition of the selected symbol is displayed), by checking the **Swap F11 (Show Symbol/Open class browser) and F12 (New Window/Bookmark) accelerators** check box on the **Short Cut Keys** sheet on the Preferences dialog. This check box is unchecked by default (that is, **false**).

The key bindings are saved in your user profile and can be extracted to a preferences initialization file by using the existing **Export Preferences** functionality provided on the **Miscellaneous** sheet of the Preferences dialog.

For details about configuring Jade shortcut keys, see "Maintaining Shortcut Keys" under "Setting User Preferences", later in this chapter. For details about using the F9 function key to run a unit test, see "Running a Unit Test", in Chapter 4.

Using Function Keys

The function keys listed in the following table enable you to perform functions in the Jade Platform development environment by using the keyboard.

Function Key	Action
Ctrl+Alt+I	Sets focus to the search combo box when the quick inspect toolbar is docked, or the search text box on the Quick Inspect form when the quick inspect toolbar is floating. (For details, see "Using the Quick Inspect Toolbar", in Chapter 6.)
Ctrl+Alt+Q	Sets focus to the search combo box on the Inspector form or the search text box in the quick inspect toolbar. (For details, see "Using the Quick Inspect Toolbar", in Chapter 6.)

Function Key	Action
Ctrl+F	Displays the Find/Replace dialog. Selected text, if any, is the default search string.
F1	Displays online help for the item with input focus.
F2	Saves the method without compiling it or from a workspace, saves it to disk.
Shift+F2	Executes the **Rename / Change** command from the Refactor submenu of the Edit menu.
Ctrl+F2	Displays a dialog that enables you to change or rename the entity selected within the body of a method in the editor pane of the Class Browser.
F3	Performs the **Find Again** function.
Shift+F3	Performs another search using your last set of search options but in the opposite direction.
Ctrl+F3	Searches for the next occurrence of the text currently under the caret and sets the search string for subsequent finds.
Ctrl+Shift+F3	Displays the Global Search and Replace dialog. Selected text, if any, is the default search string.
F4	Displays the Find Type dialog to enable you to locate a class, interface, primitive type, or a schema, depending on the current open browser.
F5	Toggles a debugger breakpoint on the line where the caret is currently positioned.
F6	Adds selected text as a translatable string when text is selected (from the editor pane).
Shift+F6	Displays a list of translatable strings available for the current schema (from the editor pane).
F7	Goes to the next linemark.
Shift+F7	Goes to the previous linemark.
Ctrl+F7	Toggles the linemark on the current line.
F8	Compiles and saves the current method (from the editor pane).
Shift+F8	Saves the method source when the method is compiled if patch control is enabled and the **Save Source on Every Compile** check box on the **Source Management** sheet of the Preferences dialog is unchecked.
F9	Saves, compiles, and executes the current **JadeScript** method, compiles and executes the current Workspace or selected text, or invokes the Unit Test Runner form for a unit test if the selected schema has user subclasses of **JadeTestCase**, the **JadeTestCase** class or subclass is selected, or a method in a **JadeTestCase** subclass is selected.
Shift+F9	Saves, compiles, executes, and debugs the current **JadeScript** method or unit test.
F10	Activates key-controlled menus, and selects the first menu on the menu bar.
F11	Displays details about the item under the caret. When repeated for a type or global constant, opens a new browser window in the current schema for that type or global constant. The information includes the creation timestamp and patch version of the entity.
Shift+F11	Opens a new browser window in the schema in which the entity under the caret is defined.

Function Key	Action
F12	Opens a new editor pane with the selected method, or when details of a property that has a mapping method are displayed after pressing F11, displays a freestanding editor pane containing the mapping method source for that property. If the current method has been changed, a message box prompts you to save and compile your changes, to discard them, or continue editing the method before the new method source window is displayed. (For details about reusing the same form to display method source, see "Reusing the Form Displaying Method Source", in Chapter 3.)
Windows key + period (.)	Pastes a selected emoji or selection of emojis from the emoji selection window into **TextBox** controls or the editor pane in a Unicode Jade system. For more details, see "Unicode Surrogate Pair Characters, including Emojis", in Chapter 1 of the *Developer's Reference*.

Using Browser Shortcut Keys

Use shortcut keys to quickly perform actions in the Jade Platform development environment.

Browser Shortcut Keys

The shortcut keys listed in the following table are available from the browsers.

Key Combinations	Action
Ctrl+B	Displays a new Class Browser, with the class selected in the new browser defaulting to the same class that is currently selected when the action is performed from a hierarchy browser
Ctrl+D	Displays the Deltas Browser
Ctrl+E	Displays the Libraries Browser
Shift+Ctrl+F	Displays the External Functions Browser
Ctrl+G	Displays the Global Constants Browser
Ctrl+H	Displays the Schema Views Browser
Ctrl+I	Displays the Schema Collection Inspector for the current class (equivalent to selecting the **Inspect Instances** command from the Classes menu)
Ctrl+J	Displays any shared transient instances of the current class (Class Browser only)
Ctrl+L	Displays the Application Browser
Ctrl+M	Displays the Class Maps Browser
Ctrl+N	Displays any Interface Browser
Ctrl+O	Displays the Relational Views Browser
Ctrl+P	Displays the Jade Painter
Ctrl+Q	Displays the Quick Navigation dialog
Ctrl+R	Displays the Run Application dialog
Ctrl+T	Displays the Primitive Types Browser
Ctrl+U	Displays the HTML Document Browser
Ctrl+Alt+←	Accesses the previous entity that was accessed in the current schema from that browser

Key Combinations	Action
Ctrl+Alt+→	Accesses the entity that was next accessed in the current schema from that browser
Ctrl+Alt+G	Displays the Find Position in Method Source dialog
Ctrl+F6	Cycles through all open MDI child windows
Ctrl+Tab	Cycles through sheets on any folder on the window that currently has focus
Esc	Closes any open menu
←	Expands collapsed trees (**+**)
→	Collapses expanded trees (**-**)
***** (numeric keypad)	Expands all sub-trees

Caret Movement Shortcut Keys

The shortcut keys listed in the following table enable you to perform caret movement functions in the editor pane by using the keyboard.

Key	Moves the caret …
←	One character to the left. When text is selected, moves caret to the left of the selected text (that is, it falls off the left).
→	One character to the right. When text is selected, moves caret to the right of the selected text (that is, it falls off the right).
Ctrl+←	To the beginning of the word (delimited by white space and an alphanumeric or a non-alphanumeric character) on the left.
Ctrl+→	To the beginning of the word (delimited by white space and an alphanumeric or a non-alphanumeric character) on the right.
↑	Up one line.
↓	Down one line.
Ctrl+↓	Scrolls down the text displayed in the editor pane by one line.
Ctrl+↑	Scrolls up the text displayed in the editor pane by one line.
Ctrl+[	To the previous paragraph (press Shift to extend the selection). A paragraph is delimited by an empty line.
Ctrl+]	To the next paragraph (press Shift to extend the selection).
Ctrl+/	To the previous word part (press Shift to extend the selection).
Ctrl+\	To the next word part (press Shift to extend the selection).
Ctrl+Enter	To a newly inserted line without performing automatic indenting.
Page Up	Up one screen.
Page Down	Down one screen.
Home	To the first non-white space character on the current line if the caret is not already positioned there.
End	To the end of the current line.
Alt+Home	To the start of the display line (that is, to the first character on the current line).

Key	Moves the caret …
Alt+End	To the end of the display line.
Ctrl+Home	To the beginning of the editor text.
Ctrl+End	To the end of the editor text.

Note If wrapping is set for the editor, the current line (for Home and End keys) is defined by the entered new-line characters, not by the displayed line.

Editor Pane Shortcut Keys

The shortcut keys listed in the following table are available from the editor pane. (See also "Using Function Keys", earlier in this chapter.)

Key Combinations	Action
Right-click	Displays the Edit menu.
Ctrl+0	Toggles the display of multiple-color or single color for the current editor pane.
Ctrl+A	Selects all logic in the current method.
Ctrl+C	Copies the selected portion of the method to the clipboard.
Ctrl+S	Inserts the remainder of the syntax for a specified instruction.
Ctrl+Y	Reapplies the last editor action.
Ctrl+Z or Alt+Backspace	Undoes the last editor action.
Ctrl+V	Pastes portion of method from the clipboard to the current position.
Ctrl+W	Opens (splits) the lower editor pane view if it is not already open or switches focus between the two views if the lower view of the editor pane is already open.
Ctrl+X	Cuts the selected portion of a method and copies it to the clipboard.
Ctrl+numeric **+** symbol	Magnifies text size (zoom functionality).
Ctrl+numeric **-** symbol	Reduces text size (zoom functionality).
Ctrl+numeric **/** symbol	Restores text size to normal (zoom functionality).
Ctrl+Shift+1	Displays properties that are local to the class being accessed; that is, the properties of the root class, as subschema copies do not have properties.
Ctrl+Shift+2	Displays methods that are local to the type being accessed. If the nearest type is a subschema copy, only the methods that are in that subschema copy are displayed; otherwise the methods of the root type are displayed.
Ctrl+Shift+3	Displays constants that are local to the type being accessed. If the nearest type is a subschema copy, only the constants that are in that subschema copy are displayed; otherwise the constants of the root type are displayed.
Shift+Ctrl+I	Indents the selected line or lines, replaces selected text with a tab character for a partially selected line, or inserts a tab character if no text is selected.
Shift+Ctrl+U	Removes indentation from the selected line or lines.
Shift+Ctrl+W	Opens or closes (that is, toggles) the horizontal editor pane view.

Key Combinations	Action
Shift+F3	Performs another search using your last set of search options but in the opposite direction.
Shift+Tab	Removes indentation from the selected line or lines, or moves the caret to the left one tab character.
Shift+Alt+ arrow keys	Makes a rectangular selection.
Alt+ dragging the mouse	Makes a rectangular selection.
Shift+Alt+↓, Tab	Selects multiple lines (without selecting any characters) and then adds tab characters into the lines.
Shift+Alt+↓, Backspace	Selects multiple lines (without selecting any characters) and then removes tabs, moving text back to the prior tab position.
Tab	Indents the selected line or lines, replaces selected text with a tab character for a partially selected line, or inserts a tab character if no text is selected.

In addition to the keyboard zoom functionality commands listed in the above table, you can use the Ctrl key and the mouse to increase, decrease, or restore text size; that is, Ctrl+ scrolling the mouse wheel forward increases the zoomed size, Ctrl+ scrolling the mouse wheel backward decreases the zoomed size, and Ctrl+ clicking the middle mouse button (pressing the mouse wheel) restores the editor pane to normal.

See also "Renaming or Changing an Entity", in Chapter 4.

Accelerator Shortcut Keys

The shortcut keys listed in the following table are the default accelerator keys available from the editor pane.

Key Combinations	Inserts the …
Shift+Ctrl+A	**abortTransaction;** instruction at the caret position
Shift+Ctrl+B	**beginTransaction;** instruction at the caret position
Shift+Ctrl+C	**commitTransaction;** instruction at the caret position
Shift+Ctrl+E	**endforeach;** instruction at the caret position

You can define your own accelerator (short cut) keys for use in the editor pane. For example, you could define the F key to insert the **foreach** instruction when you press Shift+Ctrl+F. For details, see "Setting User Preferences", later in this chapter. Note, however, that in Compact Jade, defined menu accelerator keys are ignored and are not included in the displayed menu item text.

Context-Sensitive Options

The shortcut keys listed in the following table are available from the editor pane to perform context-sensitive actions.

Keys	Caret positioned after…	Displays…
Ctrl+1	*variable-name*.	List of properties for the type of *variable-name*
	method-name(	Method signature
	dictionary-name[	Keys (comma-separated) for the **Dictionary** class or subclass
	array-name[	**indx]** text

Keys	Caret positioned after…	Displays…
	string[or *binary*[	**startPos : forLength]** text for the **String** or **Binary** primitive type
	Elsewhere…	List of properties for the current type
Ctrl+2	*variable-name*.	List of methods for the type of *variable-name*
	method-name(	Method signature
	array-name[	**indx]** text
	dictionary-name[	Keys (comma-separated) for the **Dictionary** class or subclass
	string[or *binary*[	**startPos : forLength]** text for the **String** or **Binary** primitive type
	Elsewhere…	List of methods for the current type
Ctrl+3	*variable-name*.	List of constants for the type of *variable-name*
	method-name(	Method signature
	array-name[	**indx]** text
	dictionary-name[	Keys (comma-separated) for the **Dictionary** class or subclass
	string[or *binary*[	**startPos : forLength]** text for the **String** or **Binary** primitive type
	Elsewhere…	List of constants for the current type
Ctrl+4	Anywhere in method source window	List of local variables and parameters of the method, which were valid at the time the method was last compiled
Ctrl+5	Anywhere in method source window	Signature of the last method before the caret position; for example, when the caret is positioned after a comma between parameters

Extending Selected Blocks Shortcut Keys

The shortcut keys listed in the following table enable you to extend selected blocks in the editor pane.

Key Combinations	Action
Ctrl+Shift+End	Extends selection to end of text in the editor pane
Ctrl+Shift+Home	Extends selection to start of text in the editor pane
Shift+← or Alt+Shift+←	Reduces the selected block one character to the left
Shift+→ or Alt+Shift+←	Extends the selected block one character to the right
Shift+End or Alt+Shift+End	Extends the selected block to the end of the current line
Shift+Home or Alt+Shift+Home	Extends the selected block from the caret to start of text within the current line or extends the selection to the start of the line when pressed again (for example, to the start of any whitespace)
Shift+↓ or Alt+Shift+↓	Extends the selected block to the same column in the next line
Shift+↑ or Alt+Shift+↑	Extends the selected block to the same column in the previous line
Shift+Alt+arrow keys	Makes a rectangular selection
Alt+dragging the mouse	Makes a rectangular selection
Shift+Page Down	Extends the selected block down one screen (page)
Shift+Page Up	Extends the selected block up one screen (page)

Key Combinations	Action
Shift+Ctrl+←	Moves (extends) the selected block left one word of the caret position
Shift+Ctrl+→	Moves (extends) the selected block right one word of the caret position

Insert and Delete Shortcut Keys

The shortcut keys listed in the following table enable you to perform insert and delete functions in the editor pane by using the keyboard.

Key	Action
Backspace	Deletes the character to the left of the caret
DEL	Deletes the character to the right of the caret
Ctrl+Backspace	Deletes the word to the left of the caret, and deletes the word to the left of selected text if text is currently selected
Ctrl+Delete	Deletes the word to the right of the caret, and deletes the word to the right of selected text if text is currently selected
Ctrl+Insert	Copies the selected portion of the method to the clipboard
Ctrl+K	Deletes the current line
Ctrl+N	Inserts a line above the current line
Ctrl+S	Inserts the remainder of the syntax for a specified instruction
Ctrl+Shift+Backspace	Deletes to start of line
Ctrl+Shift+Delete	Deletes to end of line
Insert	Toggles the insert mode on or off
Shift+Alt+↓, Tab	Selects multiple lines (without selecting any characters) and then adds tab characters into the lines
Shift+Alt+↓, Backspace	Selects multiple lines (without selecting any characters) and then removes tabs, moving text back to the prior tab position.
Shift+Delete	Cuts (logically deletes) selected portion of method, and copies it to the clipboard
Shift+Insert	Pastes portion of method from the clipboard to the current position

Using the Mouse within the Editor Text Area

The mouse actions that you can perform in the text area of the **JadeTextEdit** control text editor are listed in the following table.

Mouse Action	Result
Left-click	Moves the caret to the cursor location and cancels the selection
Double-click	Selects the word at the cursor location
Triple-click	Selects the line at the cursor location
Left down and move	Anchors the selection at the down position and extends to follow the cursor location

Mouse Action	Result
Right-click	Opens the popup (context) menu
Shift+left-click	Moves the caret to the cursor location and extends the selection to the new location
Alt+left down and move	Anchors a rectangular selection at the down position and extends to follow the cursor location
Alt+dragging the mouse	Makes a rectangular selection
Ctrl+left-click	Adds a caret at the cursor location (use this action to add multiple carets)
Ctrl++left-click+dragging the mouse	Makes a selection (use this action to make multiple selections)

Using the Mouse or Shortcut Keys within the Fold Margin

The mouse or shortcut key actions that you can perform within the fold margin (visible if folding is enabled) of the editor pane are listed in the following table.

Mouse or Keyboard Action	Toggles (expands or contracts)...
Ctrl+left-click	The line if it is a fold point and makes all lower-level (child) fold points match.
Ctrl+Shift+left-click	The first outer-level fold point and makes all other outer-level fold points match.
Shift+left-click	All outer (parent) fold points.
Ctrl+Shift+ numeric keyboard * (asterisk)	All outer (parent) fold points.
Left-click	The fold point. If the line is not a fold point, toggles the parent fold point of the line.
Ctrl+ numeric keyboard * (asterisk)	The fold point.

Using the Mouse within the Line Number Margin

The mouse actions that you can perform within the line number margin (visible if the line number display is enabled) of the editor pane are listed in the following table.

Mouse Action	Result
Left-click	Selects the line at the cursor location
Ctrl+left-click	Selects all text in the editor pane
Shift+left-click	Extends the line-based selection from the current anchor point to the line at the cursor location

Using Multiple Clipboards when Refactoring Code

When you copy the same code fragments (snippets) into many different unrelated classes where the code cannot be inherited from a common superclass and you have more than one code fragment, multiple clipboards can be a useful refactoring tool. For details about the Editor Clipboard toolbar, see "Using the Editor Clipboard Toolbar", earlier in this chapter.

Using multiple clipboards is a two-stage key sequence similar to the standard Ctrl+C and Ctrl+V copy and paste actions, as follows.

To copy selected code to one of the 10 multiple clipboards

1. Press Ctrl+Alt+C and then release.
2. Press one of the numeric keys 1 through 0, to copy the current selection to one of the 10 clipboards.

To paste code from one of the 10 multiple clipboards

1. Press Ctrl+Alt+V and then release.
2. Press one of the numeric keys 1 through 0 to paste the contents.

These clipboards operate independent of the standard Ctrl+C and Ctrl+V clipboard actions and they work only within a specific Jade Platform development environment instance (that is, they do not work across different development environment instances).

Tips As you *do* have to remember the contents of each clipboard, you may find it easier to use a smaller number of clipboards (for example, four or five).

You can also use keyboard bindings to replicate this; that is, use Ctrl+Alt+Insert and Shift+Alt+Insert to extend the standard Ctrl+Insert and Shift+Insert copy and paste actions with the extended clipboards.

When working with long methods, you may find it useful to use the method splitter (Ctrl+Shift+W or just dragging the resize bar at the bottom of the method source) and then sizing the top area of the split so that all of the parameters and local variables are visible while reviewing the rest of the method logic.

Painter Shortcut and Function Keys

The shortcut and function keys listed in the following table are available from the Jade Painter.

Menu Item	Shortcut or Function Key
Save Form	F2
New Form	F3
Show Properties Dialog	F4
Show Control Hierarchy Dialog	F5
Translate Properties	F6
Edit Form	F12
Select All	Ctrl+A
Align Bottom	Ctrl+B
Edit Copy	Ctrl+C
Edit Control Palette	Ctrl+E
Find Control	Ctrl+F
Grid	Ctrl+G
Same Height	Ctrl+H
Add Control	Ctrl+Insert
Align Left	Ctrl+L
Menu Design	Ctrl+M

Menu Item	Shortcut or Function Key
Align Right	Ctrl+R
Standard Size	Ctrl+S
Align Top	Ctrl+T
Edit Paste	Ctrl+V
Same Width	Ctrl+W
Edit Cut	Ctrl+X
Undo Last Layout	Ctrl+Z
Push All Parents and Control to Bottom	Ctrl+Alt+B
Bring All Parents and Control to Top	Ctrl+Alt+T
Push To Bottom	Shift+Ctrl+B
Select All Children	Shift+Ctrl+C
Centre Horizontally	Shift+Ctrl+H
Select All Siblings	Shift+Ctrl+L
Tab Ordering	Shift+Ctrl+O
Select Parent	Shift+Ctrl+P
Same Height and Width	Shift+Ctrl+S
Bring To Top	Shift+Ctrl+T
Centre Vertically	Shift+Ctrl+V
Form Wizard	Shift+Ctrl+W

Applies to Version: 2018.0.01 and higher

Using Shortcut Keys in Controls

To use shortcut keys in controls in the Jade Platform development environment and in runtime Jade applications, press the Alt key and the appropriate shortcut (underlined) key to action the required control; for example, to click a button.

When focus is on a control that does not handle keyboard events (for example, it is not a text box, list box, or combo box) and you press only the specified shortcut key, that control is actioned. (In this case, you do not need to press the Alt key as well.)

Press Ctrl+Home to move a list box or table with an associated collection to the first entry in the collection. Conversely, press Ctrl+End to move the display to the last entry in the collection. Pressing the Home key on a list box or table moves the display to the first entry that has been loaded in the control. Pressing the End key moves the display to the last entry currently loaded in the control.

Using the Ctrl key for a list box or table that does not have a collection attached has the same result as pressing the Home or End key without the Ctrl key.

Tip When focus is on a control (that is, a control in a dialog or a menu command), press F1 to display online help for the control on which the caret is positioned.

When a **ListBox** control has focus, you can copy the contents of the list box to the clipboard, by using the Ctrl+C or Ctrl+Insert shortcut key combination. The copy action starts with the first entry currently displayed in the list box and ends with the last entry in the list box. A carriage return / line feed character (**CrLf**) is added to the end of each entry's text.

Note In the Jade Platform development environment, the Edit menu includes a Ctrl+C menu accelerator for the **Copy** command. The list box control will therefore not receive that form of copy request, which is sent to the editor pane. To copy the contents of a list box in this situation, use the Ctrl+Insert shortcut keys.

Filtering Entries in Combo Box and List Box Text

Some Jade development environment and Painter forms allow filtering of combo box and list box entries, to enable you to locate a required list item more quickly, in a similar way to the F4 (Find Type) functionality. This feature uses standard combo box and list box facilities to achieve this filtering, and applies to the following forms and controls.

- Painter Load Form dialog **Form** list box.

  When editing a form, the Load Form dialog provides the **Show as Hierarchy** check box, which defaults to **false** (that is, unchecked). When unchecked, the list is displayed alphabetically sorted and it allows the filtering process to be performed. When it is checked, the list is displayed in hierarchical form and the filtering process is not performed.

- Painter Find Control dialog **Control to Find** combo box.

- Define Reference dialog **Type** combo box (only for the current class on the left of the extended form of the dialog).

- Quick Navigation dialog **Class** combo box.

- Define Class dialog **Subclass of** combo box.

- Find Unused Class Entities dialog **Search selected class in selected schema** combo box.

- Find Unused Local Variables and Parameters dialog **Search selected class in selected schema** combo box.

- Find Possible Transient Class Leaks dialog **Search for creates of selected class in all schemas** combo box.

- Navigation bar in hierarchy browser combo boxes (that is, for classes, properties, and methods lists).

When you enter text in a combo box or the text box associated with a list box, the displayed list items are only those entries that contain the specified text (case-insensitive) or if the specified text is all uppercase, the entries with the same Pascal case (for example, **CA** matches **CustomerAddress**). Because the search process handles uppercase searches if all text entered after the first character is uppercase, the list includes all entries in which the first character of the entity name matches the first entered character and the entity name has uppercase characters that match the order of the remaining entered uppercase text. For example, entering **sNT** in the method combo box of the navigation bar includes method names **searchNameTextOnly** and **someNumberedTest**.

To select the entry that you want to locate in a list box (for example, in the Find Type dialog or using AutoComplete functionality in the editor pane), use the:

- Pascal-type uppercasing. For example, entering **XDG** includes any entries that begin with **X** and have the **D** and **G** as the next uppercase characters in the name (for example, **XamlDataGrid**). This also works where the first character is lowercase; for example, **iD** would include **includeDict**.

- Camel case filtering when the:
  - First sequence of uppercase characters of the list item matches the filter text in that order.
  - Uppercase sequence anywhere in the text of a list box item; for example, **CA** matches the entry of a **CMClass_Customer_Address** class. This avoids having to type any *XXXX* prefix you may have on all your class names, which reduces the number of characters that need to be typed to find a camel case match.

To select the entry that you want to locate in the combo box or list box, perform one of the following actions.

- Type at least the first few characters of the entity. The list display starts with the entity that matches your specified value. For example, if you have classes named **ProcessStackArray**, **ProdMaint**, **Product**, and **ProductDict**, if you enter **pro** in the **Find** text box, **ProcessStackArray** is then selected in the list box. However, if you enter **produ**, the **Product** class is then selected.
- When you enter text into text box, only those entities that contain the specified text somewhere in the type name of the entry are made visible in the list box. For example, if you specify **cust** when searching for a class, only those classes that contain **cust** somewhere in the class name are displayed. (The selection is case-insensitive.)

  The first entry that begins with the specified text is selected. If no entry starts with the specified text, no entry is selected. Use the up and down arrow keys to select and move through the entries in the list. The list box behaves the same as the **AutoComplete** list box. For details, see "AutoComplete List Box Behavior", elsewhere in this chapter.

You can achieve the same thing in your own Jade systems, as follows.

- Set the value of the **style** property for the **ComboBox** control class to **ComboBox.Style_DropDown** (0).
- The **ComboBox** or **ListBox** control class **hasPictures**, **hasTreeLines**, and **hasPlusMinus** properties are all set to **false**.
- The **sorted** property of the **ComboBox** or **ListBox** control class must be set to **false** after the list is loaded.
- A disabled **description** list item is added as the first entry in the list (after sorting is turned off).
- The **change** event on the combo box or associated text box calls a filtering routine to hide or show the entries based on the new text entered.

  Setting the level of the item to **2** hides the entries, which is why there has to be an entry at item position **1** that is always at level **1**. (A child item must have a parent.)

Note For a combo box, the **click** event is not normally implemented. The selection action is performed on the **closeup** event so that nothing happens until the user completes the selection action.

Applies to Version: 2020.0.01 and higher

Using Jade AutoComplete Functionality

The Jade AutoComplete feature provides the following.

- Reduces the amount of typing required to define a Jade method.
- A list of selectable suggestions for what is being entered in the context of the current expression.

  The list contents depend on the context, and can include Jade instructions, package names, global constants, classes, properties, methods, method options (for example, **updating** and **protected**), constants, and translatable strings.

Only the keywords that are valid in a method are listed as suggestions.

The Jade AutoComplete feature selects a matching entry out of the most-recently used list for the current context in the displayed list. The selection is case-insensitive, with the exception of the first character (the case of first character determines the types of entries that are displayed).

- The ability to investigate the type of any entity by hovering the mouse over an entity.

For details, see the following subsections.

AutoComplete Editor Options

The Preferences dialog provides the following options.

- The **Use AutoComplete** check box in the Auto Complete group box on the **Editor Options** sheet controls whether the feature is enabled. The feature is on (checked), by default.
- When the **Use AutoComplete** check box in the Auto Complete group box is checked, you can check the **Insert Parentheses for Method with no parameters** check box if you want parentheses automatically inserted for a method that has no parameters.

  The **Insert Parentheses for Method with no parameters** check box is unchecked by default. The opening and closing parentheses for a method call are automatically added to the text when the following are true.

  a. The **Insert Parentheses for Method with no parameters** check box is checked.

  b. A method name is selected from the auto complete list, by pressing a semicolon (**;**), dot (**.**), or space key.

  c. The method does not have any parameters.

  For example, if you type **customer.initi** and you select **initialized** from the auto complete list by pressing the **;** character, the result is:

  ```
  customer.initialized();
  ```

- When the **Use History for Selection** check box in the Auto Complete group box on the **Editor Options** sheet is checked, the prior selection (depth of 10) whose prefix matches the text entered in the editor pane is selected by default. If there is *no* prior history that matches the specified text, AutoComplete operates as though the value of the check box is **false** (that is, it is unchecked). For more details, see "Using the AutoComplete History", later in this chapter.

  When this check box is unchecked, the entry selected in the AutoComplete list is always the first name with the prefix matching the text entered in the editor pane. The history of previously-selected AutoComplete entries is not used, and you must type more of the identifier name to locate the required entry in the list box or select the required entry using an arrow key or the mouse.

- When the **Perform Method Error Analysis** check box in the Auto Complete group box on the **Editor Options** sheet is checked (the default), the Jade editor performs error analysis of the method definition as you modify the method, which makes you aware of issues before you compile the method.

  When an error is detected, the text in error is decorated with a red wavy underline symbol. Hovering the mouse over such text displays a bubble help entry with a brief description of the error.

  For details, see "Method Definition Error Analysis", later in this chapter.

- The **Ctrl+Space** key binding on the **Short Cut Keys** sheet is the key sequence to manually invoke the AutoComplete list box based on the current cursor location.

- The **Ctrl+R** key binding on the **Short Cut Keys** sheet is the key sequence to toggle the display of RootSchema entities in the AutoComplete list box. For details, see "Toggling the Display of RootSchema Entities", later in this document.
- The **Editor** sheet enables you to change the default color of instance methods, type methods, and properties in the **AutoComplete** list box from brown, faded blue, and light blue, respectively.

Displaying Identifier Usages from the Editor Pane

The Jade Platform development environment and editor pane enable you to select an identifier in the method source and request usages of the identifier by using the **Usages of *identifier*** command in the Edit context (popup) menu in the editor pane without having to find and select the item from a hierarchy browser and performing the same action.

Usages include references, property and method local references, property read and update references, method implementors, local implementors, and implementor references.

The cursor must be positioned in the identifier name with nothing selected or the whole identifier must be selected.

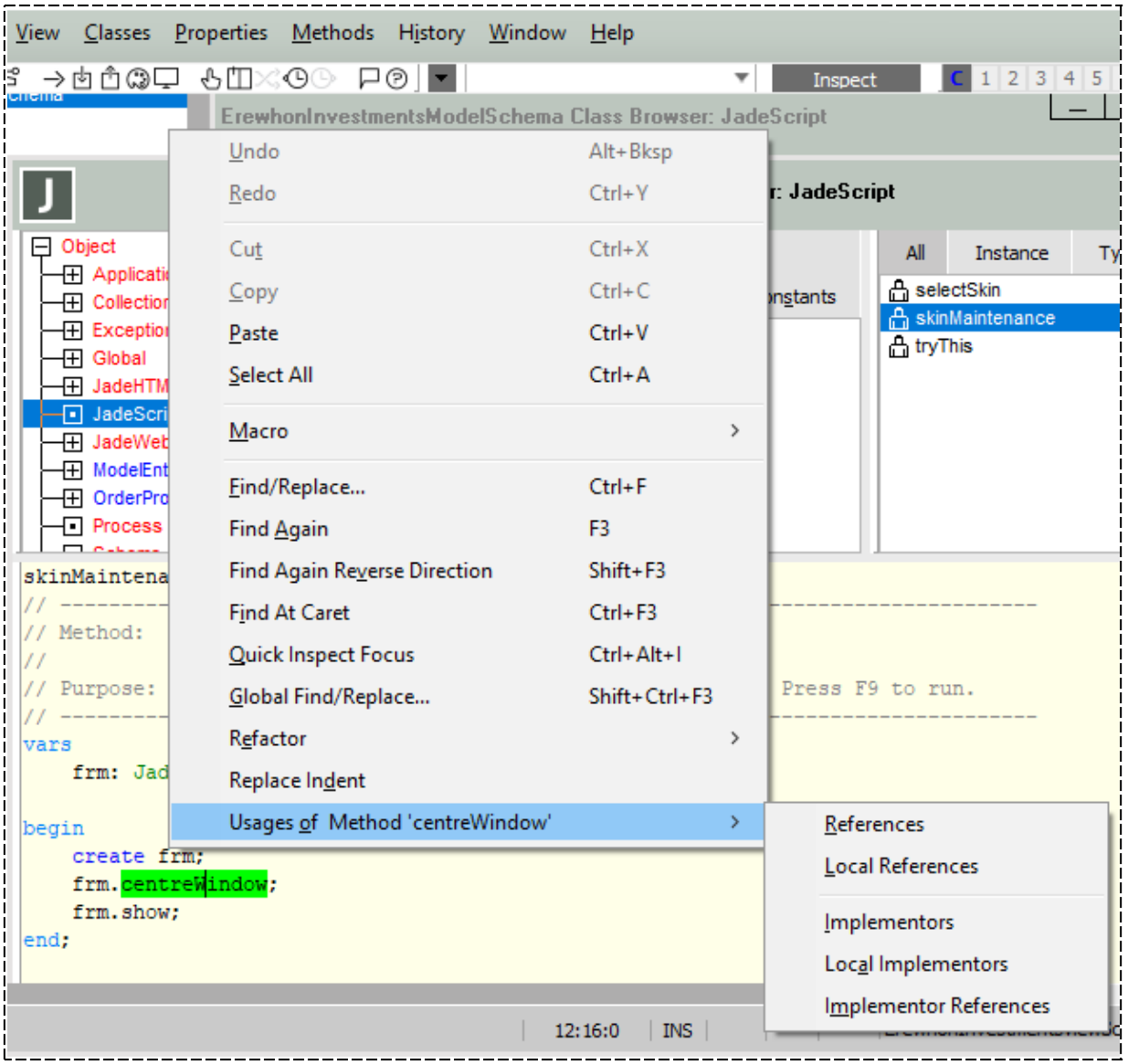
Note This functionality requires the Jade AutoComplete feature to be in use for the method (that is, the **Use AutoComplete** check box in the Auto Complete group box on the **Editor Options** sheet of the Preferences dialog is checked).

The identifier must be one of the following.

- Type name
- Class name
- Imported package class name
- Global constant
- Class constant
- Imported class constant
- Class property
- Imported class property
- Type or class method
- Imported class method

When the mouse is hovered over the method name at the start of the source, bubble help for the method includes the schema and class to which the method belongs.

Display the Edit menu by selecting the Edit menu in the menu bar or by right-clicking in the editor pane of the method, as shown in the following image. If the cursor is *not* positioned on one of the above types of identifier, the 'usages' menu item is disabled and the caption displays *Usages - no suitable identifier selected*.



If the selected identifier is one of the following, the menu command that is displayed is **References of *identifier-type identifier-name***; for example, **References of Class Customer**.

- Type name
- Class name
- Imported package class name
- Global constant

- Class constant
- Imported class constant

Selecting the menu command displays the references to that identifier in a hierarchy browser. If the identifier is a property or method, the displayed menu command has the caption **Usages of *identifier-typeidentifier-name***; for example, **Usages of Class Customer**.

Clicking the menu displays a submenu of the following options that depend on the type of identifier.

- **References**, for all property and method types
- **Local References**, for non-imported class properties and methods
- **Read References**, for property read references
- **Update References**, for property update references
- **Implementors**, for implementors of a method
- **Local Implementors**, for local implementors of a method if the schema type is a non-imported class
- **Implementor References**, for references to local implementors of a method if the schema type is a non-imported class

Clicking these commands performs the same actions as those when selecting the item in a browser list and then right-clicking and selecting the equivalent menu command.

If AutoComplete is not active, the **Usages of *identifier*** command is not displayed in the Edit menu.

Applies to Version: 2020.0.01 and higher

Contents Displayed in the AutoComplete List Box

The Jade AutoComplete list box entries are displayed with the colors defined for the editor in your user profile (that is, on the **Editor** sheet of the Preference dialog) so that the color of entities in **AutoComplete** list box entries is the same that of the entity displayed in the method source in the editor pane. In addition, instance methods in the **AutoComplete** list box are displayed in brown, type methods in faded blue, and properties in light blue, by default.

The color options allow different colors to be shown for system types and for user types (specified on the **Window** sheet of the Preferences dialog). Note, however, that these colors are the same, by default.

When entering text into a method, the **AutoComplete** list box is displayed automatically in the following situations.

Situation	List Box Contents
Entering an identifier and then **.**	All properties and methods for the type of the identifier. If the next character is uppercase, the list includes all classes, packages, package classes, primitive types, and constants for the type of the identifier.
Entering a class name followed by **.**	All properties, methods, and constants for the class as well as all properties, methods, and constants for the type of the class.

Situation	List Box Contents
Entering **$** followed by an uppercase character	The translatable strings for the current schema and superschemas, with the list trimmed to show only those strings that include the entered sequence.
Entering **.** after a translatable string	List of **String** primitive type methods.
Entering **.** after a name in a constant value	List of constants defined for the class or type.
First alphabetic character entered for an entity	All classes, packages, package classes, interfaces, primitive types, global constants, Jade instructions, system variables, and Jade method words. See also "Using the AutoComplete History", later in this chapter.
First character entered for a method parameter after **:**	All classes, packages, package classes, interfaces, and primitive types.
First character entered of the method return type after **:**	All classes, package classes, and primitive types.
First character entered for a variable type definition after **:**	All classes, packages, package classes, interfaces, and primitive types.
First character entered for a method option	All method options.
First character entered after a method parameter type	Valid parameter usage types (that is, **constant**, **input**, **io**, and **output**).
First character entered for a constant type definition after **:**	Primitive types.
Entering an identifier followed by **::**	If the identifier is a package name, the list of entities in the package is displayed. If the identifier is an interface name, the list of entities in the interface is displayed. If the identifier is a class or primitive type, the list of properties, methods, and constants defined for that type are displayed.
First character entered for a constant value after **=**	All classes, interfaces, and global constants.
Entering a **c** before the **vars** section	Single entry **constants**.
Entering a constructor with parameters	Signature of the constructor, to indicate which parameter you are editing (from version 2020.0.01 and higher).
Entering a character after a **break** or **continue** instruction	List of block labels, if any exist, for the loop (from version 2020.0.01 and higher).
Entering a character after an **endwhile** or **endforeach** statement	Label associated with the block at that level, if one exists (from version 2020.0.01 and higher).
Identifier followed by a Jade instruction	For example, if you are typing the **in** as part of **foreach indx in**, it displays the list of Jade instructions.
Next character entered after **call**	List of external function names.

For details about:

- Displaying dictionary keys, see "AutoComplete Dictionary Keys"
- Toggling the display of RootSchema entities in the AutoComplete list box, see "Toggling the Display of RootSchema Entities"

AutoComplete Dictionary Keys

The AutoComplete feature displays the required keys when a method has a **KeyType** parameter.

The displayed signature of the method is revised to display the keys of the referenced dictionary instead of the **KeyType** parameter when the cursor is positioned in the method parameters. For example, **dict.getAtKey(keys: KeyType)** is displayed as **dict.getAtKey(name: String; dob: Date)** where the keys of the dictionary are **name** and **dob**.

In addition, when hovering the mouse over the method name, the bubble help display shows the defined signature followed by **Required method call: *<revised method call with the key parameters>***.

For example:

```
(method) <schema-name><class-name>.getAtKey(keys: KeyType)
```

Required method call:

```
(method) <schema-name><class-name>.getAtKey(name : String; dob : Date)
```

This functionality also handles implied calls to **Dictionary** class **getAtKey** methods involving the*<key>***([..])** substring operator notation. In this case, the method signature of the **getAtKey** method is displayed showing the keys when the cursor is positioned inside the *<key>* field.

Use of **([..])** expressions for:

- **Dictionary** and **ExternalDictionary** types that actually mean a call to the **getAtKey** method display the signature required for the **getAtKey** call and highlight the current parameter that is being edited.
- **Array** and **ExternalArray** types that actually mean a call to the **at** method display the signature required for the **at** call and highlight the current parameter being edited.

Note, however, that if the collection type is generic (for example, **MemberKeyDictionary**), the **KeyType** parameter cannot be evaluated because the actual collection type is not known at compile time. For example, the following code fragment shows **KeyType** in the signature in AutoComplete.

```
method1(coll: MemberKeyDictionary);
begin
    coll.getAtKey("name", "value");
```

AutoComplete List Box Behavior

The displayed **AutoComplete** list box behaves by default as follows.

- Can be canceled by pressing the Esc key.
- Can be scrolled using the up (↑) and down (↓) arrow keys or the mouse. The up (↑) and down (↓) arrow keys wrap around; that is, when the:
  - First entry is selected and the up arrow is pressed, the last entry in the list box is then selected.
  - Last entry is selected and the down arrow is pressed, the first entry in the list box is then selected.
- A half-second after an entry in the list is selected, a bubble help window is displayed next to the list entry. The display includes the type and location of the entry and the first 20 lines of any user text.
- The case of the first character entered determines what entries are displayed. A lowercase first character displays only properties, methods, and Jade instructions. An uppercase first character displays only classes, types, packages, and constants.

- Entries in which the first characters match what has been entered are selected automatically. The display of entries is case-insensitive, with the exception of the first character (the case of first character determines the types of entries that are displayed).
- The list includes only those entries that have the entered character sequence somewhere in the list text.
- If no entry matches what is typed, the previous list is retained and nothing is selected.
- The more characters that are typed, the smaller the list becomes.
- Entries can also be selected by using the:
  - Pascal-type uppercasing. For example, entering **XDG** includes any entries that begin with **X** and have the **D** and **G** as the next uppercase characters in the name (for example, **XamlDataGrid**). This also works where the first character is lowercase; for example, **iD** would include **includeDict**.
  - Camel case filtering when the:
    - First sequence of uppercase characters of the list item matches the filter text in that order.
    - Uppercase sequence anywhere in the text of a list box item; for example, **CA** matches the entry of a **CMClass_Customer_Address** class. This avoids having to type any *XXXX* prefix you may have on all your class names, which reduces the number of characters that need to be typed to find a camel case match.
- Using the Backspace key reverts the list to match the entered identifier.
- The currently selected entry can be chosen by using the Tab or Enter key, or by clicking the list box entry with the mouse. Any entered text for this identifier is then replaced by the selected entry.
- An entry can also be selected by pressing a semicolon, comma, period, left parenthesis, right parenthesis, space, left square bracket, right square bracket, or colon key. The entered text is then replaced by the selected entry and the trigger character is added. However, if there is an entry in the list that exactly matches the identifier entered, that entry is used instead, even if it is not the selected entry.
- The last 20 AutoComplete selections are retained. Whenever a list is displayed that includes those entries and the entered text matches, an entry is automatically selected on the assumption that it is likely that logic will have multiple references to that entity.
- When you type a dollar sign (**$**), a list of the translatable strings for the current schema and superschemas is displayed. As you type more of the name, the list is trimmed to show only those strings that include the entered sequence.

  Notes For AutoComplete to continue to show the list of entries, the first character typed after the **$** must be uppercase.

  Entering a period (**.**) after a translatable string prompts you with a list of **String** primitive type method suggestions.
- If the entered text exactly matches the only entry left in the list, the list box is cancelled automatically.
- The type of an expression is determined and alternatives are displayed; for example:

```
(amethodCall() * 56 + 1)
```

For details about toggling the display of RootSchema entities in the **AutoComplete** list box, see "Toggling the Display of RootSchema Entities", later in this document. For details about using the last selection made whose prefix matches the text entered in the editor pane, see "Using the AutoComplete History", in the following subsection.

Using the AutoComplete History

When the **Use History for Selection** check box in the Auto Complete group box on the **Editor Options** sheet of the Preferences dialog is:

- Checked (it is unchecked by default), if there was a prior selection (depth of 10) whose prefix matches the text entered in the editor pane, it is selected by default. The word that was last selected is automatically highlighted. However, this can cause issues when you want to select an identifier that is a prefix of the selected word; for example, if the history was for **application** and you type **app**, pressing the **.** key results in **application** being inserted when you require the **app** system variable.

  If there is no prior history that matches the specified text, AutoComplete operates as though the value of the check box is **false** (that is, it is unchecked). For details, see "AutoComplete List Box Behavior", in the previous section.

- Unchecked (the default value), the entry selected in the AutoComplete list is always the first name with the prefix matching the text entered in the editor pane. The previous history of selected AutoComplete entries is not used and you must type more of the identifier name to locate the required entry in the list box or select the required entry using an arrow key or the mouse.

Applies to Version: 2018.0.01 and higher

From version 2020.0.01, when the **Use History for Selection** check box is checked, the AutoComplete feature selects an entry in the list when it is an exact (case-sensitive) match instead of the historical entry. For example, **app** is selected when you enter **app**. The list item **application** is selected when you enter **a**, **ap**, or **appl** in the editor pane.

Toggling the Display of RootSchema Entities

You can dynamically hide the display of RootSchema entities from the displayed list of possible AutoComplete entities, which makes it simpler to identify the required entity when it is known to be defined in a user schema, by reducing the available options in the list box.

The Ctrl+R shortcut key sequence for the **AutoCompleteToggleRootSchema** action in the table on the **Short Cut Keys** sheet of the Preferences dialog toggles the display of RootSchema entities in the **AutoComplete** list box.

Note This shortcut key sequence is ignored by the Jade editor unless the **AutoComplete** list box is visible. (In the Jade Platform development environment, the Ctrl+R shortcut keys initiate the Run Application dialog, and continues to do so if the **AutoComplete** list box is not visible.)

Every time the **AutoComplete** list box is initiated, RootSchema entities are included in the displayed list, if they are available. Each subsequent use of the Ctrl+R shortcut key sequence toggles the exclusion or inclusion of RootSchema entities; that is, properties, methods, constants, classes, external functions, and translatable strings (but not global constants) in the displayed list.

The first Ctrl+R sequence removes any RootSchema entities from the displayed list, the next restores them to the list, and so on.

Notes If there is no non-RootSchema entity left after the first use of the Ctrl+R sequence, the **AutoComplete** list box will close.

The next time the **AutoComplete** list box is displayed, the list will contain RootSchema entities regardless of the previous use of Ctrl+R; that is, the sequence takes effect for the currently displayed list only.

Method Definition Error Analysis

When the AutoComplete feature is in use, the Jade editor performs error analysis of the method definition as you modify the method, which makes you aware of issues before you compile the method.

When an error is detected, the text in error is decorated with a red wavy underline symbol. Hovering the mouse over such text displays a bubble help entry with a brief description of the error. This analysis is controlled by the **Perform Method Error Analysis** check box in the Auto Complete group box on the **Editor Options** sheet of the Preferences dialog. (This check box is checked, by default.)

The error analysis does not use the compiler and it runs on the client inside the editor.

The analysis detects most issues, including:

- Syntax errors
- Invalid local variables and constant definitions
- Invalid method and parameter options
- Unknown identifiers
- Invalid type comparisons and assignments
- Invalid parameter type passed to a method
- Invalid method parameter count
- Invalid use of parameters according to their declared usage; for example, assignment to a constant parameter
- Invalid calls to an updating method from a non-updating method
- Invalid statements; that is, those that do nothing (for example, **indx = 0;**)
- Unmatched block endings and break commands; for example, an **if** instruction without a matching **endif** instruction and **break** instruction not in a loop
- Invalid mixed expression types
- Unmatched parentheses
- Invalid type casts

The analysis does not check the following. (This list may grow in future, as other unhandled cases are identified.)

- Whether the method option is appropriate (except for **updating**); for example, **webService**, **unitTest**, and so on
- The class lifetime requirements for a **create** instruction

Note The impact on you should be minimal, with the exception being that the AutoComplete feature requires access to all of the entities referenced in the current page of logic. This may cause a longer lead time to retrieve these entities from the server if AutoComplete has not seen them before. Once those entities have been retrieved and cached on the client, the operation should not cause any further delays. As this functionality is on by default, if performance is ever an issue, turn off the user preference.

Method analysis is ignored for an external function definition and for a Workspace.

The method analysis is performed:

- After the editor pane gains focus and the method has been changed or it has compile errors.
- One second after you have changed a method and have paused typing.

- After you have changed the method using a selection from the **AutoComplete** list box.

Method Call Signature Handling

When the cursor is in the parameter section of method call, the method signature is displayed in a bubble help window, with the text of the current parameter displayed in red.

The method call signature is placed directly under the left parenthesis line. If the call logic spans multiple lines, when you move to a subsequent line that is still within the call signature, the display is placed above the line with the left parenthesis. If the display includes multiple lines of user text and there is insufficient room to fit the full display, the text is truncated.

From Jade 2020.0.02 and higher, this functionality also occurs for external function calls and translatable strings that have parameters. In addition, the parameter usages in the constant value are also highlighted in red as well as the parameter name definition itself.

Press the Esc key to close the bubble help window.

InheritMethod Handling

When the left parenthesis is typed for an **inheritMethod** call and there is nothing following that parenthesis on the same line, the parameters defined for the current method followed by **);** are added after the parenthesis.

Mouse Hover Investigation

To determine the type of any identifier, hover the mouse over the identifier. A bubble help window then displays the type and origin of the entry, together with up to the first 20 lines of user text.

Note The editor must have focus for this feature to function.

Manually Invoking the AutoComplete List Box

The AutoComplete feature is turned on by default when:

- A new user creates a profile.
- You are an existing user and you click the **Defaults** button on the Preferences dialog.

» To manually invoke the AutoComplete list box for an identifier

1. Position the cursor at the appropriate position.
2. Press Ctrl+space.

Note You can configure this sequence by using the **Short Cut Keys** sheet of the Preferences dialog.

The text to the left of the cursor is used to select the entries in the list box appropriate for the current content.

AutoComplete Caching

The AutoComplete process retrieves from the Jade database only the definitions of those entities referred to in the current expression. Once retrieved, those entries are held in a memory cache and are reused for each new situation.

The entries are retrieved again if they are modified by any user when referred to again by the AutoComplete processing.

If there are more than an internally defined number of classes loaded, the oldest used class definitions are discarded.

AutoComplete is versioning-aware. If the current schema selected is versioned, a versioned copy of the classes is created in the cache.

AutoComplete Persistent File

When the Jade client terminates and there were additions to the collected RootSchema class and type definitions, a persistent file of those RootSchema definitions is written into the **temp** directory of the Jade install directory. The file is named **JadeAutoCompleteCache.jac**.

When Jade is next initiated and AutoComplete is set up in an editor, that file is read and loaded. This process eliminates the need to re-retrieve those RootSchema entities from the database.

The file has the Jade version embedded in it, and if the version does not match the version of the current executable, the file is discarded.

AutoComplete Failures

If the **AutoComplete** list box fails to be displayed or it does not contain what is expected:

- Check that the syntax of the expression is correct.
- Is there a semicolon missing from the previous statement?
- Is an identifier in the expression unknown?

AutoComplete Limitations

The Jade AutoComplete feature has the following limitations.

- Entering a period after a double-quoted string or a multi-character single quoted string assumes the **String** primitive type.
- Entering a period after a single-quoted character assumes the **Character** primitive type.
- The Jade pseudo types of **InstanceType**, **MemberType**, and **SelfType** are handled, and offer AutoComplete suggestions as required.
- If the collection type is generic (for example, **MemberKeyDictionary**), the **KeyType** parameter cannot be evaluated because the actual collection type is not known at compile time. For example, the following code fragment shows **KeyType** in the signature in the AutoComplete feature.

```
method1(coll: MemberKeyDictionary);
begin
    coll.getAtKey("name", "value");
```

- The displayed method signature handles a **ParamListType** pseudo type parameter when highlighting the current parameter in the displayed method signature.

  Note, however, that if the first parameter of the method being called is a **ParamListType** and the terminating bracket of the call has not yet been entered, it is not possible to accurately determine which parameter is currently being entered when there is more than one parameter.

- AutoComplete does not offer any suggestions for the **is**, ***entry-point-name***, **in**, and ***library-name*** external method option values. (AutoComplete cannot filter or check the validity of method options, but only check that the option is in the list of possible options.)

Customizing the Layout of Hierarchy Browser Forms

You can dynamically customize the layout of Hierarchy Browser forms (for example, the Class Browser, to provide you with more flexibility when using larger monitors).

The Hierarchy Browser form enables you to:

- Drag and drop the four major panels (the Class List, Properties folder, Methods folder, and editor pane) into a new layout; for example, you could vertically dock the Class List, Properties folder, and Methods folder on the left of the form, with the editor pane occupying the full height of the right of the form.
- Float each of the four panels. When any of the panels is floated, the floating form has a caption, as follows.
  - Class List panel displays *schema-name* classes; for example, **DocEgSchema classes**
  - Properties folder panel displays *schema-name*::*class-name* properties; for example, **DocEgSchema::Reviewed properties**
  - Methods folder panel displays *schema-name*::*class-name* methods; for example, **DocEgSchema::Reviewed methods**
  - Editor pane displays *schema-name*::*class-name* editor; for example, **DocEgSchema::Reviewed editor**
- Set a changed layout as the default layout for the current schema. Each schema can have its own layout. This default layout applies only to new Hierarchy Browser forms opened for that schema.
- Set a changed layout as the default for all schemas. This default layout applies only to new Hierarchy Browser forms that are opened.

  Note The default layout for a specific schema overrides the default layout for all schemas.
- Clear a saved layout for a single schema or the default layout for all schemas.
- Restore the Jade default layout.
- Apply the saved layout for another schema to the current Hierarchy Browser.

From version 2020.0.01, you can save hierarchy browser form layouts that you created following the release of Jade 2018.0.01. Save layouts assigned to a specific schema and the default layout for all schemas to your Jade user preferences initialization file, which defaults to **JadeUserPreferences.ini**. Conversely, you can apply (import) a saved preferences initialization file. (For details about exporting and importing your preferences initialization file, see "Maintaining Miscellaneous Options", later in this chapter.)

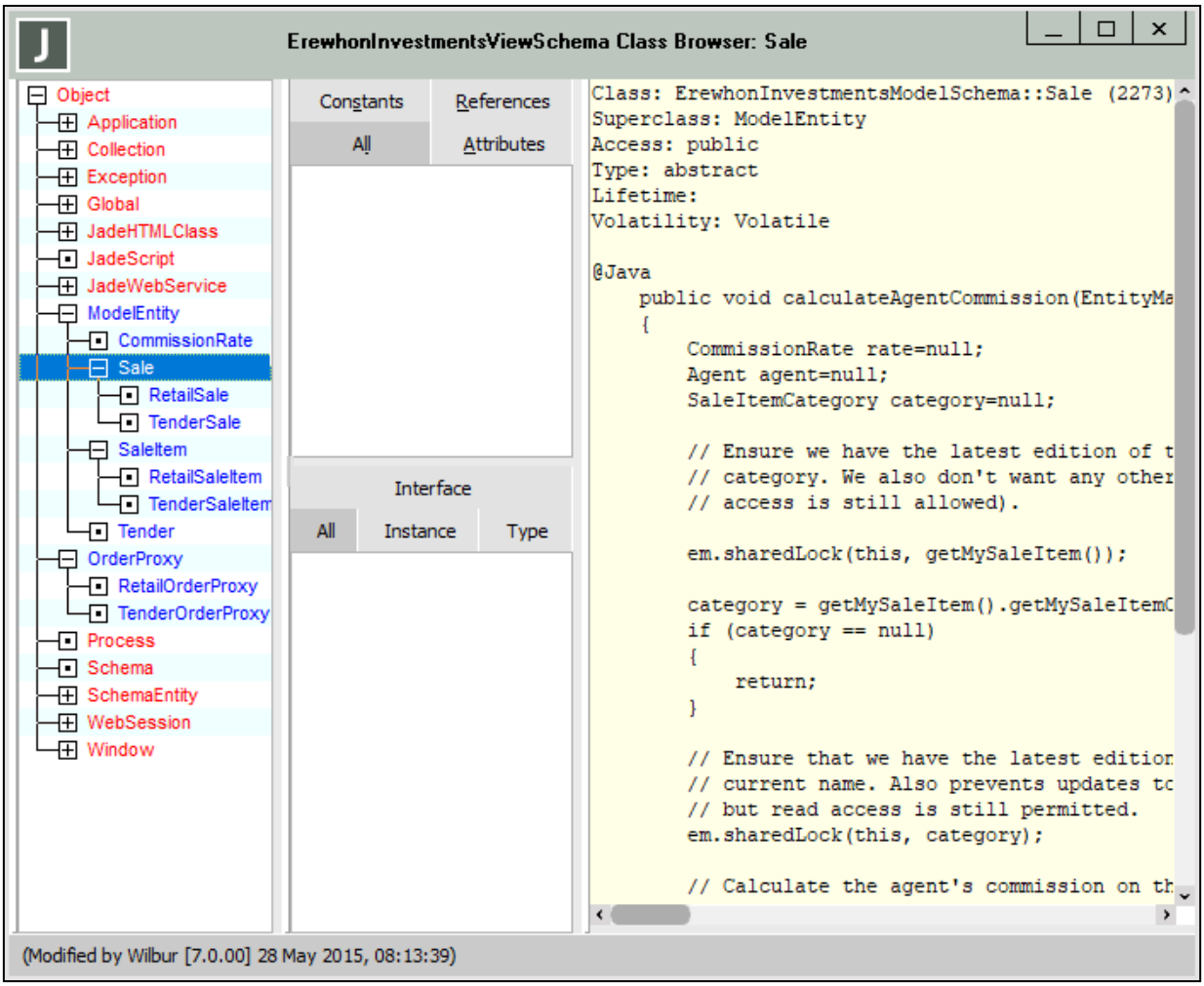
Loading a saved user preferences initialization file loads any saved layouts if the schema of the layout is present or if the layout is the default.

Note Loaded layouts replace any defined layout for the same schema or for the default layout.

Layouts for a schema that is not present are ignored. Existing layouts in your system are retained when there is no corresponding layout in your user preferences initialization file.

Loaded layouts are installed (applied) only when you click the **OK** button on the Preferences dialog after the preferences initialization file has been successfully read. Loaded layouts are not applied to any open window or form. You must select the layout that your require from the View menu **Layout** command submenu.

An example of a customized Hierarchy Browser layout is shown in the following image.



To change the layout, use the standard docking control drag and drop features, as follows.

- Left-click the docking control gripper bar for the panel to be moved and then drag the window to the position you require. When the mouse moves over a docking control gripper bar, an icon of a hand with a pointing finger is displayed. (This is standard for a dock control that can be dragged.)
- When the left mouse button is down, a drag rectangle is displayed in the area where the pane will be docked.
- To change the orientation (that is, flip vertical / horizontal orientation), press Shift while the left mouse button is down.
- To float the pane, press Ctrl while the left mouse button is down.
- Abort the drag process, by pressing Esc key before you release the left mouse button.
- To complete the position move, release the left mouse button.
- Use the panel size dividers to resize adjacent panels to meet your requirements.

Note the following:

- A panel can be docked only into its original form.
- If you have checked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog, the current layout where the panels have been moved is saved and restored, regardless of your setting of the **Save Size and Position** check box on the **Window** sheet of the Preferences dialog.
- If the layout was the Jade default, it is restored using the Jade Default layout.
- If you have unchecked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog (that is, the value is **false**), any altered layouts are lost. When you open a new Hierarchy Browser form and there is a default layout for that schema or for all schemas, that layout applies.
- The layouts are not saved when you export your user preferences.

Applies to Version: 2018.0.01 and higher

For details about the layout commands in the View menu, see the following subsections.

Using View Menu Layout Commands

The Hierarchy Browser View menu provides the **Layout** command, which has a submenu containing the commands listed in the following table.

Command	For details, see…
Apply Layout	Applying a Layout
Default Jade Layout	Applying the Default Jade Layout
schema-name	Applying the Layout of a Specific Schema
Clear the Default Layout	Clearing the Default Layout
Clear the Default Layout for Schema *current-schema-name*	Clearing the Layout of a Schema
Save as Default Layout	Saving the Default Layout
Save as Default Layout for Schema *current-schema-name*	Saving as the Default Layout for a Schema

For details, see the following subsections.

Applies to Version: 2018.0.01 and higher

Applying a Layout

Use the **Apply Layout** command from the View menu in a Hierarchy Browser to apply the default Jade layout or the layout of a specific schema. For details, see:

- Applying the Default Jade Layout
- Applying the Layout of a Specific Schema

Applies to Version: 2018.0.01 and higher

Applying the Default Jade Layout

Use the **Default Jade Layout** command from the **Apply Layout** submenu of the View menu in a Hierarchy Browser to restore the default Jade layout.

This command is disabled if the Jade layout is already the default. (Default means that the panels have not been moved, but they may have been resized.)

Applies to Version: 2018.0.01 and higher

Applying the Layout of a Specific Schema

Use the ***schema-name*** command from the **Apply Layout** submenu of the View menu in a Hierarchy Browser to apply the layout saved in that schema to the current Hierarchy Browser form.

Applies to Version: 2018.0.01 and higher

Clearing the Default Layout

Use the **Clear the Default Layout** command from the View menu in a Hierarchy Browser to discard the default layout saved for all schemas.

This does not affect any displayed layouts.

The command is disabled if there is no default layout for all schemas.

Applies to Version: 2018.0.01 and higher

Clearing the Layout of a Schema

Use the **Clear the Default Layout for Schema *current-schema-name*** command from the View menu in a Hierarchy Browser to discard the default layout for the current schema.

This does not affect any displayed layouts.

The command is disabled if there is no default layout for the current schema.

Applies to Version: 2018.0.01 and higher

Saving the Default Layout

Use the **Save as Default Layout** command from the View menu in a Hierarchy Browser to save the current layout as the default layout for all schemas. It does not affect the layout of any open forms.

The command is disabled if the current layout is the default Jade layout.

Saving as the Default Layout for a Schema

Use the **Save Default Layout for Schema *current-schema-name*** command from the View menu in a Hierarchy Browser to save the current layout as the default layout for the current schema. It does not affect the layout of any open forms.

The command is disabled if the current layout is the default.

Using Concurrent Windows

In the Jade Platform development environment, the schema of the window or browser that currently has focus is the *current* schema. (For example, although the **LockTest** schema may be selected in the Schema Browser, the **TestSchema** schema is the current schema if the **TestSchema** Class Browser is the current window; that is, it is the window that has focus.)

You can have concurrent open windows for different schemas, which enables you to view components of other schemas and to work concurrently in two or more schemas in the same Jade database. For example, you can browse the classes, properties, or methods in an existing schema while you define a new schema.

Although you can have a browser of each type (for example, a Maps, Format Browser, or an Application Browser) open for one or more schemas at any time in a work session, only one Schema Browser (and Painter window) can be open for the whole database, as the schema itself is development environment-dependent.

You can, however, have concurrent forms for different schemas open in the Painter window even though you can have only one Painter window open at any time.

In addition, you can have any number of the following types of browsers open concurrently for each schema, if required.

- Class Browser
- Primitive Types Browser
- Interface Browser
- Methods Browser

In summary, you can have:

- One Schema Browser only open at any time in a development environment work session
- Concurrent open browsers for different schemas in that session
- Concurrent open Class, Primitive Types, Interface, and Methods Browsers in a specific schema

Notes You can also have concurrent open Summary of Patches and Translator windows for different schemas in a development environment session.

Use the Schema menu **Close Windows for Schema** command to close all open windows for the current schema (that is, the schema of the window that currently has focus).

Using Window Menu Commands

Use the Window menu to arrange and manipulate the "child" windows in the Jade Platform development environment Multiple Document Interface (MDI) application.

The Window menu provides standard facilities that enable you to select tiling or cascading of windows in the Jade Platform development environment and displays a list of the currently enabled windows.

The lower portion of the Window menu lists all currently open windows (after these commands) in the order in which they were opened.

The current, or active, window is indicated by a check mark to the left of the window number and name.

Tip To access an open window, select the appropriate window from the list in the Window menu. Alternatively, you can use the Ctrl+F6 shortcut keys to cycle through all open MDI child windows.

To save your current settings of Class Browser windows when you terminate your work session by exiting from Jade, see "Maintaining Exit Options" under "Setting User Preferences", later in this chapter.

If you save your settings when exiting and you exit from the Jade Platform development environment in an orderly fashion (for example, by using the File menu **Exit** or **Logoff** command or the close icon at the top right corner of the window), your current Class Browser window settings are retained and displayed when you start the next Jade work session.

These settings are not retained if you exit from Jade abnormally (for example, you use the Windows Task Manager to end the Jade task or to terminate the **jade.exe** process.)

Use the Window menu commands to perform one of the actions listed in the following table.

Command	Description	For details, see…
Arrange Icons	Arranges all minimized icons in an orderly manner	Arranging Window Icons
Cascade	Arranges open windows in an overlapping pattern	Cascading Open Windows
Close All MDI Forms	Closes all open Multiple Document Interface (MDI) forms	Closing All MDI Forms
Tile Horizontal	Resizes and arranges windows horizontally without overlap	Tiling Windows Horizontally
Tile Vertical	Resizes and arranges windows vertically without overlap	Tiling Windows Vertically

For details, see the following subsections.

Arranging Window Icons

When you minimize a window, it becomes an icon. You can use the mouse to drag icons individually or you can use the **Arrange Icons** command from the Windows menu to arrange all minimized icons in an orderly manner across the bottom of the screen.

To arrange minimized icons

- Select the **Arrange Icons** command from the Window menu.

All minimized window icons are then arranged in an orderly manner across the bottom of the screen.

Cascading Open Windows

Use the **Cascade** command from the Window menu to arrange open windows in an overlapping pattern so that the title bar of each window remains visible.

To cascade open windows

- Select the **Cascade** command from the Window menu.

All open windows are then overlapped so that the title of each window is visible.

Closing All MDI Forms

Use the **Close All MDI Forms** command from the Window menu to close all open Multiple Document Interface (MDI) forms.

To close all open MDI forms

- Select the **Close All MDI Forms** command from the Window menu.

A confirmation dialog is then displayed, prompting you to click **Yes** to proceed with the closing of all MDI forms or **No** to abandon the request.

When you have confirmed that you *do* want to close all open MDI forms, they are then closed as if you have selected the **Close** button on each MDI child form.

Note If the MDI frame has no visible and enabled MDI child windows displayed, the **Arrange Icons**, **Cascade**, **Close All MDI Forms**, **Tile Horizontal**, and **Tile Vertical** commands are disabled.

Tiling Windows Horizontally

Use the **Tile Horizontal** command from the Window menu to resize and arrange open windows without overlap so that each window is wider than it is long and all windows are visible.

To tile open windows horizontally

- Select the **Tile Horizontal** command from the Window menu.

All open windows are then sized so that they are arranged horizontally.

Tiling Windows Vertically

Use the **Tile Vertical** command from the Window menu to resize and arrange windows without overlap so that each window is longer than it is wide and all windows are visible.

To tile open windows vertically

- Select the **Tile Vertical** command from the Window menu.

All open windows are then sized so that they are arranged vertically.

Setting User Preferences

Use the Options menu **Preferences** command from any browser window to maintain your global browser options for the schema and any subschemas. These global options apply only to you, and not to other users of the Jade Platform development environment. (For details about specifying preferences that apply to all Jade browser and Painter windows in the Jade Platform development environment work sessions for all new users, see "Specifying Your Jade Installation Preferences" under "Specifying Your Administration Options", in Chapter 2 of the *Installation and Configuration Guide*.)

The types of options that you can maintain in the Jade Platform development environment are as follows.

- Accelerator keys
- Browser
- Compiler output
- Editor
- Editor Options
- Exit
- Lock
- Message Box Suppression
- Miscellaneous

- Relationship
- Schema
- Shortcut keys
- Source Management
- Status list
- Text templates
- Window

To maintain your global options

1. Select the **Preferences** command from the Options menu. The Preferences dialog is then displayed. Option types are contained in sheets, with the **Editor** sheet displayed by default.
2. Select the type of options that you want to change, by clicking on the required sheet; for example, click the **Status List** sheet if you want to view or change your status list preferences.
3. Make the required changes.
4. Click the **OK** button to save your options. Alternatively, click the **Cancel** button to abandon your changes.

The AutoComplete feature is turned on by default when you click the **Defaults** button.

For more details about changing your user preferences or administrative installation preferences that are common to those of individual users, see the following subsections.

Maintaining Accelerator Keys

You can define your own accelerator (short cut) keys for use in the editor pane. For example, you could define the **R** key to insert **return** at the caret position when you press Shift+Ctrl+R.

For details of the accelerator keys that are defined by default, see "Editor Pane Shortcut Keys", earlier in this chapter.

To maintain accelerator keys for the editor

1. In the Preferences or Jade Installation Preferences dialog, click the **Accelerator Keys** sheet.
2. In the list box at the left of the sheet, select the letter to which you want to define an accelerator key or whose existing definition you want to change. (Letters that are reserved for system use are disabled; for example, **H** and **I**.)
3. In the editor pane at the right of the sheet, specify the text that you want to define for the selected letter; for example:
   - Instruction
   - Method option
   - System variable
   - Expression

- Statement
- Comment

For example, select the letter **F**, and then specify **foreach** in the editor pane.

4. Click the **OK** button.

When you next access the method editor pane, to insert a value defined for an accelerator key at the caret position, press Shift+Ctrl+*x* (where the *x* value is the appropriate accelerator key).

The following table lists examples of the use of accelerator keys.

Key	Keystrokes	Example Inserted Text
F	Shift+Ctrl+F	**foreach**
G	Shift+Ctrl+G	global.myCompany :=
P	Shift+Ctrl+P	app.printer
Z	Shift+Ctrl+Z	// Method modified by A. Lackey following code review

Maintaining Browser Options

In Jade Class Browser, Primitive Types Browser, and Interface Browser windows, you can specify the display and order of objects and the components of the window. For example, by default, the objects are displayed in hierarchical order.

To change your default browser options

1. In the Preferences or Jade Installation Preferences dialog, click the **Browser** sheet.
2. If you want the browser windows to display properties, methods, and constants inherited from superclasses, check the **Show Inherited** check box in the Inheritance group box. As this setting is a default only, you can override it by selecting the **Show Inherited** command from the View menu when a browser window is open.
3. If you want classes in the browser window displayed in alphabetical order, click the **Alphabetical** option button in the Order group box. As this setting is only a default, you can override it by selecting the **Sorted Order** command from the View menu when a browser window is open. See also "Changing the Sort Order of Displayed Classes", in Chapter 3.
4. If you do not want the browser windows to display an integrated editor pane, uncheck the **Integrated Editor** check box in the Window group box.
5. If you do not want the browser windows to display the Properties List, uncheck the **Show Properties** check box in the Window group box.
6. If you do not want the browser windows to display the Methods List, uncheck the **Show Methods** check box in the Window group box.
7. If you do not want the properties, constants, and methods that have public access displayed in the appropriate list of browser windows, uncheck the **Show Public** check box in the Access group box.
8. If you do not want protected access properties and methods displayed in the appropriate list of browser windows, uncheck the **Show Protected** check box in the Access group box.
9. If you do not want read-only access properties displayed in the Properties List of the Class Browser, uncheck the **Show Read Only (Properties)** check box in the Access group box.

10. If you do not want to display any of the icon symbols that indicate the access type or status of constants, properties, and methods in the Methods List, Properties List, and Constants List of the Class Browser, Primitive Types Browser, and Interface Browser, check the **Do not show Icons in Browser Window** check box.

    Icons are displayed in browser list windows by default. For details, see "Toggling Property, Method, and Constant Access Type Display", in Chapter 3.

11. If you do not want to display the most-recently accessed property and types entities in the history list, check the **Consolidate History to show methods only** check box. Type, property, and method entities are still added to the history list but accessing a method removes any existing type and property entities for a type. Subsequent type and property entries are added but a method entry again removes them.

    Method, property, and type entities are displayed in the history list windows by default. For details, see "Displaying or Clearing a History of Accessed Entities", earlier in this chapter.

12. If you want to display the navigation bar in hierarchy browsers (for example, the Class Browser), check the **Show Navigation Bar** check box. The navigation bar contains three check boxes, to enable you to select a schema, class, and method and refresh the display. For example, selecting a schema refreshes the display to that schema (as though a new browser was opened) and selecting a class selects the class (as though a class was selected using F4).

    The navigation bar is displayed in browsers, by default. For details, see "Using the Navigation Bar in Hierarchy Browsers", earlier in this chapter.

13. If you do not want forms to be displayed in the default standard Multiple-Document Interface (MDI) style, select the required option button in the Mdi group box. When you select:

    - **Use Mdi**, standard MDI forms are displayed.
    - **Use Mdi With Tabs**, a line of tabs is displayed above the MDI client window (the default value).

      Each tab is associated with a displayed MDI child form and displays the form's caption. Clicking on that tab brings the associated form to the front. The MDI child forms can still be maximized, restored, and minimized within the MDI client area.
    - **Use Tabs Only**, a line of tabs is displayed above the MDI client window.

      Each tab is associated with a displayed form and displays the form's caption. Clicking on that tab brings the associated form to the front. The forms are always maximized and cannot be restored or minimized. The parent of these forms is the associated tab sheet; not the MDI client window.

For the MDI with tabs and the tabs-only styles:

- Each tab displays a close button unless the **Form** class **allowClose** property is set to **false** (as is always the case for the Schema Browser). Clicking on the close button at the right of the tab closes the form.
- The width of each tab is restricted. If the caption is too large to be displayed, the left and right parts of the caption are displayed, separated by points of ellipsis (**...**).
- Class Browser tabs exclude the **Class Browser** part of the tab's caption, and display only the *schema-name*: *class-name* part.
- Moving the mouse over the tab displays a bubble help window containing the full form caption.
- Only those tabs that fit within a single line are displayed.
- The right of the tab strip displays a down arrow, which when clicked, displays a menu list of the MDI child forms in alphabetic caption text order with the currently active MDI child form checked. Clicking on a list entry brings that form to the top.
- You can alter the order of the tabs by clicking the tab of the active child form and dragging it left or right.
- Right-clicking on a tab displays a menu that provides the following commands.
  - **Close**, which provides the ability to close the form, and which is available only when the value of the **allowClose** property is set to **true**.
  - **Close All But This**, which closes all MDI child forms that have the **allowClose** property set to **true** except for the form associated with the tab.
  - **Close All But Pinned**, which closes all MDI child forms that have the **allowClose** property set to **true** except for those that have been pinned.
  - **Dock**, which docks a floating MDI child form back into the MDI frame. The docked position will be the saved position prior to the form being floated, unless the form was floated as part of the development environment restore process, in which case it will be docked at position 0,0.
  - **Float**, which floats an MDI child form; that is, it takes an MDI child form out of the MDI frame and allows it to be moved independently from the MDI frame (for example, on to another monitor on the PC).

    The floated MDI child form is made a Window's child of the frame and will then always be above the MDI frame in z-order. The menus associated with the form when active remain in the MDI frame.
  - **Pin**, which pins a tab by placing it to the left of all unpinned tabs in the tab list. The tab has a pinned icon drawn on the left of the tab. Clicking on the checked **Pin** command in the popup menu or clicking on the pinned icon unpins the tab.

    Pinned tabs can be dragged only to a position within the pinned tab list. Similarly, an unpinned tab can be dragged only to a position within the unpinned tab list.

Notes For those MDI styles with tabs, right-clicking on the caption of an MDI child form displays the same menu as the one displayed when the tab is right-clicked. For the default standard MDI style, right-clicking the MDI child form caption displays a similar menu that provides all commands other than **Pin**. Standard MDI-style child forms can still be floated and docked. (For details about programmatically providing the functionality to float, dock, and pin MDI child forms in your own application logic, see "Floating and Docking MDI Child Forms", in Volume 3 of the *Encyclopaedia of Classes*.)

The current MDI style is saved when the development environment is closed and restored when it is next initiated. In addition, when the **Save Windows** check box on the **Exit** sheet of the Preferences dialog is checked, the current order and pinned statuses of the saved MDI child forms is saved and will be restored when the development environment is next initiated. If a saved MDI child form is floating, it will be restored in its float state and position.

14. If you want the Window menu to list MDI windows in the order in which the MDI child forms were last used (that is, the most-recent use through to the oldest use), select the **Last Use Order** option button in the Mdi Window List Order group box.

    By default, windows are listed in creation order; that is, the order in which open MDI child forms were created (the oldest through to the most recent).

    The MDI child form order is not affected by the MDI style setting, specified in step 12 of this instruction.

    (For details about programmatically providing the functionality in your own application logic for tabs on MDI child forms and specifying the list order, see "MDI Child Form Tabs", in Volume 3 of the *Encyclopaedia of Classes*.)

15. If you want to change the toolbar icon size, in the Toolbar Icon Size group box, select the size of icons on Jade Platform development environment toolbars. The size options are **Small** (16x16 pixels), **Medium** (32x32 pixels), and **Large** (48x48 pixels).

    The toolbar icon size affects the following toolbar icons.

    - Development environment toolbar
    - Painter toolbars
    - Painter Properties dialog
    - Painter Hierarchy for Form dialog
    - Debugger toolbar

    When you change the icon size and click the **OK** button, the new icon size is immediately applied to all open relevant windows except for the Jade Debugger, where the option is loaded only at debugger initiation.

Tip You can also change the toolbar icon size from a Hierarchy Browser View menu, by selecting the **Toolbar Icon Size** command, which displays the submenu containing the **Small Icons**, **Medium Icons**, and **Large Icons** commands. A checkmark indicates the size that is currently selected. For details, see "Changing the Size of Toolbar Icons", earlier in this chapter.

16. In the Class Browser Options group box:

    a. Select one of the following list box items to specify whether multiple sheets are enabled in the editor pane and, if so, where the tabs for the sheets are located.

       - **Top**, to display the tabs at the top of the editor pane.
       - **Bottom**, to display the tabs at the bottom of the editor pane.

- **Hidden**, if you do not want the editor pane to display multiple sheets.

b. Check the **Preserve Selection** check box to preserve the currently selected attribute, reference, constant, or method when switching sheets. This check box is not checked, by default.

Note The settings in the Class Browser Options group box apply to hierarchy browser forms. For details about using multiple sheets in the editor pane, see "Displaying Multiple Sheets in the Editor Pane", in Chapter 3.

17. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Note As the settings in the Inheritance, Window, and Access group boxes are only default values, you can override them for the current browser by selecting the appropriate command from the View menu.

The current window then has focus. Your selected preferences are displayed by default when you next open a browser window.

Maintaining Compiler Output Options

Use the **Compiler Output** sheet to change the default behavior of the Compiler Output Viewer.

To change the Compiler Output Viewer default options

1. Click the **Compiler Output** tab of the Preferences or Jade Installation Preferences dialog. The **Compiler Output** sheet is then displayed.
2. In the table of compiler messages, perform the following actions for each message, as required. In the:
   - **Suppressed** column, check the check box if you want to suppress the message from being produced in the system. (This column is not available on the Preferences dialog of an individual user, as it is set by the administrator.)
   - **Filtered** column, check the check box if you want to filter the message from being displayed by default on the Compiler Output Viewer. (This column is available on the Preferences dialog only, as it is user-specific.)

     Note Messages that are suppressed by the administrator have a light-red background and do not display a check box in the **Filtered** column.

   - **Active Level** column, select the active level of the message; that is; **Warning** or **Info**. The active level determines whether the message is displayed as a warning message or an information message on the Compiler Output Viewer. The **Default Level** column displays the system default value. (The **Active Level** column is read-only on the Preferences dialog of an individual user, as it is set by the administrator.)

   For details about the messages, see "Compiler Output Messages", in Chapter 4.
3. To log compiler messages to a log file, check the **Log to File** check box. Messages are logged to the **compiler.log** file in the **logs** folder of your Jade installation. By default, this check box is not checked; that is, compiler messages are not logged to the **compiler.log** file. (This option is not available on the Preferences dialog of an individual user, as it is set by the administrator.)
4. To reset the Compiler Output Viewer preferences options to those set by your administrator or to the Jade system default values if you *are* the administrator, click the **Reset** button.

5. To specify the threshold of compiler messages that are displayed on the Compiler Output Viewer, select one of the following option buttons in the Compiler Message Threshold group box. Select the:

   - **All** option button, to display all warning and error messages.
   - **Warnings** option button, to display only warning messages.
   - **Errors** option button, to display only error messages.

   This option is read-only on the Preferences dialog of an individual user, as it is set by the administrator.

6. If you do not want to automatically display the Compiler Output Viewer if there are messages after a compile, uncheck the **Show on Compile** check box. By default, this check box is checked; that is, the Compiler Output Viewer is displayed after a compile if there are no suppressed messages. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

   Note If the **Show on Compile** check box is checked and a compile results only in suppressed or filtered messages, or the messages do not meet the compiler output threshold specified by the administrator, the Compiler Output Viewer is not displayed and focus is not set (even if **Set Focus on Compile** is selected).

7. To always set focus on the Compiler Output Viewer after a compile that results in no suppressed messages, check the **Set Focus on Compile** check box. By default, this check box is not checked; that is, the existing active window retains focus unless the Compiler Output Viewer is being created. After the Compiler Output Viewer is created, additional compiles will not set focus again if this check box is not checked. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

8. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Defaults** button to reset all preferences to the Jade default values.

Applies to Version: 2022.0.01 and higher

Maintaining the Editor Display

In the editor pane, the various components of your methods are displayed in different colors by default. For example, all Jade language instructions (keywords) are displayed in blue in the editor pane and operators (that is, **and**, **or**, and **not**) are displayed in black. You can use the Ctrl+0 shortcut keys in the editor pane to toggle the display of multiple-color or single color method components.

To change your default editor display

1. In the **Editor** sheet of the Preferences or Jade Installation Preferences dialog, double-click the component whose default color you want to change. For example, double-click the blue panel to the right of **Keywords** in the Editor list to change the default color for instructions in your Jade methods. The common Color dialog is then displayed.

   For details about the Jade installation default display of Jade elements in the editor pane, see "Using the Editor Pane", in Chapter 3.

2. Click on the required color. The selected color is then outlined in black. (This common dialog also enables you to define your own custom color, if required.)

3. Click the **OK** button. The Preferences or Jade Installation Preferences dialog is then redisplayed, with your selected color displayed beside the component (for example, **Keywords**, or Jade language instructions) in the Editor list and in the editor pane that displays a sample method.

   To make all of your required color display changes, repeat steps 1 through 3.

4. If you want all components of your methods to be displayed in a single color (black), check the **Single Color** check box.

   Tip The single color display provides improved performance when using the editor.

5. If you want to change the default tab stops position in Jade methods from **4**, specify the appropriate value in the **Tab Stops** text box. (The maximum value is **64**.)

6. Click the **Font** button if you want to change the default Jade methods font from Courier New 9 point. The common Font dialog is then displayed, to enable you to make your font selections. The bold and italic features are ignored and you can select a monospaced font (for example, Courier New) or a proportional font (for example, System).

   When you have made your font selections, focus is then returned to the Preferences or Jade Installation Preferences dialog.

7. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The editor pane in the browser window then has focus, and your selections are displayed in the editor pane of the current method.

Maintaining Editor Options

You can configure the Jade editor pane with the settings that you require for the Jade Platform development environment. For details about the Jade Platform installation default display of Jade elements in the editor pane, see "Using the Editor Pane", in Chapter 3.

Use the **Editor Options** sheet to change the default editor pane behavior.

» To change the default editor options

1. Click the **Editor Options** sheet of the Preferences or Jade Installation Preferences dialog. The **Editor Options** sheet of the Preferences dialog is then displayed.

2. In the Entry Options group box, check the:

   a. **Use Smart Indent** check box if you want the caret indented one tab position when a new line is inserted and the previous non-blank line begins with one of the Jade smart words. The Jade *smart* words are **constants**, **vars**, **begin**, **epilog**, **if**, **elseif**, **foreach**, and **while**.

   b. **Use Tabs For Indent** check box if you want to insert tabs for indenting (the default value). When you uncheck this control, the appropriate number of spaces is inserted instead of tab characters for editor pane indentation.

3. In the Folding Options group box, select the **Normal** or **Compact** option button if you want the option to hide lines in the editor pane by using fold points. By default, the **Normal** folding option is selected.

   Select the **Normal** option button if you do not want folding to include the terminating line or the **Compact** option button if you want folding to include the terminating line. When you select normal or compact folding, folded lines are indicated by a line drawn below the folded line and a square symbol in the left margin of the line, which you can use to make folded lines in the editor visible or invisible.

   For details about folding multiple-line comments, see "Using Comments in Your Methods", in Chapter 1 of the *Developer's Reference*.

4. In the Auto Complete group box, check the:

   a. **Use AutoComplete** check box if you want to enable the Jade AutoComplete functionality, which is on by default. AutoComplete functionality is turned on (that is, the check box is checked) when you click the **Defaults** button on the Preferences dialog.

      The Jade AutoComplete functionality:

      - Reduces the amount of typing required to define a Jade method.
      - Provides a list of selectable suggestions for what is being entered in the context of the current expression. The list contents depend on the context, and can include Jade instruction, package names, global constants, classes, properties, methods, and constants.
      - Provides the ability to investigate the type of any entity by hovering the mouse over an entity.

      For details, see "Using Jade AutoComplete Functionality", earlier in this chapter.

   b. **Insert Parentheses for Method with no parameters** check box if you want parentheses automatically inserted for a method that has no parameters.

      The opening and closing parentheses for a method call are automatically added to the text when the following are true.

      i. The **Insert Parentheses for Method with no parameters** check box is checked.

      ii. A method name is selected from the auto complete list, by pressing a semicolon (**;**), dot (**.**), or space key.

      iii. The method does not have any parameters.

      For example, if you type **customer.initi** and you select **initialized** from the auto complete list by pressing the **;** character, the result is:

      ```
      customer.initialized();
      ```

   c. **Use History for Selection** check box if you want a prior selection (depth of 10) whose prefix matches the text entered in the editor pane selected by default. If there is *no* prior history that matches the specified text, AutoComplete operates as though the value of the check box is **false** (that is, it is unchecked). For more details, see "Using the AutoComplete History", earlier in this chapter.

      When this check box is unchecked, the entry selected in the AutoComplete list is always the first name with the prefix matching the text entered in the editor pane. The history of previously-selected AutoComplete entries is not used, and you must type more of the identifier name to locate the required entry in the list box or select the required entry using an arrow key or the mouse.

   d. **Perform Method Error Analysis** check box if you want the Jade editor to perform error analysis of the method definition as you modify the method, which makes you aware of issues before you compile the method. This check box is checked, by default.

      When an error is detected, the text in error is decorated with a red wavy underline symbol. Hovering the mouse over such text displays a bubble help entry with a brief description of the error.

      For details, see "Method Definition Error Analysis", earlier in this chapter.

5. In the Display Options group box, check the:

   a. **Caret Finder** check box if you want to locate the caret in the editor pane. When the **Caret Finder** check box is checked and you press Ctrl, a large vertical rectangle of the defined caret color then flashes twice at the caret position. By default, the caret finder is not enabled.

b. **Use Wrap for Text Windows** check box if you want lines of text that exceed the viewable screen to be wrapped in windows that display the **Text** value of classes, properties, and so on. By default, lines of long text are not wrapped.

   When you check this control, lines are broken after space or tab characters. However, if a word is wider than the editor pane, the break occurs after the last character that completely fits on the line. When wrap mode is enabled, the horizontal scroll bar is not activated.

   New line characters are not inserted in the text at the wrapping point. Wrapping brings into view the right-hand side of long lines that are normally outside the display area.

   When you enable text wrapping and you want an indication when a line reaches a specified length (for example, as a format convention that limits lines to 80 characters), see the edge mode options in step 5 of this instruction.

c. **Use Wrap to Source Windows** check box if you want lines of source code that exceed the viewable screen to be wrapped in windows that display source code. By default, lines of long source code are not wrapped.

   When you check this control, lines are broken after space or tab characters. However, if a word is wider than the editor pane, the break occurs after the last character that completely fits on the line. When wrap mode is enabled, the horizontal scroll bar is not enabled.

   New line characters are not inserted in the code at the wrapping point. Wrapping brings into view the right-hand side of long lines that are normally outside the display area.

   When you enable code wrapping and you want an indication when a line reaches a specified length (for example, as a format convention that limits lines to 80 characters), see the edge mode options in step 6 of this instruction.

d. **View Line Numbers** check box if you want to display line numbers for each line in the editor pane By default, line numbers are displayed.

e. **View End Of Line Characters** check box if you want to display end of line characters in the editor pane. By default, end of line characters are not displayed.

f. **View Whitespace** check box if you want to display spaces as small centered dots and tabs as arrows. By default, whitespace is displayed as an empty background color.

g. **View Indent Guidelines** check box if you want to display vertical indentation guidelines in the editor pane. By default, indentation guidelines are not displayed.

h. **View Status Line Info** check box if you want to show editor information in the status line of the background window. By default, **View Status Line Info**, editor information is displayed in the status line.

   When this check box is checked, the editor pane information in the status line of the background window displays a field made up of the line number in which text is selected, the position of the first selected character, and the number of selected characters; the row and column positions; **Insert** or **Overtype** mode; and whether **Caps Lock**, **Num Lock**, and **Scroll Lock** are active.

i. **View Caret Line Color** check box if you want to highlight the line in which the caret is positioned with the color specified for the caret line in the editor pane. By default, the caret line is not highlighted.

j. **Only highlight whole words matching selection** check box (from version 2018.0.01 and higher) to specify if only whole words that match your selected text are highlighted, as follows.

   - Unchecked (the default value), any other text matching the case-sensitive selection is also highlighted using the **Additional Selections** color specified on the **Editor** sheet of the Preferences dialog (lime green, by default), unless the selection contains *space-type* characters; for example, selecting the word **to** highlights any occurrence of **to** in the editor pane.

- Checked, any selection causes other text matching text in the editor pane to be highlighted if the selection is a full-word identifier and there are other occurrences of that word in the editor pane. This match is case-sensitive and full-word; for example, selecting the word **caption** highlights any other occurrence of the full word **caption**.

k. **Highlight entire source line for breakpoints** check box, which causes the background of the entire line with a breakpoint to be drawn using your preferred breakpoint highlight color. By default, this option is enabled; that is, it is checked.

Uncheck this check box to remove the highlighted color of all source lines that have a breakpoint set so that the background of the breakpointed source line is drawn normally by the editor.

6. In the Edge Mode Options group box:

a. In the **Edge Column** text box, specify the number of the column at which the long line marker is displayed. Although no long-line marker is displayed by default (that is, the default value is zero), you can specify a value in the range **1** through **200**.

b. Select or specify the edge mode that is used to display long lines in the editor pane, by selecting one of the following values.

- **None** (the default value), when no long-line marker is displayed.
- **Background**, in which the background color (edge marker) is used to mark long lines.
- **Line**, in which a vertical line is drawn at the column number specified in step a.

Note Although this mode works well for monospaced fonts, as the line is drawn at a position based on the width of a space character, it may not work very well if you use proportional fonts.

Maintaining Exit Options

In Jade, the exit process performs certain actions by default. You can use the exit options to change the default exit behavior.

To change your default exit options

1. In the Preferences or Jade Installation Preferences dialog, click the **Exit** sheet.
2. If you want all breakpoints in all methods in the application and any options specified for those breakpoints saved when you exit from the Jade Platform, check the **Save Breakpoints** check box.
3. If you want the current open Class Browser windows, free-standing method editor windows, and the Schema Browser window to be saved when you exit from Jade, check the **Save Windows** check box.

Note This option applies only to open Class Browser windows, free-standing method editor windows, and the Schema Browser window.

4. If you do not want the Exiting Jade confirmation dialog displayed when you exit from Jade, uncheck the **Exit Confirmation** check box.
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus. Your selected preferences are displayed by default when you exit from Jade and then restart a Jade work session.

Maintaining Lock Options

In a Jade multiuser environment, an object is locked by another object, guaranteeing that the lock operation is the first reference to the object that is the target of the lock. You can use the **Lock** sheet in the Preferences dialog to change the retrying of locked objects from your workstation.

To change the default lock options

1. In the Preferences or Jade Installation Preferences dialog, click the **Lock** sheet.
2. Change the number of retries on a locked object in the **Number of times to Retry** text box, if required. (By default, locks are retried 10 times before the lock request is rejected.)
3. If you want the user in a multiuser Jade environment to be prompted if the requested object is locked, select the **Prompt to Retry** option button. (By default, the retry of a locked object is automatic.) Locks are released only when the user no longer wants to retry the lock.
4. If you do not want a request for a lock on an object that is currently locked to be retried, select the **Do not Retry** option button. (By default, the retry of a locked object is automatic; that is, the **Retry Automatically** option is selected.)

The current window then has focus, and displays your selected preferences.

Maintaining Message Box Suppression Options

There are a number of messages boxes that appear frequently to which the response is usually always the same, for example:

- After a schema load: *Load Completed* or a .**mul** file load of multiple schemas: *Load Completed Successfully*
- When a change prompts recompilation: *Your change requires n method(s) to be recompiled. Recompile now?*
- When compiling a method: *Do you want to view the error list?*
- Confirmation when moving or copying a method
- When adding a method: *Do you want to reimplement superclass method?*
- Find and replace in the editor pane: *Search text not found* and *Replace successful*

You can specify that the display of a number of message boxes is suppressed and your answers are retained for your future work sessions. When a message box is suppressed, the button you set is recalled and its **click** event is automated.

By default, no message boxes are suppressed. Suppress message boxes by performing one of the following actions.

- When a message box that can be suppressed is displayed, check or uncheck the **Hide in the future** check box if there is one button only. If there is more than one button, select the button you want to be automated and then check the **Remember my selection and hide in the future** check box.
- On the **Message Box Suppression** sheet of the Preferences dialog, by enabling or disabling each message box listed in the displayed table.

To change your default message box suppression options

1. In the Preferences dialog, click the **Message Box Suppression** sheet.
2. In the **Suppressed** column of the row of a check box, check the check box if you want that message box suppressed and then in the **Stored Value** column, select the button in the combo box that is to be automatically clicked when that message box is displayed.

   When the **Stored Value** embedded combo box is set to the first empty entry in the combo box, the **Suppressed** check box is unchecked. Conversely, if the **Suppressed** check box is not checked and you select a valid stored value in the embedded combo box, the **Suppressed** check box is checked.

   Note Your suppression options are not committed until you click the **OK** button on the Preferences dialog.
3. Click the **Suppress All** button at the bottom of the sheet to suppress all message boxes listed in the table, or the **Clear All** button to clear all of your message box suppression settings.
4. In the **Pop-up Display Time (ms)** text box, specify the number of milliseconds that the Status Line Popup dialog is displayed in the lower-left corner of the Jade Platform development environment if you want to decrease or increase the default display time of 4.75 seconds (that is, 4750 milliseconds).

   The default display time can be within the range 1500 through 15000 milliseconds, which adequately updates the user to the system state while suppressing the message-box and requiring no user-interactions. If you set a value lower than 1500 milliseconds or higher than 15000 milliseconds, the minimum or maximum value is set after a message box indicates that the value is invalid.
5. Check the **Hide Pop-up** check box if you do not want the Status Line Popup dialog displayed. By default, this check box is unchecked.

   The Status Line Popup dialog is displayed indefinitely when the cursor is over it. The timer resumes after the cursor ceases to hover over the dialog. Press Esc or click the close button, marked with **X**, to close the Status Line Popup dialog immediately.
6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Defaults** button to reset all preferences to the Jade default values.

The current window then has focus. Your selected preferences are displayed by default when you exit from Jade and then restart a Jade work session.

Applies to Version: 2020.0.01 and higher

Maintaining Miscellaneous Options

Use the miscellaneous preferences to change your default file suffixes (for example, the schema file suffix of **.scm**), the base locale, and interface and versioning options.

To change your miscellaneous preferences

1. In the Preferences or Jade Installation Preferences dialog, click the **Miscellaneous** sheet.
2. To change a file suffix, enter the required file suffix in the appropriate text box in the File Suffixes group box. (Jade files have default predefined suffixes.)

   Your file suffix can be in the range one through three characters. (You can specify lowercase or uppercase characters.) Do not specify the period character (**.**) that separates the file prefix from the suffix. This is assumed, and is applied automatically by Jade.

3. In the Interface Options group box:

   a. If you want generated stub methods in classes that implement an interface to automatically compile and generate class methods with errors so that they can be dealt with, check the **When implementing an interface, generate class methods with errors so they must be dealt with** check box.

      A compile-time error (caused by a text block added to the beginning of the method body) is then generated in implementing classes. This can help you to find the methods that you are likely to want to flesh out to make the implementation useful. For details, see "Adding Interface Methods", in Chapter 14.

   b. Check the **Generate stub methods without prefix** check box if you want new interface method names to be generated with the same name as the original interface method name, with no added prefix.

      This check box is unchecked by default, indicating that interface methods are prefixed with a default value of **stub_** until you define another prefix in the **Generated stub method prefix** text box or you check this check box to set this value to null.

   c. If you want to specify the method name prefix that is generated for all new implemented interface methods, specify the prefix that you require in the **Generated stub method prefix** text box.

Note These options apply to new implemented interface methods only; that is, they do not affect existing implemented interface method names. For details about implementing interfaces, see "Implementing an Interface", in Chapter 14.

4. In the Versioning Options group box:

   a. Check the **Warn when schemas are versioned** check box if you want a warning dialog displayed to all users signed on to the Jade Platform development environment who have a window open against the newly-versioned schema so that they can continue working in the current schema version or change to the latest (uncommitted) version. By default, all users of the Jade Platform development environment are not warned when schemas are versioned.

   b. Check the **Warn before making changes to any versioned source…** check box if you want a warning dialog displayed when an attempt is made to change source code when a schema is versioned. By default, a warning is not displayed when source code is changed.

   c. If you want to remove or change the Jade skin to decorate opened source windows in the latest (that is, uncommitted) version of schemas to provide a distinction between current version and latest version windows, select **<None>** or the skin that you require from the list in the **Select an alternative Jade skin…** list box.

      By default, the **Cashmere** skin is used in the latest version to distinguish between current and latest version windows; that is, **JADE2016Cashmere** is displayed in this list box. (For details, see "Using Jade Skins in Your Runtime Applications", in Chapter 1 of the *Runtime Application Guide*.)

      Tip We recommend that you use a different skin for the latest version, to provide strong visual feedback of the versioned state of your schema.

      If required, remove or change the skin for the current version from the **Window** sheet of the User Preferences dialog and for the latest version from the **Miscellaneous** sheet of the User Preferences dialog.

   d. If you want to show the composite view in new windows opened in versioned schemas rather than the selected version only, select the **Show composite version view by default** value from the list in the **Default viewing preference for new windows** list box.

Composite views show all versions of an object (for example, a class, property, or method in one window), with icons indicating the versions. By default, the selected version view is displayed; that is, windows for the current version or for the latest version.

Note You can toggle the display of the composite view and a selected version only view, by selecting the **Show Composite View** command from the View menu. When a composite version view is displayed, a check mark is displayed to the left of the command in the View menu.

For details about versioning, see "Object Versioning", in Chapter 13 of the *Developer's Reference*.

5. For a specific schema, the Jade Platform development environment can maintain forms and translatable strings in that schema for multiple locales.

   If a schema supports several locales (by using the **Locales** command from the Schema menu), the **Base Locale** combo box in the Select the Base Locale group box enables you to select the locale that you want to use as your default form and string translations in the development environment for all schemas.

   When you have set your base locale, any forms that you access in the Jade Painter are loaded from this locale. Additionally, any translatable strings that you access in the String Browser are obtained from your base locale.

6. If you want to save (export) all of your values on all Preference dialog sheets to a text file, click the **Export Preferences** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

   The common Save As dialog is then displayed, to enable you to change the name of the file and to specify the location of your extracted user preferences. The file name defaults to **JadeUserPreferences.ini** and the location defaults to your Jade working directory (for example, the Jade **bin** directory or to the last directory to which you saved a file in Jade).

   The Export User Preferences message box then advises you that your user preferences were successfully written to the specified directory and location.

   For details about loading your exported user preferences using the non-GUI **jadclient** or **jade** executable command line, see "Loading User Preferences from the Command Line", in Chapter 1 of the *Runtime Application Guide*.

   The loading of an exported file of your user preferences does not support the older-style editor key bindings entries in your user preferences that were created in releases earlier than Jade version 2018.0.01. The loading of such files is therefore rejected when you import an earlier user preferences file and by a batch file load using the non-GUI **jadclient** or **jade** executable command line.

   You should therefore export your older user preferences to a file in Jade 2018.0.01 and higher before importing or loading them into Jade 2020.0.01 and higher.

7. If you want to load (import) user preference values from an existing user preferences extract file, click the **Import Preferences** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

   The common Open dialog is then displayed, to enable you to select the user preferences text file that you want to load. (For details about extracting user preferences to a file, see the previous step of this instruction.)

   The Import User Preferences message box then advises you that the selected user preferences file has been successfully read and that the Preferences dialog is updated with all values from this file. When you click the **OK** button on the Preferences dialog, the Jade Platform development environment is updated.

8. If you want to save (export) your user-defined theme that is currently displayed in your work session to an Extensible Markup Language (XML) format file so that you can import it into your next or another work session, click the **Export Jade Theme** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

   The common Save As dialog is then displayed, to enable you to change the name of the file and to specify the location of your extracted user-defined themes. The file name defaults to **JadeTheme.xml** and the location defaults to your Jade working directory (for example, the Jade **bin** directory or to the last directory to which you saved a file in Jade).

   A message box then advises you that your current user-defined theme was successfully written to the specified directory and location. The exported theme consists of the current **Window**, **Editor**, and **Relationship** sheet colors, the currently selected theme name, and the current color mode.

9. If you want to load (import) a user-defined theme from an existing user-defined theme extract file, click the **Import Jade Theme** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

   The common Open dialog is then displayed, to enable you to select the user-defined theme XML file that you want to load. (For details about extracting your user-defined themes to a file, see the previous step of this instruction.)

   A message box then advises you that the selected theme file has been successfully read and that the Preferences dialog is updated with all values from this file. When you click the **OK** button on the Preferences dialog, the Jade Platform development environment is updated.

10. If you want the Jade default miscellaneous values applied to all of your files, click the **Defaults** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)

11. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus. Your specified values then become your miscellaneous defaults for your schemas.

For details about importing and exporting hierarchy browser form layouts, see "Customizing the Layout of Hierarchy Browser Forms", earlier in this chapter.

Maintaining Relationship Options

When viewing relationships for a class, by default, the various components of your window are displayed in different colors.

Use the **Relationship** sheet to change the default relationship display. For example, all explicit many-to-many relationship lines are displayed in yellow and the foreground color is displayed in dark gray by default.

To change your default relationship options

1. In the Preferences or Jade Installation Preferences dialog, click the **Relationship** sheet.

2. Double-click the relationship component whose default color you want to change. For example, double-click the white panel to the right of **Background Color** in the Window list to change the default background color for your Jade windows. The common Color dialog is then displayed.

   The components whose colors can be changed are listed in the following table.

Component	Description
Background	Background color of the Relationship View window

Component	Description
Foreground	Font color
Source Class	Background color of the rectangle representing the selected class
Source Class Border	Border color of the rectangle representing the selected class
Target Class	Background color of the rectangle representing all other classes
Target Class Border	Border color of the rectangle representing all other classes
DragDrop Border	Border color of the rectangle that is displayed when moving a rectangle
Inverse One to One	Line color of the line representing inverse (explicit) one-to-one relationships
Inverse One to Many	Line color of the line representing inverse (explicit) one-to-many
Inverse Many to Many	Line color of the line representing inverse (explicit) many-to-many relationships
No Inverse One to One	Line color of the line representing no inverse (implicit) one-to-one relationships
No Inverse One to Many	Line color of the line representing no inverse (implicit) one-to-many relationships
Relationship Default	Line color of relationship lines when relationship details are not displayed

3. Click on the required color. The selected color is then outlined in black. (This common dialog also enables you to define your own custom color, if required.)
4. Click the **OK** button. The Preferences dialog is then redisplayed, with your selected color displayed beside the component.

   Repeat steps 2 through 4 of this instruction, to make all your required color display changes.
5. If you want to customize the relationship view by hiding the display of one or more relationship types (for example, an inverse one-to-one relationship), uncheck the appropriate check box in the Relationships group box.
6. If you want to display both transient and persistent classes in the relationship, check the **Show Transient Classes** check box.
7. If you do not want properties involved in a relationship to be displayed, uncheck the **Show Detail** check box.
8. If you do not want the relationship legend displayed at the bottom of the Relationship View window, uncheck the **Show Legend** check box.
9. If you do not want your relationship view output to a monochrome (black and white) printer, check the **Enable Color Printing** check box to specify that your relationship views can be output to a color printer.

   When relationship views are output to a monochrome printer, all lines, borders, and text are printed as black, as some of the lighter colors that you may have defined for your relationship views would not be visible when printed in black and white.
10. Click the **Font** button if you want to change the default window font from Arial Regular 8.25 points. The common Font dialog is then displayed, to enable you to make your font selections.

Note In order for the **Zoom In** and **Zoom Out** facilities to work in the Relationship View window, only True Type fonts are available for selection.

When you have made your font selections, focus is then returned to the Preferences or Jade Installation Preferences dialog.

11. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus, and displays your selected preferences.

Maintaining Schema Options

By default, access to classes, references, and attributes in schemas is protected, and classes are persistent with a **Real** type. You can use the **Schema** sheet of the Preferences or Jade Installation Preferences dialog to change these default options for your schemas.

To change your default schema options

1. In the Preferences dialog, click the **Schema** sheet. As the **Protected** option in the Access group box of the Class Options frame is reserved for future use, only public access to classes is supported; that is, selecting the **Protected** option button has no effect. (Instances of classes that have public access can be created or deleted by another class.)
2. In the Type group box of the Class Options frame, select the **Abstract** option button if you do not want the classes in your schemas to be of **Real** type by default.
3. In the Persistence group box of the Class Options frame, select the **Transient** option button if you do not want the classes in your schemas to be persistent by default.
4. Check the **Dictionary string keys are case insensitive** check box in the Class Options frame if you do not want dictionary string keys to be case-sensitive.
5. In the Access group box of the Reference Options frame, select the appropriate type of access if you do not want access to the properties in your schemas to be protected by default.
6. In the Access group box of the Attribute Options frame, select the appropriate type of access if you do not want access to the attributes in your schemas to be protected by default.
7. In the Methods Access group box, select the **Protected** type of access if you do not want access to the methods in your schemas to be public by default. For details, see "protected Option", in Chapter 1 of the *Developer's Reference*.
8. Check the **Generate Get/Set Methods for Protected and Set Method for Read Only Properties** check box if you want get and set methods generated for protected properties and the set method generated for read-only properties in your schemas.

   Protected properties can then be got or set and read-only properties set from methods, but still cannot be updated directly.
9. If you want the Jade default schema options applied to all your schemas, click the **Defaults** button. (This option is not available on the administrative Jade Installation Preferences dialog, as it is user-specific.)
10. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus. Your specified options then become your default schema preferences.

Maintaining Shortcut Keys

You can change most of the development environment and editor shortcut keys to values of your choice; for example, changing the editor Ctrl+S key binding (**Insert Syntax**) to the **Save** function. However, you cannot change editor pane accelerator key bindings, standard Windows edit functionality shortcuts (for example, for copy, paste, and undo actions), and Windows form shortcuts. In addition, you cannot:

- Add a shortcut to a menu item that does not have a shortcut defined in the Jade Painter
- Change the key accelerator in the caption of a menu item; for example, **&File** to **F&ile**

For details about the default function keys, see "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter. See also "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*.

To change the default shortcut keys

1. Click the **Short Cut Keys** sheet of the Preferences or Jade Installation Preferences dialog. The **Short Cut Keys** sheet of the Preferences dialog is then displayed, with all shortcuts in all category grouping displayed by default; that is, the **All** category is displayed in the **Category** combo box in the Menu Short Cut Keys group box.

   The category groupings are listed in the following table.

Category	Displays...
Action	The shortcuts for the action key category; for example, **Open Painter**.
All	All shortcut keys defined in all categories. The displayed table includes the category name in which each shortcut is defined.
Browse	The shortcut keys for the Browse *entity*.
Debugger	The shortcut for the Jade debugger.
Edit	The edit shortcuts, which are disabled and cannot be changed because they are standard cut, paste, undo, and so on, actions defined by the Windows environment.
Editor	The Jade editor shortcuts for the Jade editor pane. Any standard cut, paste, and so on entries are disabled for the standard Windows edit actions. The Jade editor shortcut keys (for example, Shift+Ctrl+A) are the only other shortcuts that cannot be changed (pasted text can be changed but not the shortcut). The **Editor** category table also includes entries that are available but for which no shortcut key is currently assigned by default.
Find	The shortcuts for the find actions.
Inspect	The shortcuts for the Inspect object.
Monitor	The shortcuts for the Jade Monitor.
Painter	The shortcuts for the Jade Painter.
UnitTest	The shortcuts for the Unit Test Runner form.
Windows	The shortcuts for the Windows category. These shortcuts cannot be changed because they are standard in Microsoft Windows for actions such as **Close Form**.

2. To display only those shortcut keys in a specific category, select that category in the **Category** combo Only those shortcuts in that category are then displayed in the table, together with a description of the functionality to which the shortcut applies.

The table is initially sorted by description name. Entries that have been changed by you are displayed with background color associated with versioned entries.

3. To change the sorting of entries in the table, click on the fixed cell (heading) of the column whose sort order you want to change. Clicking on the **ShortCut** column heading sorts the table by the alphabetic shortcut key, ignoring the Shift, Ctrl, and Alt qualifiers; for example, all of the defined **A** shortcuts are listed together.

4. Click on an enabled table row to display the currently defined shortcut key and its description in the Change Menu Short Cut Key group box.

5. To change the shortcut key binding, move the focus to the **Key Combination** text box and then press the keys of the shortcut you require.

   When you override a default shortcut key binding, you can no longer use the key combination for its original purpose. For example, pressing Ctrl+S in the Jade editor pane inserts Jade syntax at the caret position. If you assign Ctrl+S to a new editor action, you cannot insert syntax by pressing Ctrl+S unless you restore the default shortcut key binding by removing the key binding that you assigned to another action.

   After you change one or more shortcut keys, the **Change** button is enabled. When you click this button, your assigned shortcut is displayed in the table.

   Notes Conflict detection is performed only when the Preferences dialog is closed after you click the **OK** button. This allows conflicting shortcuts to be changed more easily.

   Some entries (for example, **Toggle Breakpoint**) are displayed in more than one category, and changing one value changes the assigned shortcut in each category. Shortcuts in Painter, Debugger, and Monitor are independent of other categories, so can use a key binding defined in other categories.

6. If the selected shortcut is no longer set to the Jade defined default, the **Restore Default** button is enabled. Click this button to restore the selected entry back to the defined Jade default value.

7. If you have changed any shortcuts, the **Restore All Defaults** button is enabled. Click this button to discard all of your changes and restore the Jade default values.

8. To restore all of your defined shortcut key values in all categories to the Jade default values, click the **Restore All Defaults** button.

9. To toggle the default F5 and F9 shortcut key functions, check the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) short cut keys** check box. This check box is unchecked by default; that is, the F5 shortcut key causes break points to be toggled and the F9 shortcut key executes or debugs a script method and continues execution of the Jade debugger.

   Visual Studio uses the F9 key to set breakpoints, by default, and uses the F5 key to run and continue debug execution. Jade uses the F5 key to set breakpoints and the F9 key to run and continue debug execution. Switching between the two development environments can lead to confusion and frustration when using the F5 and F9 keys.

   If you check this check box, the menu caption and shortcut key for the Jade debugger Breakpoints menu **Toggle** command is changed to be triggered by the F9 key. In addition, the Jade Platform development environment Jade menu **Execute**, **Debug**, **Unit Test**, and **Unit Test Debug** commands and the Jade debugger Debug menu **Continue** and **Animate** commands will be changed to use the F5 key.

   Notes The setting of the check box is saved when you extract your user preferences to a file and they are restored when your reload your preferences.

   Changing the setting of the check box takes effect in the Jade Platform development environment as soon as you close the Preferences dialog. It will not take effect within any currently active Jade debug session, and it is read only when you initiate a new debug session.

10. To toggle the default F11 and F12 shortcut key functions, check the **Swap F11 (Show Symbol/Open class browser) and F12 (New window/Bookmark) short cut keys** check box. This check box is unchecked by default. You could swap the F11 and F12 shortcut key functionality, for example, if you want the F12 key in the Jade editor pane to have a similar meaning to F12 in Visual Studio, where the definition of the selected symbol is displayed.

    Note Swapping the F11 and F12 shortcut keys in the editor pane does not affect the interpretation of the F11 function key in the Jade Debugger.

    When this option is **false** (the default) while in the editor pane of a method, when pressing:

    - F11, information for the currently selected symbol is displayed, including the creation timestamp and patch version of the entity.

      If the symbol is a method, a new method window is opened for the method or if it is the name of the method in the method signature, the Implementors of *class-name*::*method-name* form is displayed.
    - F11+Shift, a new Class Browser is opened if the symbol is a class.
    - F12, a new editor pane is opened for the currently displayed method.
    - F12+Shift, the current state is saved as a bookmark or the currently defined bookmark state is restored.

    If you check this check box, the F11 and F12 shortcut key functionality is swapped.

    Notes The setting of the **Swap F11 (Show Symbol/Open class browser) and F12 (New window/Bookmark) shortcut keys** check box is saved when you extract your user preferences to a file and it is restored when you reload your preferences.

    Changing the setting of the check box takes effect in the Jade Platform development environment as soon as you close the Preferences dialog. It will not take effect within any currently active Jade session.

11. After you have made your shortcut key binding changes, click the **OK** button to validate and save your changes.

    The save action is rejected if there are conflicts where the same shortcut is assigned to different actions in the same area. A message explaining the conflict is displayed, the Preferences dialog close action is rejected, and no changes are saved.

When the changes are valid, the changes are:

- Stored in your user profile.

  Note The shortcut key bindings are saved in your user profile. You can extract them to a preferences initialization file by using the existing **Export Preferences** functionality provided on the **Miscellaneous** sheet of the Preferences dialog. Exported (saved) preferences are restored if you import that export file. The preferences initialization file contains sections for each category that has been changed.

  A preferences initialization file with previously changed editor key bindings from a release earlier than Jade 2018 can still be read and handled. The key binding entries from your old file are added to the entries displayed in the table for the Editor category.
- All open forms in the Jade Platform development environment are updated with the new menu shortcuts.
- All open editor windows are updated with the new editor shortcuts.
- Are not applied to any open Monitor, Painter, or Debugger session. To pick up the new key bindings, you must restart these sessions.

Applies to Version: 2018.0.01 and higher

For information about the default shortcut keys by category, see the following subsections. (The default **All** category is a combination of the actions in all other categories.)

Action Shortcut Key Category

The following table lists the Jade default action shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Shift+F12	Bookmark	Creates a bookmark for the current position in a Class, Primitive Types, or Interface Browser window
Shift+F9	Debug Execute	Saves, compiles, executes, and debugs the current **JadeScript** method or unit test
F9	Execute It	Saves, compiles, and executes the selected **JadeScript** method or Workspace
Ctrl+Alt+G	Go To Line	Accesses the Find Position in Method Source dialog
Ctrl+P	Open Painter	Accesses the Jade Painter
Ctrl+Q	Quick Navigation	Accesses a specified class, an interface or a method
Delete	Remove Breakpoint	Removes the selected breakpoint
Delete	Remove Global Constant	Removes the selected global constant
Delete	Remove HTML Document	Removes the selected HTML document
Ctrl+R	Run Application	Runs an application
F2	Save	Saves the text in the active window
F5	Toggle Breakpoint	Runs and continues debug execution

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Applies to Version: 2018.0.01 and higher

Browse Shortcut Key Category

The following table lists the Jade default browse shortcut key bindings (that is, for commands in the Jade Platform development environment Browse menu).

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+L	Browse Applications	Opens the Application Browser window

Keystroke	Action	Description
Ctrl+B	Browse Classes	Opens the Class Browser window, with the class selected in the new browser defaulting to the same class that is currently selected when the action is performed from a hierarchy browser
Ctrl+D	Browse Deltas	Opens the Delta Browser window
Shift+Ctrl+I	Browse Display Version Info	Opens the Version Info window
Shift+Ctrl+R	Browse External Component Libraries	Opens the External Components Browser window
Shift+Ctrl+F	Browse External Functions	Opens the External Functions Browser window
Ctrl+G	Browse Global Constants	Opens the Global Constants Browser window
Ctrl+U	Browse HTML Documents	Opens the HTML Document Browser window
Ctrl+N	Browse Interfaces	Opens the Interface Browser window
Ctrl+E	Browse Libraries	Opens the Library Browser window
Ctrl+M	Browse Maps	Opens the Class Maps Browser window
Ctrl+K	Browse Methods and Properties	Opens the Methods and Variables Browser window when you had set the user preference to show classes only
Ctrl+T	Browse Primitive Types	Opens the Primitive Types Browser window when you had set the user preference to show classes only
Ctrl+W	Browse Properties	Opens the References Browser when you had set the user preferences to show classes only
Ctrl+O	Browse Relational Views	Opens the Relational Population Service Browser window
Shift+Ctrl+P	Browse RPS Mappings	Opens the Relational Population Service Browser window
Ctrl+H	Browse Schema Views	Opens the Schema Views Browser window
Ctrl+E	Browse Schemas	Accesses the Schema Browser from the translation forms
Shift+Ctrl+C	Browse Status List	Opens the Method Status List dialog
Ctrl+T	Browse Types	Opens the Class, Interface, or Primitive Types Browser window from the translation forms

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Debugger Shortcut Key Category

The following table lists the Jade default debugger shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green. (The **(alt)** value indicates an alternative shortcut that performs the same action.)

Keystroke	Action	Description
F9	Continue Debug	Continues code execution without interaction with the debugger
Ctrl+I	Inspect	Displays the Inspect Variable dialog
Ctrl+F8	Run To Caret	Executes code up to the line containing the caret
Shift+F11	Run To Caret (alt)	Executes code up to the line containing the caret
Shift+F8	Run To Return	Executes code until the current function returns to its call
Shift+F10	Run To Return (alt)	Executes code until the current function returns to its call
F7	Step Into	If the next instruction in the current line of code is a method call, displays the method and enables you to step through that method
F11	Step Into (alt)	If the next instruction in the current line of code is a method call, displays the method and enables you to step through that method
F8	Step Over	Executes the current line of code
F10	Step Over (alt)	Executes the current line of code
F5	Toggle Breakpoint	Sets or unsets a breakpoint on the line of code where the caret is positioned

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Edit Shortcut Key Category

The following table lists the Jade default browse shortcut key bindings.

Note The edit shortcuts are disabled and cannot be changed because they are standard actions such as cut, paste, and undo defined by the Windows environment.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+C	Edit Copy	Copies the selected text to the clipboard
Ctrl+X	Edit Cut	Cuts the selected text to the clipboard
Ctrl+V	Edit Paste	Pastes text copied or cut to the clipboard
Ctrl+Y	Edit Redo	Redoes the last edit actions
Ctrl+A	Edit Select All	Selects the complete contents of the current window or form
Ctrl+Z	Edit Undo	Undoes the last action
Alt+Backspace	Undo with Backspace	Uses the Backspace key action to undo the last action

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Editor Shortcut Key Category

The following table lists the Jade default shortcut key bindings, with all current key bindings that can apply to editor source windows.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green. Editor pane accelerator key bindings and standard Windows edit functionality shortcuts (for example, for copy, paste, and undo actions) are disabled for the standard Windows edit actions.

Keystroke	Action	Description
F6	AddTranslatableString	Opens the Add Translatable String dialog
Ctrl+Space	AutoComplete	Manually invokes the **AutoComplete** list box based on the current cursor location (for details, see "Using Jade AutoComplete Functionality", earlier in this chapter)
Ctrl+R	AutoCompleteToggleRootSchema	Toggles the display of RootSchema entities
Shift+Backspace	Backspace	Deletes the character to the left of the caret
Backspace	Backspace	Deletes the character to the left of the caret
Shift+Ctrl+U	BackTab	Removes indentation from the selected line or lines, or moves the caret to the left one tab character
Shift+Tab	BackTab	Removes indentation from the selected line or lines, or moves the caret to the left one tab character
Shift+F12	BookMark	Creates and toggles bookmarks for current positions
Escape	Cancel	Returns focus to the editor pane
LeftArrow	CharacterLeft	Moves the caret one character to the left
Shift+LeftArrow	CharacterLeftExtend	Extends the current selection one character to the left
Alt+Shift+LeftArrow	CharacterLeftRectExtend	Extends the rectangular selection one column to the left
RightArrow	CharacterRight	Moves the caret one character to the right
Shift+RightArrow	CharacterRightExtend	Extends the selection one character to the right

Keystroke	Action	Description
Alt+Shift+RightArrow	CharacterRightRectExtend	Extends the rectangular selection one column to the right
F8	CompileMethod	Compiles the current method
Shift+F8	CompileMethodSavePatchHistory	Compiles the current method and saves the patch history
Ctrl+Insert	Copy	Copies the selected text to the clipboard
Ctrl+C	Copy	Copies the selected text to the clipboard
	CopyLine	Copies the current line to the clipboard
Alt+Ctrl+C	CopyToClipBuffer	Copies the selected text to the clipboard buffer defined by the next numeric keystroke
Ctrl+X	Cut	Copies the selected text to the clipboard and deletes it
Shift+Delete	Cut	Copies the selected text to the clipboard and deletes it
	CutLine	Copies the current line to the clipboard and deletes it
Delete	Delete	Copies the current line to the clipboard
Ctrl+K	DeleteLine	Deletes the current line
Shift+Ctrl+Backspace	DeleteLineLeft	Deletes the line to the left of the caret
Shift+Ctrl+Delete	DeleteLineRight	Deletes the line to the right of the caret
Ctrl+Backspace	DeleteWordLeft	Deletes the word to the left of the caret, and deletes the word to the left of selected text if text is currently selected
Ctrl+Delete	DeleteWordRight	Deletes the word to the right of the caret, and deletes the word to the right of selected text if text is currently selected
Alt+End	DisplayLineEnd	Moves the caret to the end of the display line (that is, to the last character on the current line)
	DisplayLineEndExtend	Extends the selection to the end of text within a line
Alt+Home	DisplayLineStart	Moves the caret to the start of the display line (that is, to the first character on the current line)
	DisplayLineStartExtend	Extends the selection to the start of text within a line
Ctrl+End	DocumentEnd	Moves the caret to the end of the text
Shift+Ctrl+End	DocumentEndExtend	Extends the selection to the end of the text
Ctrl+Home	DocumentStart	Moves the caret to the start of the text
Shift+Ctrl+Home	DocumentStartExtend	Extends the selection to the start of the text
	DuplicateLine	Duplicates the current line

Keystroke	Action	Description
Alt+Ctrl+RightArrow	HistoryNext	Displays the method, property, or type that was next accessed
Alt+Ctrl+LeftArrow	HistoryPrior	Displays the method, property, or type that was previously accessed
Ctrl+N	InsertLineAbove	Inserts a line above the current line
Ctrl+S	InsertSyntax	Inserts the remainder of the syntax for a specified instruction
Shift+Ctrl+A	InsertText (A): abortTransaction	Inserts the **abortTransaction** instruction
Shift+Ctrl+B	InsertText (B): beginTransaction	Inserts the **beginTransaction** instruction
Shift+Ctrl+C	InsertText (C): commitTransaction	Inserts the **commitTransaction** instruction
Shift+Ctrl+D	InsertText (D): <null>	
Shift+Ctrl+E	InsertText (E): endforeach	Inserts the **endforeach** instruction
Shift+Ctrl+F	InsertText (F): <null>	
Shift+Ctrl+G	InsertText (G): <null>	
Shift+Ctrl+J	InsertText (J): <null>	
Shift+Ctrl+K	InsertText (K): <null>	
Shift+Ctrl+L	InsertText (L): <null>	
Shift+Ctrl+N	InsertText (N): <null>	
Shift+Ctrl+O	InsertText (O): <null>	
Shift+Ctrl+P	InsertText (P): <null>	
Shift+Ctrl+Q	InsertText (Q): <null>	
Shift+Ctrl+R	InsertText (R): <null>	
Shift+Ctrl+S	InsertText (S): <null>	
Shift+Ctrl+T	InsertText (T): <null>	
Shift+Ctrl+V	InsertText (V): <null>	
Shift+Ctrl+X	InsertText (X): <null>	
Shift+Ctrl+Y	InsertText (Y): <null>	
Shift+Ctrl+Z	InsertText (Z): <null>	
DownArrow	LineDown	Moves the caret down one line
Shift+DownArrow	LineDownExtend	Extends the selection down one line
Alt+Shift+DownArrow	LineDownRectExtend	Extends the rectangular selection down one line
End	LineEnd	Moves the caret to the end of the current line
Shift+End	LineEndExtend	Extends the selection to the end of the current line
Alt+Shift+End	LineEndRectExtend	Extends the rectangular selection to the end of the line

Keystroke	Action	Description
	LineEndWrap	Moves the caret to the end of the visible line when wrapping is enabled
	LineEndWrapExtend	Extends the selection to the end of the visible line when wrapping is enabled
Home	LineStart	Moves the caret to the start of the current line
Shift+Home	LineStartExtend	Extends the selection to the start of the current line
Alt+Shift+Home	LineStartRectExtend	Extends the rectangular selection to the start of the line
	LineStartWrap	Moves the caret to the start of the visible line when wrapping is enabled
	LineStartWrapExtend	Extends the selection to the start of the visible line when wrapping is enabled
UpArrow	LineUp	Moves the caret up one line
Shift+UpArrow	LineUpExtend	Extends the selection up one line
Alt+Shift+UpArrow	LineUpRectExtend	Extends the rectangular selection up one line
Ctrl+3	ListConstants	Lists the constants for the current type
Shift+Ctrl+3	ListConstantsLocal	Lists the constants that are local to the type being accessed
Ctrl+4	ListLocals	Lists the local variables for the current type
Ctrl+2	ListMethods	Lists the methods for the current type
Shift+Ctrl+2	ListMethodsLocal	Lists the methods that are local to the type being accessed
Ctrl+1	ListProperties	Lists the properties for the current type
Shift+Ctrl+1	ListPropertiesLocal	Lists the properties that are local to the type being accessed
Shift+F6	ListTranslatableStrings	Lists the translatable strings
	Lowercase	Converts the selected text to lowercase
	MacroPlaySaved	Saves the temporary macro as a persistent macro
	MacroPlayTemp	Plays the temporary macro
	MacroRecordStart	Starts recording keystrokes as the current temporary macro
	MacroRecordStop	Stops recording keystrokes as the current temporary macro
Return	NewLine	Inserts a new line below the current line
Shift+Return	NewLine	Inserts a new line above the current line
Ctrl+Return	NewLineNoIndent	Inserts a new line with no indentation
	NextLinemark00	Goes to the next line with linemark zero (**0**)

Keystroke	Action	Description
	NextLinemark00to04	Goes to the next line with a linemark in the range zero (**0**) through **4**
	NextLinemark01	Goes to the next line with linemark **1**
	NextLinemark02	Goes to the next line with linemark **2**
	NextLinemark03	Goes to the next line with linemark **3**
	NextLinemark04	Goes to the next line with linemark **4**
	NoAction	Does nothing (that is, performs no action)
Shift+F11	OpenClassBrowser	Displays the Class Browser
	OpenFindReplace	Displays the Find / Replace dialog
F1	OpenHelp	Displays online help for the item with input focus
	OpenMacroLibrary	Selects the persistent macro to play or edit
F12	OpenSourceWindow	Opens a new editor pane with the selected method, or when details of a property that has a mapping method are displayed after pressing F11, displays a freestanding editor pane containing the mapping method source for that property
PageDown	PageDown	Moves the caret down one screen
Shift+PageDown	PageDownExtend	Extends the selection down one screen
Alt+Shift+PageDown	PageDownRectExtend	Extends the rectangular selection down one screen
PageUp	PageUp	Moves the caret up one screen
Shift+PageUp	PageUpExtend	Extends the selection up one screen
Alt+Shift+PageUp	PageUpRectExtend	Extends the selection block up one screen
Ctrl+]	ParagraphDown	Moves the caret to the next paragraph (which is delimited by an empty line)
Shift+Ctrl+]	ParagraphDownExtend	Extends the selection down one paragraph
Ctrl+[	ParagraphUp	Moves the caret to the previous paragraph (which is delimited by an empty line)
Shift+Ctrl+[	ParagraphUpExtend	Extends the selection up one paragraph
Shift+Insert	Paste	Pastes the contents of the clipboard, replacing the selection (if any) or inserting it after the caret position
Ctrl+V	Paste	Pastes the contents of the clipboard, replacing the selection (if any) or inserting it after the caret position

Keystroke	Action	Description
Alt+Ctrl+V	PasteFromClipBuffer	Pastes the contents of the specified clip buffer defined by the next numeric keystroke, replacing the selection (if any) or inserting it after the caret position
	PriorControl	Displays the previous control
	PriorLinemark00	Goes to the previous line with linemark zero (**0**)
	PriorLinemark00to04	Goes to the previous line with a linemark in the range zero (**0**) through **4**
	PriorLinemark01	Goes to the previous line with linemark **1**
	PriorLinemark02	Goes to the previous line with linemark **2**
	PriorLinemark03	Goes to the previous line with linemark **3**
	PriorLinemark04	Goes to the previous line with linemark **4**
Ctrl+Y	Redo	Redoes the last action
Ctrl+F2	RefactorMenu	Displays the Refactor menu at the cursor position
Shift+F2	RefactorRename	Changes or renames an identifier selected in the body of a method
Ctrl+DownArrow	RollDown	Moves the text displayed in the editor down by one line, but the caret remains on the same line
Ctrl+UpArrow	RollUp	Moves the text displayed in the editor up by one line, but the caret remains on the same line
Ctrl+A	SelectAll	Selects all text in the editor pane
	SelectWholeLines	Selects a number of whole lines
Ctrl+5	ShowMethodSignature	Displays the signature of the last method before the caret position; for example, when the caret is positioned after a comma between parameters
F11	ShowSymbolInfo	Displays details about the item under the caret, including the creation timestamp and patch version of the entity
Shift+Ctrl+W	SplitOpenClose	Toggles (opens or closes) the lower view of an editor pane
Ctrl+W	SplitOpenRefocus	Opens the lower view of an editor pane or toggles focus between the two views
Shift+Ctrl+I	Tab	Indents the selected line or lines, replaces selected text with a tab character for a partially selected line, or inserts a tab character if no text is selected

Keystroke	Action	Description
Tab	Tab	Indents the selected line or lines, replaces selected text with a tab character for a partially selected line, or inserts a tab character if no text is selected
F5	ToggleBreakpoint	Toggles a debugger breakpoint on the line where the caret is currently positioned
Ctrl+NumMultiply	ToggleFold	Toggles the fold point
Shift+Ctrl+NumMultiply	ToggleFoldAllBase	Toggles all outer (parent) fold points
	ToggleFoldTree	Toggles the line if it is a fold point and makes all lower-level (child) fold points match
Insert	ToggleInsertOvertype	Toggles the **Insert** or **Overtype** mode
	ToggleLinemark00	Toggles linemark zero (**0**) on the current line
	ToggleLinemark01	Toggles linemark **1** on the current line
	ToggleLinemark02	Toggles linemark **2** on the current line
	ToggleLinemark03	Toggles linemark **3** on the current line
	ToggleLinemark04	Toggles linemark **4** on the current line
Ctrl+0	ToggleSingleColor	Toggles the display of multiple-color or single color for the current editor pane
	TranslateHexToUtf8	Translates the selected hexadecimal text to a character (Unicode system only)
	TranslateUtf8ToHex	Translates the selected character to its Unicode hexadecimal value (Unicode system only)
	TransposeLine	Transposes the current line with the previous line
Alt+Bksp	Undo	Undoes the last action
Ctrl+Z	Undo	Undoes the last action
	Uppercase	Converts the selected text to uppercase
Ctrl+LeftArrow	WordLeft	Moves the caret to the previous word
Shift+Ctrl+LeftArrow	WordLeftExtend	Extends the selection to the previous word
Ctrl+/	WordPartLeft	Moves the caret position to the previous uppercase letter or punctuation character
Shift+Ctrl+/	WordPartLeftExtend	Extends the selection to the previous uppercase letter or punctuation character
Ctrl+\	WordPartRight	Moves the caret position to the next uppercase letter or punctuation character
Shift+Ctrl+\	WordPartRightExtend	Extends the selection to the next uppercase letter or punctuation character
Ctrl+RightArrow	WordRight	Moves the caret to the next word

Keystroke	Action	Description
Shift+Ctrl+RightArrow	WordRightExtend	Extends the selection to the next word
Ctrl+NumPlus	ZoomIn	Magnifies the text size
Ctrl+NumMinus	ZoomOut	Reduces the text size
Ctrl+NumDivide	ZoomZero	Restores text size to normal

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Find Shortcut Key Category

The following table lists the Jade default search shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
F3	Edit Find Again	Finds the next occurrence of specified text
Ctrl+F3	Edit Find At Cursor	Finds the next occurrence of the text on which the cursor is positioned
Ctrl+F	Edit Find Replace	Locates and optionally replaces an element
Shift+F3	Find Again Reverse	Performs another search using your last set of search options but in the opposite direction
F4	Find Entity In List	Locates an entity in a list
Shift+Ctrl+F3	Global Search	Locates and displays all occurrences of specified text
Ctrl+6	Search For Constant Name	Searches a hierarchy Class Browser for a specific constant
Ctrl+6	Search For Property Name	Searches a browser for a specific property
Ctrl+7	Search For Method Name	Searches a browser for a specific method

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Inspect Shortcut Key Category

The following table lists the Jade default inspector shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+I	Inspect Instances	Displays instances of the selected class
Ctrl+I	Inspect Item	Displays the Schema Collection Inspector for the current class (equivalent to selecting the **Inspect Instances** command from the Classes menu) to inspect the value of a primitive item, an item containing an object reference, or an item in the Call Stack Browser
Ctrl+J	Inspect Shared Transients	Display all shared transient instances of the current class

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Monitor Shortcut Key Category

The following table lists the Jade default monitor shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+F8	Activate Timer	Activates the sampling of activities at a specified period
F6	Clear Table Sort Settings	Clears the sorting of a table column
F3	Find	Opens the Version Info window
Ctrl+1	Font Bigger	Increases the size of the displayed font
Ctrl+0	Font Smaller	Decreases the increased size of the displayed font
F4	Overview	Toggles the display of the Overview pane
Ctrl+P	Print	Opens the HTML Document Browser window
F5	Refresh	Takes a new sample of the current activity and displays the sampled data
Ctrl+F5	Refresh All	Takes a new sample of all activities and updates the displayed the sampled data on all views
F2	Save	Saves the current sampling setup to the default directives file

Applies to Version: 2018.0.02 (Service Pack 1) and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter

- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Painter Shortcut Key Category

The following table lists the Jade default painter shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+Insert	Add Control	Adds a new control to your current form
Ctrl+B	Align Bottom	Aligns the bottom edges of the selected controls
Ctrl+L	Align Left	Aligns the left edges of the selected controls
Ctrl+R	Align Right	Aligns the right edges of the selected controls
Ctrl+T	Align Top	Aligns the top edges of the selected controls
Ctrl+Alt+T	Bring All Parents and Control to Top	Brings the selected control and all parents to the top of all overlapping controls
Shift+Ctrl+T	BringToTop	Brings the selected control to the top of the control or controls that overlap it
Shift+Ctrl+H	Centre Horizontally	Centers a control or group of controls relative to the horizontal axis of your form
Shift+Ctrl+V	Centre Vertically	Centers a control or group of controls relative to the vertical axis of your form
Ctrl+E	Edit Control Palette	Edits the buttons on the Control palette
Ctrl+C	Edit Copy	Copies the selected controls and their children to the clipboard
Ctrl+X	Edit Cut	Cuts the selected controls and their children to the clipboard
F12	Edit Form	Loads an existing form so that you can maintain it
Ctrl+F	Find Control	Locates a specified control on the form
Ctrl+G	Grid	Accesses the Grid Options dialog
Ctrl+M	Menu Design	Access the Menu Design dialog
F3	New Form	Opens a new form
Ctrl+V	Edit Paste	Pastes the control or controls from the clipboard to the form
Ctrl+Alt+B	Push All Parents and Control to Bottom	Pushes the selected control and all of its parents behind all controls they overlap
Shift+Ctrl+B	Push to Bottom	Pushes the selected control behind the control that it overlaps
Ctrl+H	Same Height	Makes all selected controls the same height
Shift+Ctrl+S	Same Height and Width	Makes all selected controls the same height and width
Ctrl+W	Same Width	Makes all selected controls the same width
F2	Save Form	Saves your current form

Keystroke	Action	Description
Ctrl+A	Select All	Selects all of the controls on the form
Shift+Ctrl+C	Select All Children	Selects all controls on the form with the same parent as the selected form or control
Shift+Ctrl+L	Select All Siblings	Selects all controls on the form with the selected control's parent as their ancestor
Shift+Ctrl+P	Select Parent	Selects the parent, or superform
F5	Show Control Hierarchy Dialog	Accesses the Control Hierarchy dialog
F4	Show Properties Dialog	Accesses the Properties dialog
Ctrl+S	Standard Size	Resizes all selected controls to their default sizes
Shift+Ctrl+O	Tab Ordering	Changes the tab sequence that the user experiences at run time
F6	Translate Properties	Accesses the Translatable Property Browser
Ctrl+Z	Undo Last Layout	Undoes the last layout command
Shift+Ctrl+W	Wizard	Accesses the Form Wizard

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

UnitTest Shortcut Key Category

The following table lists the Jade default User Test Runner form shortcut key bindings.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green.

Keystroke	Action	Description
Ctrl+1	Font Bigger	Increases the size of the displayed font
Ctrl+0	Font Smaller	Decreases the increased size of the displayed font
F5	Refresh	Refreshes the unit test results displayed in the **Results** pane

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Windows Shortcut Key Category

The following table lists the Jade default Window shortcut key bindings.

Note The Windows shortcuts are disabled and cannot be changed because they are standard in Microsoft Windows for actions.

Default key bindings are indicated by the Window background color, which defaults to white. User-created key bindings are indicated by the versioned background color, which defaults to light green. Editor pane accelerator key bindings and standard Windows edit functionality shortcuts (for example, for copy, paste, and undo actions) are disabled for the standard Windows edit actions.

Keystroke	Action	Description
Alt+F4	Close Form	Closes the form
Ctrl+F4	Close Mdi Form	Closes the MDI form
Alt+Hyphen	Mdi Window Menu	Displays the MDI Window menu
Ctrl+PageUp	Next Folder Tab	Moves focus to the next folder tab
Ctrl+F6	Next Mdi Form	Moves focus to the next MDI form
Ctrl+Tab	Next Mdi Form	Moves focus to the next MDI form
Ctrl+PageDown	Prior Folder Tab	Moves focus to the prior folder tab
Shift+Ctrl+F6	Prior Mdi Form	Moves focus to the prior MDI form
Shift+Ctrl+Tab	Prior Mdi Form	Moves focus to the prior MDI form
Alt+Space	Window Menu	Displays the Window menu

Applies to Version: 2018.0.01 and higher

For details about changing shortcut keys, see "Maintaining Shortcut Keys", earlier in this chapter. See also:

- "Using Function Keys and Shortcut Keys" under "Using the Jade Platform Development Environment", earlier in this chapter
- "Using the JadeTextEdit Control" under "JadeTextEdit Class" and the **JadeTextEdit** class **bindKeyToCommand** method, in Chapter 2 of the *Encyclopaedia of Classes*

Maintaining Source Management Options

You can use the **Source Management** sheet to configure your delta, source comparison, miscellaneous, and source control preferences and behavior.

In Jade, your method sources are compared using default behavior. Additionally, your workstation emits a beep sound when an error is detected in a method.

To change your default source management options

1. In the Preferences or Jade Installation Preferences dialog, click the **Source Management** sheet.
2. If you do not want to be prompted when checking out methods, select the **Automatic Checkout** or **Manual Checkout (No Prompt)** options button in the Checkout Options group box of the Delta Options frame.
3. If you do not want methods to be modified without a delta being set, check the **Delta Must Be Set** check box in the Delta Options frame.

4. If you do not want newly defined methods to be automatically checked out, uncheck the **Check Out New Methods** check box.

5. If you want the comparison of your methods sources to be displayed for changes only, select the **Changes Only** options button in the View Options group box of the Compare Sources frame.

6. If you selected the **Changes Only** option button in the View Options group box, you can specify the number of lines either side of the current method line whose changes are to be compared by selecting the appropriate context number from the **Context** list box in the View Options group box.

7. If you want white space compared when your methods sources are compared, uncheck the **Ignore White Space** check box in the Compare Sources frame.

8. If you want the case (that is, capitalization) ignored when your method sources are compared, check the **Ignore Case** check box in the Compare Sources frame.

9. If you do not want your workstation to emit a beep sound when an error is detected in your methods, uncheck the **Beep On Error** check box.

10. If you have patch control enabled and you do not want your method sources saved every time you compile a method, uncheck the **Save Source on Every Compile** check box.

    Tip If this check box is unchecked to disable the saving of source each time a method is compiled, you can press Shift+F8 to save the source during the compilation.

    By default, every time a method is compiled when patch control is enabled, the previous source (and hence a history entry) is saved. When you uncheck this check box, the source is saved only when the user leaves the method.

11. If you want to reuse the same form to display the source of a method, when possible, rather than creating a new form for each such request, check the **Reuse Same Method Source Window For All** check box. For details, see "Reusing the Form Displaying Method Source", in Chapter 3.

    Checking this check box unchecks the **Reuse Same Method Source Window For Each Origin** check box if that was checked.

12. If you want to use a new form to display the source of a method, check the **Reuse Same Method Source Window For Each Origin** check box; for example, pressing F11 on a method name in a method in the editor pane opens an extended source window that is populated only with actions in the original browser or in the displayed extended source window. For details, see "Reusing the Form Displaying Method Source", in Chapter 3. Checking this check box unchecks the **Reuse Same Method Source Window For All** check box if that was checked.

13. If you want to extract and load definition files (for example, form, database, and ActiveX definitions) in the legacy DDB format instead of the XML DDX format, uncheck the **Extract as DDX (xml format) instead of DDB** check box.

    This check box, which is checked by default, controls whether the **Forms/Mappings as XML (ddx file)** check box on the **Extract Options** sheet of the Extract dialog is initially set or unset.

14. If you do not want to set UTF-8 with a Byte Order Mark (BOM) as the default encoding format during all extract processes that you perform, uncheck the **Extract as UTF-8 Encoded** check box. If you do not check this check box, the default encoding format is native ANSI or Unicode.

    This check box, which is checked by default, controls whether the **UTF-8 Encoded** check box on the **Schema Options** sheet of the Extract dialog is initially set or unset.

15. If you do not want a Jade Control File (**.jcf**) created during all extract processes that you perform, check the **Unset Extract option 'Create JCF Command File'** check box.

This check box, which is unchecked by default, controls whether the **Create Command File** check box on the **Schema Options** sheet of the Extract dialog is initially set or unset by turning off the **.jcf** file creation. The **Create Command File** check box is checked by default; that is, a **.jcf** file is created.

16. In the Source Control group box, which relates to Git *client* functionality, specify:

    a. Your Git user name in the **Committer Name** text box. You must specify a value, as it is used to identify the author of a Git commit operation.

    b. Your email address in the **Committer Email** text box. You must specify a value, as it is used to identify the author of a Git commit operation.

    c. Your working (local) directory in the **Working Directory** text box. Alternatively, browse to and select the location on your workstation of your working directory. This must be a valid folder; otherwise an error is raised.

    Notes You must enter a valid working directory to use the Git client functionality.

    If the specified directory is already a Git repository, you cannot clone into it.

17. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus. Your selected preferences are the default options for your methods when you exit from Jade and then restart a Jade work session.

Applies to Version: 2022.0.01 and higher

Maintaining Status List Options

Jade includes a Methods Status List window, which by default displays a list of all methods and class constants that are in error or have not been compiled.

» To change your default status list options

1. In the Preferences or Jade Installation Preferences dialog, click the **Status List** sheet.
2. If you do not want uncompiled methods and class constants to be displayed in the status list, uncheck the **Uncompiled Methods** check box.
3. If you do not want methods and class constants that contain errors to be displayed in the status list, uncheck the **Methods in Error** check box.
4. If you want compiled methods and class constants to be displayed in the status list, check the **Compiled Methods** check box. By default, compiled methods and class constants are not displayed in the status list.
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current window then has focus.

Maintaining Text Templates

To make it easier for your development team or teams to adhere to your development standards, Jade enables you to define or modify text templates for all of your Jade:

- Classes
- Methods in your Jade development database classes or primitive types
- Properties

- Class constants
- Interfaces

To add or change your default text template options

1. In the Preferences or Jade Installation Preferences dialog, click the **Text Templates** sheet.

   The **Method** folder is then displayed, enabling you to define or modify the template that is included in the method source when a method is created.

   You can specify the variable elements listed in the following table for all Jade methods, classes, class constants, properties, methods, and interfaces that are defined after the template definition.

Element	Syntax		Description
Name	<*entity-type*>Name:	#<*entity-type*>Name	Name of the created entity
Date	Date created:	#date	Current system date in the format of your locale
Developer	Created by:	#userId	ID of the user who defined the entity
Patch number	Patch Number:	#patchNumber	Current patch number or blank, if none

   When you create an entity, these elements are expanded to include the entity name, the user code of the user who added the entity, the date that it was created, and the current patch number of the user or blank if there is no current patch number.

   Notes To ensure that your methods compile without syntax errors, you must enclose your variable template elements (that is, the entity type name, date, patch number, and user id) within stroke asterisk (/*) and asterisk stroke (***/**) character strings.

   As the variable options are case-sensitive, ensure that you specify these exactly as shown in this table; that is, **#<*entity-type*>Name** (for example, **#methodName**), **#date** , **#patchNumber**, and **#userId**.

   You can specify other elements between the stroke asterisk (**/***) and asterisk stroke (***/**) character strings in a text template; for example, if you were to specify a **Usage:** element in a method template, developers could then enter free-format text to document the usage of the method.

   The following example shows the text template for a new Jade method.

```
/*
Date created: #date
Created by:   #userId
Method Name:  #methodName
Patch Number: #patchNumber

Usage:
*/
vars

begin

end;
```

Notes As the editor pane displays only what was defined in your method template, the **vars**, **begin**, and **end**; statements in the above example were also specified in the template, following the asterisk stroke (***/**) character strings.

When adding a condition method using a template, the text template strips out the **vars** and **epilog** sections if they had been included in your template.

The following example shows the text template output for a new property.

```
**************************
Date created:   23 February 2018
Created by:     Wilbur
Property name:  salePrice
Patch Number:
Usage:
**************************
```

2. To define or modify a text template for a class, click the **Class** tab and then enter in the text box the text that your require for all classes; for example, your Jade class naming conventions.

   Alternatively, click the **Property** tab, **Constant** tab, or **Interface** tab, and then enter in the text box the text that your require for all properties, class constants, or interfaces, respectively.

All Jade classes, properties, class constants, methods, and interfaces that you then define in the current Jade development database have the elements specified in your template automatically displayed in the editor pane when you select the applicable **Add** or **New**… command from the corresponding Class or Primitive Types Browser menu (for example, the **Add** command from the Classes menu or the **New Jade Method** command from the Methods menu).

Maintaining Window Options

In Jade windows, the various components of your windows are displayed in different colors by default; for example, all superschema objects are displayed in blue and system objects are displayed in red.

To change your default window options

1. In the Preferences or Jade Installation Preferences dialog, click the **Window** sheet.
2. Double-click the component whose default color you want to change. For example, double-click the white panel to the right of **Background Color** in the Window list to change the default background color for your Jade windows. The common Color dialog is then displayed.
3. Click on the required color. (This common dialog also enables you to define your own custom color, if required, by double-clicking on the table and then selecting the color that you require or defining your own custom color on the Color dialog.)
4. Click the **OK** button.

   Repeat steps 2 through 4, to make all your required color display changes.
5. If you want to save the size and position of windows that you resize or reposition, check the **Save Size and Position** check box.

   If you have more than one open copy of a window (for example, the Class Browser), only the size and position of the last window to be opened is saved.
6. If you do not want the toolbar displayed in the Jade window, uncheck the **Show Tool Bar** check box.

7. If you do not want to display the status line at the bottom of the Jade window, uncheck the **Show Status Line** check box.

8. If you do not want to display bubble help, uncheck the **Show Bubble Help** check box.

9. If you want to display the backdrop image on the Jade Platform development environment background form, check the **Show Backdrop** check box.

10. If you do not want the Tips dialog displayed when you sign on to the Jade Platform development environment, uncheck the **Show Tips At Start-up** check box. For details about the Tips dialog, see "Signing On to the Jade Platform Development Environment", earlier in this chapter.

11. If you do not want to display the Jade release note splash screen when you sign on to the Jade Platform development environment, uncheck the **Show Release Note At Startup** check box.

    When this check box is checked, the release note splash screen, which displays the major features of the current release and hyperlinks to further information, is displayed:

    - The first time an upgraded Jade system is started, regardless of the value of the **ShowSplashScreen** parameter in the [Jade] section of the Jade initialization file.
    - The first time a new installation of Jade is started.
    - Every time a system with no user-defined schemas is started, unless the value of the **ShowSplashScreen** parameter in the [Jade] section of the Jade initialization file is set to **false**.

12. From version 2018.0.01, if you want to show or hide the display of alternating rows in list boxes and tables in browser-type forms (for example, the Class Browser) using the value of the **backColor** property, check or uncheck the **Show Alternating Row BackColor** check box.

13. If you do not want the editor clipboard toolbar or the floated Jade Clipboard Text Contents form displayed in the Jade window, uncheck the **Show Clip Board Toolbar** check box on the **Window** sheet of the Preferences dialog (or select the **Show Clipboard Toolbar** command in the View menu or context menu).

    If the editor clipboard toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the editor clipboard toolbar.

14. If you do not want the quick inspect toolbar or the floated Quick Inspect form displayed in the Jade window, uncheck the **Show Quick Inspect Toolbar** check box on the **Window** sheet of the Preferences dialog (or select the **Show Quick Inspect Toolbar** command in the View menu or toolbar context menu).

    If the quick inspect toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the quick inspect toolbar.

15. If you want to change the width of the quick inspect toolbar combo box from the default value of 300 pixels, in the **Quick Inspect Width** text box, specify the number of pixels you require, with a minimum value of 200 pixels and a maximum of 1,000 pixels.

16. Click the **Font** button if you want to change the default window font from Tahoma, regular, 8.25 points. The common Font dialog is then displayed, to enable you to make your font selections.

    When you have made your font selections, focus is then returned to the Preferences or Jade Installation Preferences dialog.

17. If you want to change between a light theme or a dark theme, select the **Light Mode** or the **Dark Mode** option button in the Color Mode group box. (The Preference dialog sheets then reflect the selected theme.)

18. For details, see "Displaying a Light or Dark Development Environment Theme", earlier in this chapter.

19. If you want to change the theme from the default Jade theme, select the theme that you require in the **Select Jade Theme** combo box.

20. If you want to remove or change the series of images applied to the caption line, menu line, and border areas of each form in the Jade Platform development environment to provide an enhanced look and feel to each form in the current (committed) version, select **<None>** or the skin that you require in the **Select Jade Skin** combo box.

    The skin can also define images for buttons, **JadeMask** controls, check boxes, and option buttons to further enhance the look and feel of forms. By default, the **Lincoln** skin is used for the current version; that is, **Jade Lincoln** is displayed in this combo box.

    Note As Jade must be able to upgrade existing systems by replacing the Jade system files, you cannot update the skins used by the Jade Platform development environment and you cannot access user-defined skins for deployed applications from the development environment.

    However, you can create your own skin images for definition and maintenance by users of the runtime applications, if required. For details, see "Defining and Maintaining Jade Skins" under "Using Jade Skins in Your Runtime Applications", in Chapter 1 of the *Runtime Application Guide*. See also Chapter 9, "Using Skins to Enhance Jade Applications", in the *Developer's Reference*.

21. Click the **Add Theme** button if you want to add a theme. The Define Jade Theme dialog is then displayed. For details, see "Adding a Theme", later in this chapter.

22. Click the **Edit Theme** button if you want to edit an existing user theme. The Edit Jade Theme dialog is then displayed. For details, see "Editing a Theme", later in this chapter.

    This button is disabled when **Default Jade Theme** or **Default Dark Jade Theme** is selected in the **Select Jade Theme** combo box.

23. Click the **Reset** button if you want to reset the current Jade themes Window, Editor, and Relationship colors to those of the default Jade theme for the currently selected color mode.

    This button is disabled when **Default Jade Theme** or **Default Dark Jade Theme** is selected in the **Select Jade Theme** combo box.

24. Click the **Remove Theme** button if you want to remove the user theme selected in the **Select Jade Theme** combo box.

25. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

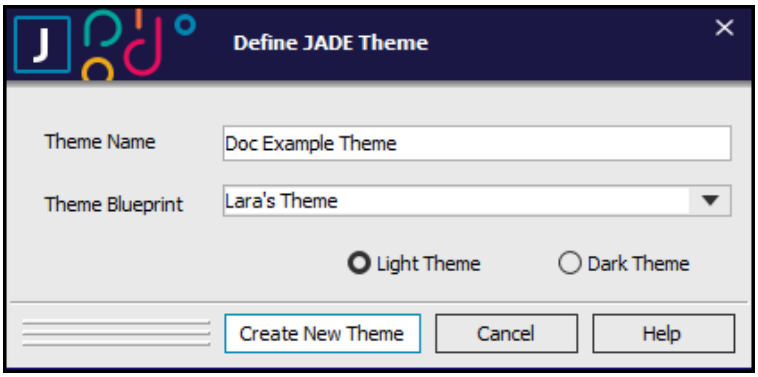
The current window then has focus, and displays your selected preferences.

Adding a Theme

The **Window** sheet of the Preferences dialog enables you add your own theme to the development environment if you do not want to use the default Jade theme.

To add your own theme

1. Click the **Add Theme** button. The Define Jade Theme dialog, shown in the following image, is then displayed.



2. In the **Theme Name** text box, specify the name of your new theme.
3. In the **Theme Blueprint** drop-down list box, select the theme that is to be the base for your new theme.
4. If you do not want a light theme (the default), select the **Dark Theme** option button.
5. Click the **Create New Theme** button, to add your new theme, which is then displayed in the **Select Jade Theme** combo box on the **Window** sheet of the Preferences dialog.

   Alternatively, click the **Cancel** button to abandon your selections and return focus to the **Window** sheet.

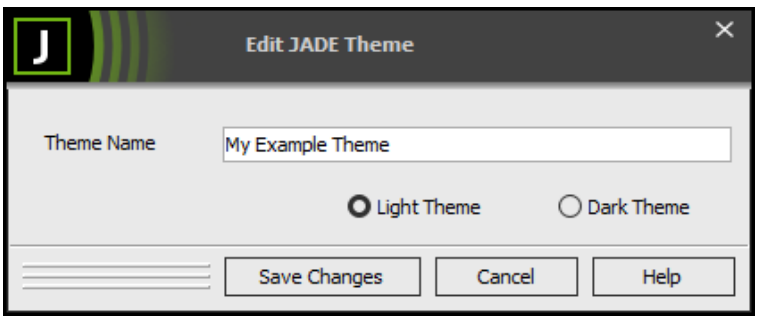
Applies to Version: 2020.0.01 and higher

Editing a Theme

The **Window** sheet of the Preferences dialog enables you edit a user-defined theme. (You cannot edit the default Jade theme.)

To edit your own theme

1. Click the **Edit Theme** button. The Edit Jade Theme dialog, shown in the following image, is then displayed.



2. In the **Theme Name** text box, specify the new name for your theme, if required.
3. If you want to change from the light or dark theme, select the required option button.
4. Click the **Save Changes** button to update your theme, which is then updated in the **Select Jade Theme** combo

box on the **Window** sheet of the Preferences dialog.

Alternatively, click the **Cancel** button to abandon your selections and return focus to the **Window** sheet.

Note You cannot modify the name or update the color mode of a Jade theme template that is created by a system administrator in the Jade Installation Preferences dialog, and the **Default Jade Theme** (light) and **Default Dark Jade Theme** themes.

Applies to Version: 2020.0.01 and higher.

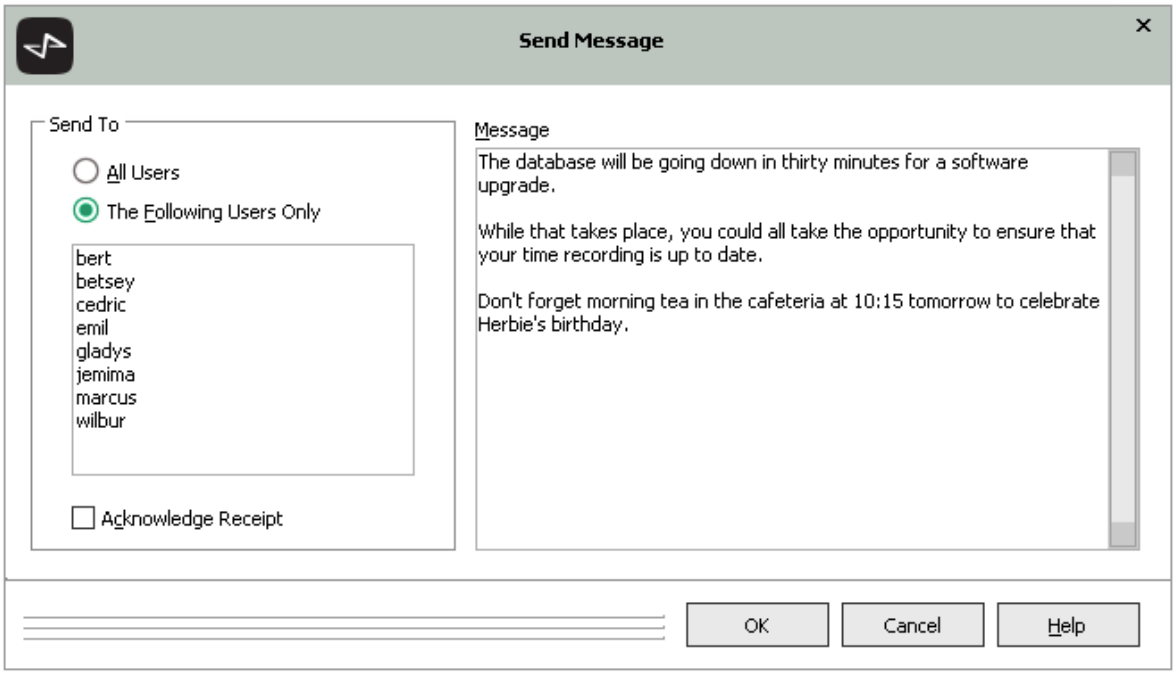
Sending Messages to Other Developers

Use the **Send Message** command in the Jade Platform development environment File menu to send a message to other developers who are connected to the Jade server.

To send a message

1. Select the Send Message command from the File menu.

   The Send Message dialog, shown in the following image, is then displayed.



   The user names of all developers who are currently logged on to Jade are displayed in the list box of the Send To group box. (These user names are the values in the **Username** text box on the Jade sign-on dialog.)

2. If you do not want to send the message to all users signed on to the same server, select the **The Following Users Only** option button. By default, messages are sent to all users.

3. If you chose to send the message only to specific users (by selecting the **The Following Users Only** option button in step 2 of this instruction), select the user or users to whom the message is sent from those in the Send To group box list. Use the Ctrl key to make multiple user selections, if required.

4. If you want to be advised when each recipient clicks the **OK** button in the Message Received dialog, check the **Acknowledge Receipt** check box. By default, you are not advised each time a recipient views your message.

   If the **Acknowledge Receipt** check box on the Send Message dialog was checked, a modal message box pops up on your workstation each time a recipient clicks the **OK** button. This message box displays the name of the recipient who acknowledged the message you sent. Click the **OK** button when you have read the name of the recipient who has viewed your message.

5. In the **Message** text box, enter the message that you want to send to send to all developers or to your selected developers. The message text can be of an arbitrary length.

6. Click the **OK** button to send your message to the selected developers. Alternatively, click the **Cancel** button to abandon your selections.

   A modal Message Received dialog then pops up on the workstation of each user to whom you addressed the message. This dialog displays the user code of the sender (to avoid anonymous messages) and the date and time that it was sent.

7. When you have viewed a received message, click the **OK** button to close the dialog.

Maintaining Debugger Breakpoints from the Editor Pane

The Jade Platform development environment provides functionality for setting and maintaining debugger breakpoints from the editor pane, as follows.

- If the line number display or folding is enabled in the editor pane for a Jade method, a column is added to the display in the left margin after the line number and before the folding margin; for example:

```
1   setUp() updating, unitTestBeforeClass;
2
3   vars
4
5   begin
6       if Company.firstInstance() <> null then
7           self.tearDown();
8       endif;
9
10      beginTransaction;
11      self.company := create Company();
12      commitTransaction;
13
14      self.buildCompany(company);
15
16      self.retailSale := company.allSalesByItem.first();
17      self.tenderSale := company.allSalesByItem.last();
18
19  end;
```

- When an enabled breakpoint is set on a line, a red circle is displayed in the breakpoint margin. Clicking on an enabled breakpoint (indicated by a red circle) removes the breakpoint.
- When a disabled breakpoint is set on a line, a white circle is displayed in the breakpoint margin. Clicking on a disabled breakpoint (indicated by a white circle) enables the breakpoint.
- Clicking on an empty breakpoint margin for a line sets a breakpoint.

- Right-clicking in the breakpoint margin displays a breakpoint menu that has the following commands.
  - **Clear All Breakpoints From Method**.
  - **Breakpoint Options**, which initiates the Breakpoint Options dialog that allows you to disable or enable a breakpoint, set a pass count, and set a conditional expression for the breakpoint. For details, see "Using Conditional Breakpoints and Pass Counts", in Chapter 7.

    The Breakpoint Options dialog provides a large **Condition** control that allows for multiple-line input and AutoComplete functionality. The dialog can also be resized.
  - **Disable** or **Enable**, which toggles the disabled or enabled status of the current breakpoint.
  - **Remove Breakpoint**.

  The **Disable Breakpoint** or **Enable Breakpoint** and the **Breakpoint Options** commands are disabled if the method has been changed and not compiled or if there is no breakpoint on that line.
- Hovering the mouse over a breakpoint indicator displays a bubble help window with the breakpoint details; for example, **Disabled breakpoint**.
- The Jade Platform development environment editor pane right-click breakpoints menu includes the ability to invoke the Breakpoint Options dialog and to clear all breakpoints.

A breakpoint always shows an icon in the breakpoint column of the user method in the development environment editor pane and the Jade Debugger method source window.

The Preferences dialog **Editor Options** sheet now provides the **Highlight entire source line for breakpoints** check box that controls whether the display of a breakpoint highlights the whole source line for a breakpoint. (This check box is checked by default.)

Applies to Version: 2022.0.01 and higher

Performing Edit Actions

Use the commands in the Edit menu from Browser windows for editing functions; for example, a copy or a search function.

Edit commands are disabled if the Schema Browser has the focus.

The Edit menu commands are listed in the following table.

Command	For details, see…
Undo	Undoing an Action, later in this chapter
Redo	Redoing an Action, later in this chapter
Cut	Cutting Text, later in this chapter
Copy	Copying Text, later in this chapter
Paste	Pasting Text, later in this chapter
Select All	Selecting All Editor Pane Content, later in this chapter
Macro	Recording and Replaying a Series of Keystrokes, later in this chapter
Find/Replace	Searching for or Replacing an Element in a Schema, in Chapter 3
Find Again	Finding the Next Occurrence of Specified Text, in Chapter 3

Command	For details, see…
Find Again Reverse Direction	Searching for Text in a Reverse Direction, in Chapter 3
Find At Caret	Locating Text on Which the Caret is Positioned, in Chapter 3
Quick Inspect Focus	Using the Quick Inspect Toolbar, in Chapter 6
Global Find/Replace	Searching for an Element in all Classes in the Current Schema, in Chapter 3
Refactor	Refactoring a Jade Method, in Chapter 4
Replace Indent	Replacing Editor Pane Indentation, later in this chapter
Usages	Displaying Identifier Usages from the Editor Pane, earlier in this chapter

Edit menu commands that are not available for selection are disabled. For example, if you have not selected any text in the editor pane, the **Copy**, **Cut**, and **Paste** commands are disabled so that you cannot select them.

Tip To quickly access the Edit menu without having to click the Edit menu on the menu bar when the editor pane is the current window, right-click to access the context (popup) menu. Alternatively, click an edit function toolbar icon.

Undoing an Action

Use the Edit menu **Undo** command to undo your previous editor pane actions. You can undo the previous 256 editor operations, if required.

You can undo the following types of action.

- Last single keystroke
- Cut
- Paste

This command is disabled when a command cannot be undone; that is, you have not previously performed an action in the editor pane that can be undone.

» To undo your last action, perform one of the following actions

- Select the **Undo** command from the Edit menu
- Press Ctrl+Z
- Press Alt+Backspace

Your previous action is then undone. Repeat this action to undo up to 256 previous actions, if required.

Redoing an Action

Use the Edit menu **Redo** command to redo previous actions that you have undone in the editor pane by using the **Undo** command. You can redo up to 256 editor operations, if required. This command is disabled when a command cannot be redone; that is, you have not previously performed an **Undo** action. (For details, see "Undoing an Action", in the previous subsection.)

To redo the next undone action, perform one of the following actions

- Select the **Redo** command from the Edit menu
- Press Ctrl+Y

The last undo action is then reversed.

Cutting Text

Use the Edit menu **Cut** command to logically delete a selected portion of a method in the current editor pane. This command is disabled when there is no selected text to be cut.

Select the portion of the method that you want to delete. The selected portion of the method is then highlighted.

You can physically delete whole methods only by using the **Remove** command from the Methods menu. (For details, see "Removing a Schema Element", in Chapter 3.)

To cut the selected portion of a method, perform one of the following actions

- Select the **Cut** command from the Edit menu
- Press Ctrl+X
- Click the **Cut** icon in the Browser toolbar

The selected portion of text is then removed from the current method.

When you use the **Cut** command, the selected portion of the method is copied to the clipboard. The clipboard is not saved between Jade sessions.

Copying Text

Use the Edit menu **Copy** command from any editor pane to copy a selected portion of a method to another position in the current method or to another method or the contents of the current list window to the clipboard. This command is disabled when there is no selected portion of the current method to be copied or a list window does not have focus.

Select the portion of the method or the list window that you want to copy. The selected portion of the method or an entity in a list window is then highlighted.

Note If focus is in a list window that has a hierarchical structure of nodes, only those entities that have been displayed since the browser window was opened are copied. For example, if focus is in a Class List and no collapsed nodes have been expanded, only the highest level classes are copied. Conversely, if one or more nodes has been expanded and subsequently collapsed while the browser window had focus, all lower-level entities that have been displayed are copied to the clipboard.

To copy a selected portion of a method or a list window, perform one of the following actions

- Select the **Copy** command from the Edit menu
- Press Ctrl+C
- Click the **Copy** icon in the Browser toolbar

The selected portion of the method or the entities in the list window are then copied to the clipboard. The clipboard is not saved between Jade sessions.

You can also copy and paste content using multiple carets and rectangular selection. For details, see "Editing Multiple Selections and Rectangular Selections", later in this section.

Pasting Text

Use the Edit menu **Paste** command from any editor window to paste a selected portion of a method from the clipboard into the current method. This command is disabled when there is no selected portion of a method in the clipboard.

Note You can paste a portion of a method to the same or a different method, or a complete method to a different method.

You can also copy and paste content using multiple carets and rectangular selection. For details, see "Editing Multiple Selections and Rectangular Selections", in the following section.

Before you can paste a portion or a complete method to the same or a different method, you must first have copied or cut the required code to the clipboard. (For details, see "Copying Text" or "Cutting Text", earlier in this section.)

To paste of a method or full method from the clipboard

1. Position the caret in the current method at the position in which the paste operation is to begin. If you are pasting a complete method into a newly defined method, you will first have to create the new Jade method, by using the **New Jade Method** command from the Methods menu.
2. Paste the code from the clipboard, by performing one of the following actions.
   - Select the **Paste** command from the Edit menu
   - Press Ctrl+V
   - Click the **Paste** icon in the Browser toolbar

   The code from the clipboard is then pasted into the current method, with the first part of the code located at the caret position.
3. If you have pasted a complete method into an "empty" method, now modify the method signature, and remove the **vars**, **begin**, and **end** statements.

When you use the **Paste** command, the code is copied from the clipboard. If you cut any code from a method (by using the Edit menu **Cut** command), any subsequent paste operations will use the text from that cut action.

The clipboard is not saved between Jade sessions.

Editing Multiple Selections and Rectangular Selections

The editor pane supports the following behaviors to select and edit content (in addition to single caret selection).

- Multiple caret selection and editing
- Rectangular selection

This section uses the terms *stream format* and *rectangular format* to distinguish between content copied by using single or multiple carets, and rectangular selection, respectively.

Multiple Carets

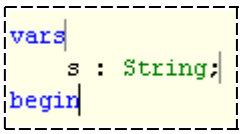
You can create multiple carets and select multiple (non-contiguous) ranges in the editor pane, as follows.

- Multiple carets enable you to enter text at multiple locations at the same time.
- Multiple selections enable you to perform edit actions on selected (non-contiguous) content at the same time.

To create additional carets

- Press Ctrl+left-click in each location at which you want to add a caret.

The main caret is the color specified in your user preferences (that is, the **Caret** component on the **Editor** sheet of the Preferences dialog) and additional carets are gray, as shown in the following image.



Note The main caret is the last one that you add.

You can then enter text, paste content, and perform the **Undo** and **Redo** actions at each caret at the same time.

To make multiple selections

- Press Ctrl and then click and drag your cursor to select multiple lines of content.

You can then enter text and perform the **Copy**, **Cut**, and **Paste** actions on selected content at the same time.

Multiple selection copying concatenates the selections in the order in which you made each selection. One instance of the concatenated selections is then pasted in stream format at each caret. Alternatively, if you paste into a rectangular selection, a single instance of the concatenated selections is pasted for each line of the rectangular selection.

Rectangular Selection Virtual Space

Rectangular selection functionality takes into account virtual space in the selected region.

You can create a rectangular selection anywhere within the text in the editor pane. You cannot extend the selection beyond the end of the text.

Note A rectangular selection overrides any single caret or multiple caret selections you have made in the editor pane.

Create a rectangular selection, by pressing the:

- Alt key when dragging the mouse
- Shift+Alt+arrow keys

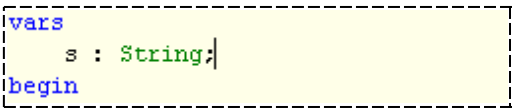
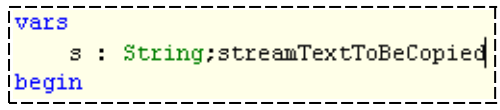
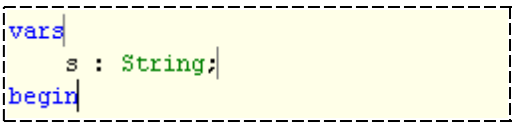
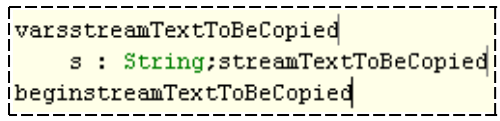
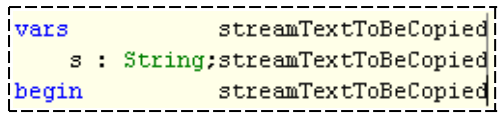
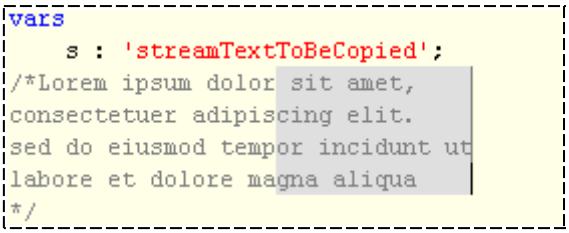
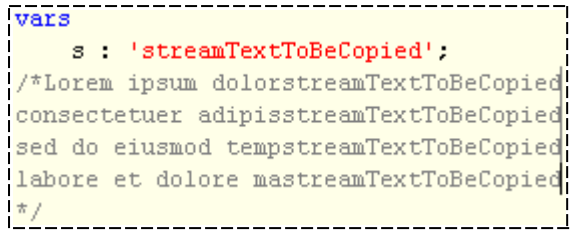
You can then perform the **Copy**, **Cut**, and **Paste** actions.

Pasting Behavior

This section shows examples of how pasting behaves for content selected and copied at single and multiple carets (stream format) or as a rectangular selection (rectangular format).

Stream format pasting examples

The following table lists examples of pasting stream format text (**streamTextToBeCopied**) into different destinations.

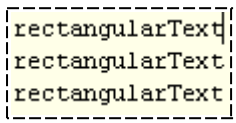
Pasting Destination	After Pasting
Single caret	
	
Multiple carets	
	
Rectangular selection with no content	
	
Rectangular selection with content	
	

Rectangular format pasting examples

Pasting rectangular format content into a virtual region adheres to the value you have specified for the **Use Tabs For Indent** check box on the **Editor Options** sheet of the Preferences dialog.

When you paste rectangular format content into a rectangular selection, the copied content is pasted from the top left position of the rectangular region and the caret is positioned at the end of the first line to ensure that consecutive paste actions align.

The following table lists examples of pasting rectangular format text in to different destinations.

Pasting Destination	After Pasting
Rectangular selection is smaller than pasted text	
	

Pasting Destination	After Pasting
Rectangular selection is larger than pasted text	

```
begin
    s := 'rectangularText
         rectangularText
         rectangularText';
```

```
begin
    s := 'rectangularText
         rectangularText   rectangularText
         rectangularText';rectangularText
                          rectangularText
```

Copying content in a rectangular selection copies the entire selected region including whitespace, but ignores virtual space. When you paste the rectangular format content into a rectangular selection, the virtual space is ignored and only text and whitespace is pasted, as shown in the following example.

Copied Region	After Pasting
Copied rectangular region includes white space	Virtual space is ignored and only text and whitespace is pasted

```
s := 'rectangularText
     rectangularText
     rectangularText';
```

```
s := 'rectang';ularText
     rectangularText
     rectangularText';
```

Editor Clipboard Notes

The Jade Platform development environment editor clipboard toolbar supports copying and pasting of selected rectangular content in the clipboard buffers **C**, **1** through **9**, or zero (**0**). The rectangular state of copied content is now retained in the clipboard buffer.

Additional clipboard toolbar items do not support selected rectangular content, which is pasted in stream format.

For details, see "Using the Editor Clipboard Toolbar", earlier in this chapter.

Selecting All Editor Pane Content

Use the Edit menu **Select All** command from the Class Browser, Primitive Types Browser, or Interface Browser to select the complete contents of the current editor pane. For example, you can select a complete method to copy it to the clipboard before pasting it into another method.

To select the contents of the editor pane, perform one of the following actions

- Select the **Select All** command from the Edit menu
- Press Ctrl+A

The contents of the current editor pane are then selected; that is, highlighted. (For details about cutting or copying the selected method to the clipboard, see "Copying Text" or "Cutting Text", earlier in this section.)

Replacing Editor Pane Indentation

If editor options specify that tabs are used for editor pane indentation (that is, the default check mark is displayed in the **Use Tabs For Indent** check box in the Entry Options group box on the Editor Options sheet of the Preferences dialog), the Edit menu **Replace Indent** command from the Class, Primitive Types, or Interface Browser enables you to replace leading spaces, if present, with tabs.

Conversely, if editor options specify the use of spaces for editor pane indentation, leading tabs are replaced with spaces.

To replace indentation

- Select the **Replace Indent** command from the Edit menu.

  The **Replace Indent** command is disabled if a method is not displayed in the editor pane.

The editor pane indentation is then replaced with leading spaces or with tabs, as applicable. You can undo this action. Note, however, that replacing the indentation or undoing the replacement indentation does not save the method source displayed in the editor pane. For details about specifying editor options, see "Maintaining Editor Options", earlier in this chapter.

Recording and Replaying a Series of Keystrokes

You can record and replay a series of keystrokes in a method source editor pane.

Tip You can bind a keystroke combination (for example, Ctrl+Shift+F5) to the **OpenMacroLibrary** command action on the **Short Cut Keys** sheet of the Preferences dialog, to quickly invoke the **Library** command from the Macro submenu.

The **Macro** command enables you to perform the actions documented in the following subsections.

- Starting to Record the Temporary Macro
- Stopping the Recording of the Temporary Macro
- Playing the Temporary Macro
- Editing the Temporary Macro
- Selecting a Persistent Macro from the Library

See also "Keystroke Macro Language", later in this chapter.

The **Macro** command in the Edit menu is enabled only when a method source editor pane has focus.

Each method source editor pane has its own separate temporary keystroke macro buffer. Each pane of a split method source editor has a separate temporary keystroke macro buffer.

Persistent saved macros are stored with your user preferences. Use the import and export facilities to share saved macros with other users or with other Jade environments. For details, see "Editing the Temporary Macro", later in this chapter.

You cannot:

- Embed macros within macros
- Record or play another macro when recording is in progress

- Record or play another macro while a macro is playing

Macro replay halts when the caret is moved outside the document limits (before the start or after the end of the method in the editor pane) or when a **Find** command has no match.

For details about configuring Jade shortcut keys and editor key bindings, see "Maintaining Shortcut Keys" under "Setting User Preferences", earlier in this chapter.

Starting to Record the Temporary Macro

To start recording keystrokes as the current temporary macro

1. Select the **Macro** command from the Edit menu and then select the **Start Macro Record** command from the submenu that is displayed.
2. Type the keystrokes that you want to store as a temporary macro for subsequent use in the editor pane during the current work session.

Your keystrokes are then stored, to enable you to replay them in the editor pane of the same method source or another method source, by selecting the **Play Temp Macro** command from the **Macro** command submenu.

The **Stop Macro Record** command is the only **Macro** submenu command that is enabled while keystroke macro recording is in progress.

Stopping the Recording of the Temporary Macro

To stop recording keystrokes as the current temporary macro

- Select the **Macro** command from the Edit menu and then select the **Stop Macro Record** command from the submenu that is displayed.

The recording of keystrokes is then stopped.

Playing the Temporary Macro

You cannot play a macro when recording is in progress or when another macro is playing.

To play the current temporary macro

- Select the **Macro** command from the Edit menu and then select the **Play Temp Macro** command from the submenu that is displayed.

The current temporary macro is then played in the editor pane.

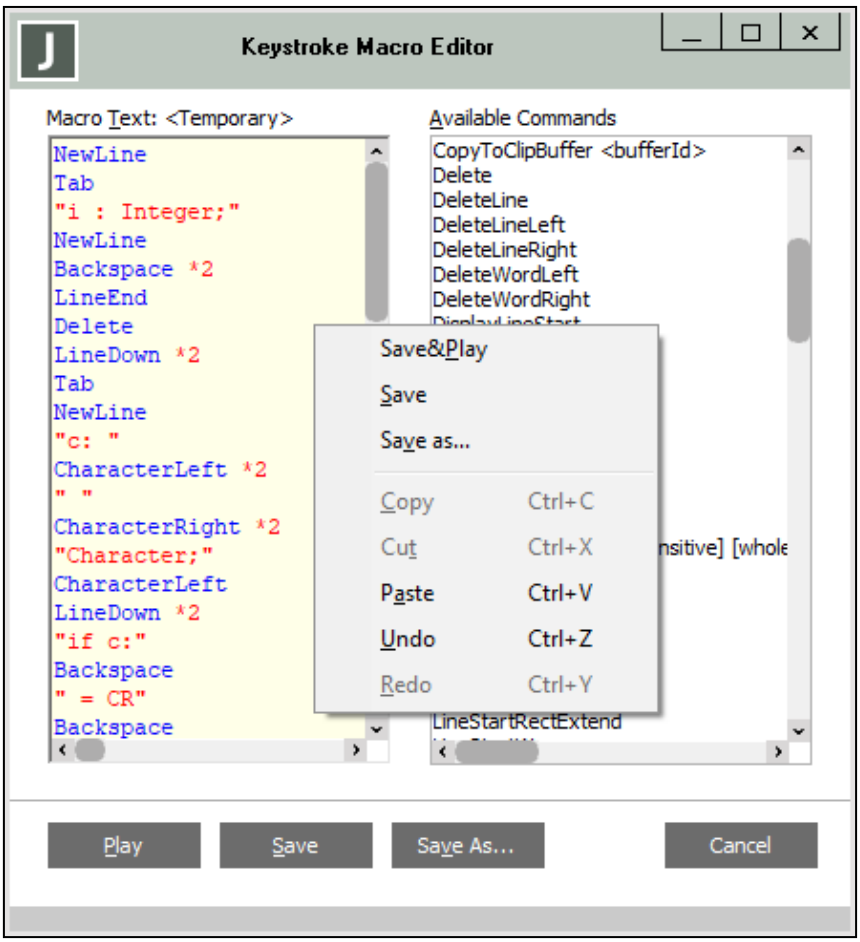
Macro replay halts when you move the caret outside the document limits (before the start or after the end of the method in the editor pane) or when a **Find** command has no match.

Editing the Temporary Macro

To edit the current temporary macro

1. Select the **Macro** command from the Edit menu and then select the **Edit Temp Macro** command from the submenu that is displayed.

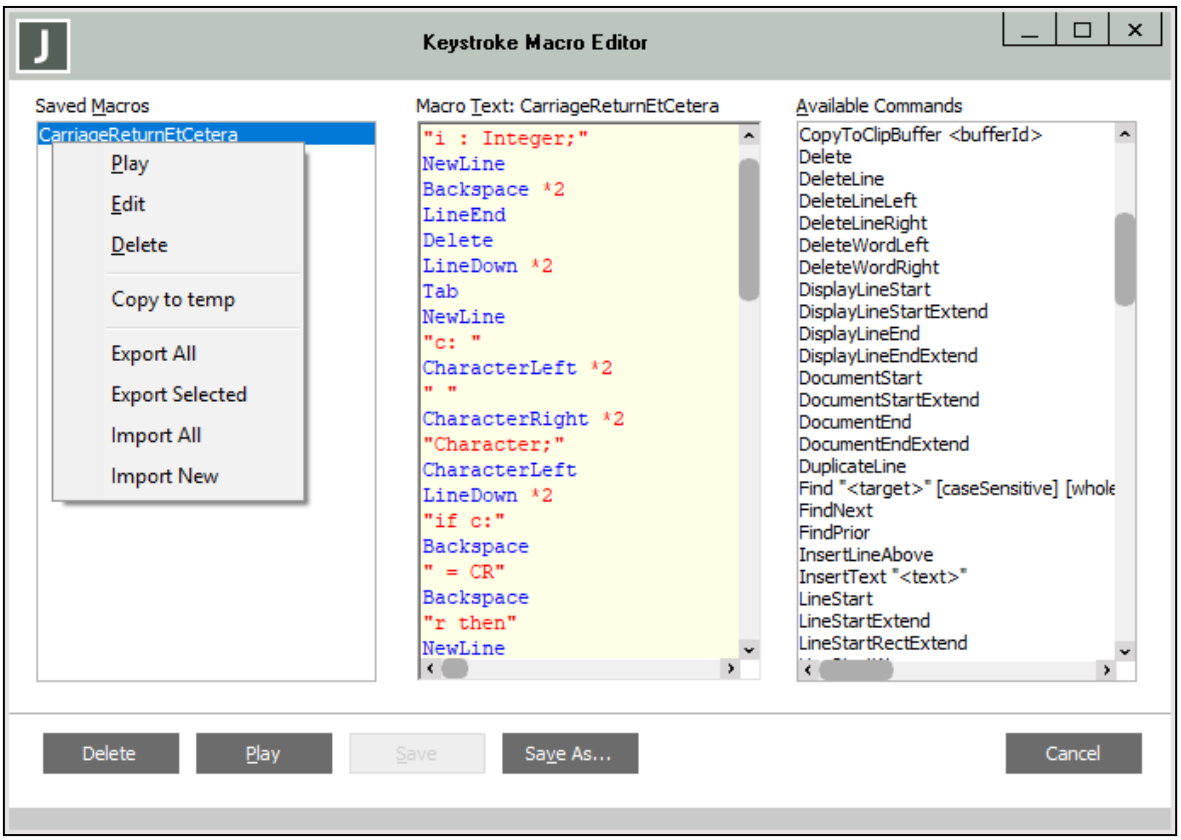
The Keystroke Macro Editor dialog, shown in the following image, is then displayed with the current temporary macro in the macro editor pane at the left of the dialog.



2. If the current macro is the temporary macro, select the **Save as** command from the editor pane context menu or click the **Save As** button, to display the Save Persistent Keystroke Macro dialog, which enables you to specify the name that you require for the persistent macro.

   Conversely, if you have edited an existing persistent macro, select the **Save** command from the editor pane context menu or click the **Save** button, to update the macro.

When you have saved the temporary macro as a persistent macro, it is displayed in the **Saved Macros** list box, as shown in the following image.



The **Saved Macros** list box is displayed only when you perform one of the following actions.

- Select the **Save as** command from the editor pane context menu
- Click the **Save As** button
- Click the **Edit** button on the Keystroke Macro Editor dialog that is displayed when you select the **Library** command from the **Macro** submenu

The middle pane (the editor pane) shows the macro text for the selected macro. Syntax highlighting, using your current user preference colors, indicates correct lines. The color of:

- Keywords (blue, by default) is used for the command words and command option words
- Numeric literals (red, by default) is used for digits
- String literals (red, by default) is used for quoted text
- Comments (gray, by default) is used for comment lines
- Identifiers (black, by default) is used for unrecognized or invalid text

You cannot save or play a macro that has errors. A save attempt is refused and the first unrecognized or invalid line is displayed as selected text.

The **Available Commands** list box displays the commands that you can add to the macro. These commands include a list of all possible parameters, where appropriate. Command parameters enclosed in square brackets are optional. Optional parameters separated by vertical bars are mutually exclusive. For details, see "Keystroke Macro Language", later in this chapter.

3. If you want to insert a command into the macro, perform one of the following actions in the **Available Commands** list box.
   - Double-click on the command.
   - Right-click on the command and then select the **Insert in macro** command.
   - Select the command and then press Enter or the space bar.

   The command is then inserted in the macro at the caret position.
4. Click on an entry in the **Saved Macros** list box to display another persistent macro.

   If the existing macro has unsaved changes, a confirmation dialog is displayed, asking whether to save or discard the changes.
5. When the **Saved Macros** list box has focus, perform one of the following actions to play the selected macro and then close the Keystroke Macro Editor dialog.
   - Click the **Play** button.
   - Right-click and then select the **Play** command from the popup menu.
   - Press Enter or the space bar.

Popup menus in the **Macro Text** pane and the **Saved Macros** list box provide the commands listed in the following table in addition to the standard edit actions available in the **Macro Text** pane (for example, copy and undo actions).

Command	Description
Play	Plays the selected macro and then closes the dialog.
Save	Saves the changes to the selected macro.
Save as	Displays the Save Persistent Keystroke Macro dialog, enabling you to specify the name that you require for the persistent macro.
Save&Play	Saves the selected macro, plays it, and then closes the Keystroke Macro Editor dialog.
Edit	Displays the selected macro in the **Macro Text** pane.
Delete	Deletes the selected macro. (A confirmation dialog is displayed.)
Copy to temp	Copies the contents of the selected macro so that you can save it as the current temporary macro.
Export All	Writes all saved persistent macros to the file specified in the common Save As dialog.
Export Selected	Writes the selected persistent macro to the file specified in the common Save As dialog.
Import All	Displays the common Open dialog to enable you to select the file of exported macros you want to import, updates macros with existing names, and adds new macros.
Import New	Displays the common Open dialog, to enable you to select a file containing exported macros, adds the macros that do not already exist, and ignores the rest (for example, if you want to import a macro from another user and retain your own macros).

Selecting a Persistent Macro from the Library

To select a persistent macro to edit or play

- Select the **Macro** command from the Edit menu and then select the **Library** command from the submenu that is displayed.

The Keystroke Macro Editor dialog, providing only the **Saved Macros** list box, is then displayed.

Select the persistent macro that you want to play or edit, and then click the **Play** or **Edit** button, respectively. You can also press Enter or the space bar to play the selected macro.

When you click the **Play** button or press Enter or the space bar, the macro is played in the current method source editor pane, and the Keystroke Macro Editor dialog is then closed.

When you click the **Edit** button, the extended Keystroke Macro Editor dialog is displayed. For details about editing a macro, see "Editing the Temporary Macro", in the previous section.

Keystroke Macro Language

A keystroke macro consists of one or more command lines.

Lines that are empty or that contain only spaces or tabs are ignored.

Lines that have a forward slash character (*/*) as the first non-blank character are ignored, and can be used for comments.

Each line contains a single command followed by an optional repeat count (an asterisk followed by a number); for example, the following command line moves the caret four characters to the right.

If the repeat count is zero (**0**), the command is not executed.

```
CharacterRight*4
```

Most commands equate to a single keystroke. Some commands are not bound to keystrokes, by default, and cannot be recorded (for example, **TransposeLine**).

Command names that end with **Extend** modify the current selection.

Some keystroke macro commands are listed in the following table.

Command	Details
"quoted text"	Text keystrokes. Double quote (**"**) and backslash (\) characters are escaped from with a leading backslash; for example, **"\""** and **"\\"** indicate one double quote and one single backslash character, respectively.
CopyToClipBuffer <bufferId> PasteFromClipBuffer <bufferId>	The **<bufferId>** parameter defines which of the ten clip buffers to use and it must be a single digit in the range **0** through **9**.
Find "<target>" <options>	Searches for the next occurrence of the specified **"<target>"** parameter value, and if found, selects it. The **<options>** parameter can be one or more of the following values. ■ **caseSensitive**, for a case-sensitive search.

Command	Details
	■ **wholeWord**, to exclude partial word matches. ■ **wordStart**, to match the target to the beginning of words. ■ **unslash**, **regexpr**, or **posixRegexpr**, to indicate that the target contains backslash escape characters, a regular expression, or a Posix regular expression. The default value is literal text. ■ **backwards**, to search back towards the start of the document. The default value is forward to the end of the document. ■ **fromCaret** or **inSelection**, to select the search start point. The default value is the whole document.
FindNext or FindPrior	Searches for the next or prior occurrence of the current search target. The direction of these commands reverses if the **Find** command included the **backwards** option.
PasteDynamic *name*	The ***name*** parameter is the name of the dynamic text element to insert. The Jade Platform development environment supplies the **className**, **date**, **date_ymd**, **methodName**, **patchNumber**, **schemaName**, **time_hms**, **timestamp**, and **userId** dynamic elements.
Uppercase or Lowercase	Changes the case of the selected text.
AddText "<text>"	Inserts the specified text after the current position, which remains unchanged.
AppendText "<text>"	Adds the specified text to the end of the existing text and moves the caret to the end of the new text and into view.
InsertText "<text>"	Inserts the specified text before the current caret position and moves the caret to the end of the added text, but does not force it into view.
PasteText "<text>"	Deletes the selection (if any) and then inserts the text specified in the **"<text>"** parameter at the caret, leaving the caret at the start of the text.
PushPosition	Pushes the caret location onto a stack that holds 20 entries. When the stack is full, the oldest entry is discarded.
PopPosition	Pops the top entry of the stack. If the stack was not empty, the caret is moved to that location.
ReplaceAll "<target>" "<new>" <options>	Replaces all occurrences of the target text with the new text. The values for the **<options>** parameter are those of the **Find "<target>" <options>** command.
ReplaceFind "<new>" <options>	Replaces the current find target with the new text and then searches for the next find target.
SelectWholeLines	Extends the current selection so that the start is at the beginning of the first line of the selection and the end is at the end of the last line of the selection.
SpaceToColumn <column>	Inserts whitespace to move the caret to the specified column. This command does nothing if the caret is already at or past the specified column.

In **Find**, **ReplaceFind**, and **ReplaceAll** commands **<target>** and **<new>** quoted text parameters, a backslash is used to escape the following characters.

- **\"** is one double quote
- \\ is one backslash

A backslash followed by any other character is those two characters. These escape sequences are interpreted before regular expression processing begins.

In a find target regular expression, the special characters that are interpreted are listed in the following table.

Character	Description
.	Matches any character.
\(	Marks the start of a region for tagging a match.
\)	Marks the end of a tagged region.
\n	The *n* parameter is a digit in the range **1** through **9**, which refers to the first through ninth tagged region when replacing; for example, if the search string was **Fred\([1-9]\)XXX** and the replace string was **Sam\1YYY**, this would generate **Sam2YYY** when applied to **Fred2XXX**.
\<	Matches the start of a word. A word is defined to be a character string beginning or ending, or both beginning and ending, with characters in the ranges A through Z, a through z, 0 through 9, and an underscore. In addition, it must be preceded or followed, or both preceded and followed, by any character outside those mentioned.
\>	Matches the end of a word.
\x	Enables you to use a character *x* that would otherwise have a special meaning; for example, **\[** would be interpreted as **[** and not as the start of a character set.
[...]	Indicates a set of characters; for example, **[abc]** means any of the characters a, b, or c. You can also use ranges; for example, **[a-z]** for any lowercase character.
[^...]	Complement of the characters in the set; for example, **[^A-Za-z]** means any character except an alphabetic character.
^	Matches the start of a line (unless used inside a set).
$	Matches the end of a line.
*	Matches zero (**0**) or more times; for example, **Sa*m** matches **Sm**, **Sam**, **Saam**, **Saaam**, and so on.
+	Matches one or more times; for example, **Sa+m** matches **Sam**, **Saam**, **Saaam**, and so on.

Example Keystroke Macros

This section contains examples of keystroke macros.

Comment Selection

The following macro adds **"//"** at the beginning of each line in the selected text.

```
SelectWholeLines
ReplaceAll "^" "//" regexpr inSelection
```

Uncomment Selection

The following macro removes the first **"//"** from the beginning of each line in the selected text.

```
SelectWholeLines
ReplaceAll "^//\([^/]\)"  "\1" regexpr inSelection
ReplaceAll "^///" "/" regexpr inSelection
```

Return to Saved Position

The following macro moves the caret back to the most-recently saved position.

```
PopPosition
```

Add an Entry to the Method Change History

The following macro:

1. Saves the current position.
2. Moves to the end of the method change history text (which is assumed to be bounded by a line of asterisks).
3. Adds a new change header including the current date, time, and user id.
4. Adds the test in clipBuffer 6, where the user saves the current change details for applying to multiple methods.

```
PushPosition
DocumentStart
Find "^\*\*\*\*" regexpr
InsertLineAbove
"--------------------------------------------------"
NewLine
"UPDATE: "
PasteDynamic timestamp
". BY: "
PasteDynamic userId
"."
NewLine
PasteFromClipBuffer 6
```

Jade Platform Online Help

This section covers the following topics relating to obtaining online help from the Jade Platform Product Information Library in HTML5 and Adobe Portable Document Format (PDF).

- HTML5 Online Help
- Product Information Library in Portable Document Format
- Obtaining Help in a Window or Dialog
- Obtaining Help in the Editor Pane
- Obtaining General Help
- Obtaining Jade Version Information
- Creating Context Links to Your Own Application Help File

Notes You cannot make changes to a Jade Platform HTML5 file opened in a browser or to a PDF file opened in Adobe Reader, but you can copy text and paste it into other applications.

HTML5 Online Help

The Jade Platform provides the product information library, the white papers, and the *Erewhon Demonstration System Reference* in HyperText Markup Language 5 format (HTML5) in addition to Portable Document Format (PDF) format. By default, context-sensitive help from the Jade Platform development environment is obtained from .**htm** topics in the HTML5 web format of the product information. For details, see the **UseJadeWebHelp** parameter and the **JadeHelpBaseUrl** parameter in the [JadeHelp] section of the Jade initialization file.

Access the HTML5 online help in your browser at:

https://secure.jadeworld.com/developer-centre/Jade2022/OnlineDocumentation/

The Jade Platform HTML5 online help enables you to:

- Access a specific topic in the web format of the product information library from the Jade Platform development environment, by pressing F1 from the entity whose information you want to view.
- Return to a summary of and access the documents in the product information library at any time, by clicking the dark gray jade text (which acts as the **Home** function), in the upper left of your browser above the Contents pane.
- Click an image thumbnail on a web page, to display the full-sized image. Conversely, to collapse an expanded image (for example, to reduce scrolling), click on the image again to restore the thumbnail format.
- Print a web page with the image expanded to the full size, by clicking the **Print** icon at the upper left of the topic pane, as shown in the following example of HTML5 for a Jade Platform 2022 release.

  Tip When you have printed a topic that contains an image, the image is not minimized to a thumbnail until you refresh the display (by pressing F5, for example) or you navigate to another topic and then back to the topic that you printed.

  When you click the **Print** icon at the upper left of a topic, only the topic itself is printed to fit the width of the output page. (Selecting **Print** from the browser menu also outputs the **Contents** pane at the left, which results in only about half of the page width for the topic, making it harder to read and necessitating a lot of scrolling.)

For details about formulating search expressions in the Jade Platform HTML5 help, see the **Search and Print Tips for HTML5 Help** topic, which is the second entity in the **Contents** pane at the left of your browser.

Applies to Version: 7.1.08 (Service Pack 7) and higher

Product Information Library in Portable Document Format

This section covers the following topics.

- Overview
- Getting Adobe Reader
- Obtaining Assistance when Using Adobe Reader
- Searching the Product Information Library using Adobe Reader
- Printing PDF Documents

Overview

Jade includes the Jade Platform product information library in web (HTML5) and in Adobe Portable Document Format (PDF) format. For details about specifying whether web (HTML5) or print (PDF) format is used as the default for context-sensitive help from the Jade Platform development environment, see the **UseJadeWebHelp** parameter and the **JadeHelpBaseUrl** parameter in the [JadeHelp] section of the Jade initialization file. See also "Jade HTML5 Online Help", earlier in this chapter.

Note The examples files are not part of the installation process, and must be downloaded and installed from the Jade website separately, if required.

Adobe PDF makes documents available across several platforms. You can open PDF files by using the Adobe Reader application that is freely available from the Adobe website, by using the licensed Adobe Acrobat application, or another PDF reader. Depending on the PDF reader that you are using, you:

- Can adjust PDF documents for optimal viewing, print and copy from them, and navigate quickly using features such as bookmarks and document links.
- Cannot make changes to a Jade PDF file opened in Acrobat Reader but you can copy text and paste it into other applications or you can add your own comments.

In this chapter, "Adobe Reader" is used to indicate both the full version of Adobe Acrobat (which allows the distilling, reading, and editing of PDF files) and Adobe Reader (which is a free product distributed for reading and printing PDF files). In either case, the procedures are the same.

For a summary of and access to the documents in the Jade Platform product information library, see the *Product Information Library* document (**Jade.pdf**), displayed when you select the **Index** command in the Jade Platform development environment Help menu or when you click the **General Help** toolbar button and the **UseJadeWebHelp** parameter in the [JadeHelp] section of the Jade initialization file is set to **false**.

» To access Jade documentation when web help is disabled, perform one of the following actions

- Select the **Index** command in the Help menu from within the Jade Platform development environment or a Jade utility, to access the directory of Jade Platform documentation.
- Click the **General Help** toolbar button from within the Jade Platform development environment or a Jade utility.
- Press F1 in the Jade Platform development environment editor pane when the caret is positioned on a Jade instruction, a primitive type, a system or Window class, property, method, event, or method option.
- Click the **Help** button on a dialog from within the Jade Platform development environment or a Jade utility.
- Click the **Context Help** toolbar button from within the Jade Platform development environment or a Jade utility and when the cursor changes to include a question mark (**?**) symbol, click on the window for which you require help.
- Double-click on a PDF file (for example, from Windows Explorer).
- Select the **Open** command from the File menu of Adobe Reader (or Adobe Acrobat) or the appropriate action in the PDF reader that you are using.
- Select the **Jade Help** item in the **Jade** program folder that is displayed when you click on the Start menu (at the lower left corner of your monitor) and then select **Programs**.

  The **Jade.pdf** document, which provides access to the Jade Platform product information library, is then displayed.

Getting Adobe Reader

Although Jade Platform documentation PDF files are compatible with Adobe Acrobat Reader versions 7.0 and higher, we recommend the latest version of Adobe Reader as it offers advanced functionality and may reduce printing problems in some situations. For example, the latest version of Adobe Reader automatically provides you with the ability to search over all documents installed during the upgrade or installation process, but some versions earlier than Adobe Reader 6.0 provided this functionality only if you specifically selected the download with the **Search and Accessibility** option, and Acrobat Reader 4.0 did not provide this functionality.

Download and install the free Adobe Reader from the Adobe website.

Notes If you have installed the entire Jade Platform product information library in print (PDF) format, a search index over all PDF documents is provided. The Acrobat search index functionality is available only if you are using Adobe Acrobat, the recommended latest full version of Adobe Reader (the default version), Adobe Reader 7.0 or higher, or version 5.0 with the **Search and Accessibility** option.

If you are using the Adobe Reader and you click a hyperlink in any Jade PDF document or you access the PDF documentation from Jade, the correct page is displayed but the specific topic to which you navigated may not be visible on your monitor. Scroll up or down a little until the topic that you require is visible. In addition, if you click a hyperlink that targets a topic on the same page, nothing happens. Scroll up or down a little until the topic that you require is visible.

When you scroll through a document, you can show your current location in the **Bookmarks** tab of the Navigation pane by clicking the **Expand Current Bookmark** toolbar icon or by selecting the **Expand Current Bookmark** command from the menu that is displayed when you click on the **Options** menu in the Bookmarks palette at the top of the Navigation pane at the left of the Adobe Reader window.

Obtaining Assistance when Using Adobe Reader

When you are running Jade and you are using the Adobe Reader, you can use the **Search** command from the Edit menu to search for any word, partial word, or combination of words in the current PDF document (and in all PDF documents). For details, perform one of the following actions from the Adobe Reader Help menu.

- Select the **How To** command and select the **General Topics** command from the submenu that is then displayed.

  The General Task Topics pane is then displayed at the right of the Adobe Reader window, enabling you to select the topic for which you require assistance.

- Select the **Adobe Reader Help** command in the Help menu or press F1.

  The *Adobe Reader Help* guide that is then displayed provides extensive assistance in using Adobe PDF files; for example, extensive explanations about the Adobe Reader tools, commands, and keyboard shortcuts.

  Note If you did not download the full Adobe Reader, you cannot access the complete Adobe Reader help system. However, the *Adobe Reader Help* guide provides information that enables you to do so.

Searching the Product Information Library using Adobe Reader

Using the Adobe Reader, you can search the Jade Platform product information by using the Adobe PDF index file (**Jadeindex.pdx**) or you can search for all text in the documents.

Tip Searching by using the Jade PDF index file is much faster than using the standard Adobe Reader search functionality to search for all text in the documents.

You can search the Adobe Jade PDF index file only when you have downloaded the recommended latest full version of Adobe Reader (the default version), Adobe Reader 7.0 or higher, or version 5.0 with the **Search and Accessibility** option selected and you have downloaded all documentation (which includes the **Jadeindex.pdx** file and subdirectories).

Note The **Jadeindex.pdx** file and the **Jadeindex** directory and its subdirectories must be located in the same directory as your Jade Platform documentation PDF files.

Using a PDF index with the Adobe Reader **Search** command, you can make full text searches of PDF documents and collections of PDF documents that have been indexed beforehand. The **Search** command enables you to:

- Search for a single word or short phrase
- Extend your search criteria by using wild-card characters and Boolean (**AND**, **OR**, **NOT**) operators
- Specify rules of proximity and word variants
- Use search options to refine your searches; for example, search for words that contain part of your search text, match cases, find similar words, and limit the proximity of pairs of words

Search results are presented ranked in order of relevance, and you can browse from result to result in your documents.

For details about performing basic searches and using advanced search criteria in Adobe Reader, see "Searching Adobe PDF Documents" in the *Adobe Reader Help* guide for details.

Selecting Indexes

The Reader development-related Jade PDF files are associated with the **Jadeindex.pdx** PDF index file, created to enable you to define queries for rapid searches of the Jade Platform product information library.

Note You will normally not have to find or select this index file yourself. Opening a PDF file associated with an index automatically makes the index searchable when you select **Use Advanced Search Options** at the bottom of the Search PDF pane.

However, when you first use the product library, you must add the **Jadeindex.pdx** index to the PDF indexes over which the advanced search facility operates.

To add the index to the indexes currently available for your Acrobat searches

1. Perform one of the following actions.
   - Select the **Search** command from the Edit menu.
   - Click the **Search** toolbar button.
   - Press Ctrl+F.

   The Search PDF pane is then displayed at the right of the Adobe Reader window.
2. Select **Use Advanced Search Options** at the bottom of the Search PDF pane.
3. In the **Look In** list box, select the **Select Index…** item. The Index selection dialog is then displayed.

4. To add the index:

   a. Click the **Add** button.

   b. In the standard Open Index dialog, navigate to the directory into which the product information library was installed.

   c. Double-click on the PDX file (that is, **Jadeindex.pdx**) or select the index and then click the **Open** button to select the index and close the dialog.

5. In the Index selection dialog, select the index, check the check box at the left of the selected index name (that is, **Jadeindex**) and then click the **OK** button.

   To view information about an available index, in the Index selection dialog, highlight the index name, click the **Info**, button, and then click the **OK** button. Information about the selected index is then displayed (for example, the date the index was created, the number of documents in the index, and the location and status of the index).

The item **Currently Selected Indexes** is then displayed in the **Look In** list box in the Search PDF pane. For more details, see "Searching Adobe PDF index files", in the *Adobe Reader Help* guide.

Printing PDF Documents

If you have trouble printing PDF files, particularly large PDF files with a high proportion of graphics content, you might consider some of the following options. If the problem is caused by resource memory, try working around the problem by printing your PDF document in shorter, successive sections by using the **Pages from**, **to**, controls of the Print dialog.

Adobe suggests the following fixes: in the Adobe Reader Print dialog, select the **Save Printer Memory**, or **Print as image** options (note that the latter may cause slow printing). Alternatively, change the print resolution to something lower; for example, 300 dpi.

Keep at least 50 percent of system resources free for printing and make sure that plenty of disk space is available (three to five times the size of the file that you are printing is recommended). You may also need to install more printer memory. If you are printing a PDF document to a non-postscript printer, Adobe recommends at least 2M bytes of memory for 300 dots per inch (dpi) printing and 4M bytes to 6M bytes for 600 dpi printing.

If you have Internet access, click on the **Printing Tips** button of the Adobe Reader Print dialog to access a troubleshooting page at the Adobe website. (You can also locate this page by searching the Adobe website.) You cannot make changes to a PDF file opened in Acrobat Reader but you can copy formatted text and paste it into other applications.

Obtaining Help in a Window or Dialog

» To access help in a window or dialog, perform one of the following actions

- Press F1 when the caret is positioned in a window or control (for example, in the Methods List of the Class Browser or a dialog text box).
- Click the **Context Help** toolbar button. When the cursor changes to include a question mark (?) symbol, click on the window for which you require help.
- Click the **Help** button in the current dialog.

Depending on the value of the **UseJadeWebHelp** parameter in the [JadeHelp] section of the Jade initialization file, help for the current window or control is then displayed, with the appropriate topic on the web page in your browser or in the top left corner of a PDF file in Adobe Reader document pane.

Obtaining Help in the Editor Pane

Depending on the value of the **UseJadeWebHelp** parameter in the [JadeHelp] section of the Jade initialization file, help from the editor pane accesses an HTML5 topic on a web page in your browser or in Adobe Reader at a specific section in a PDF file. (By default, context-sensitive help is in web format.)

Tip If you press F1 from the Jade Platform development environment to launch the Adobe Reader and access the appropriate topic in a Jade PDF file using Adobe Reader 7.0 or higher from a slower machine (for example, from an older laptop), Jade can time-out before it has time to load the appropriate document at the required topic after it has launched the Adobe Reader application. Although the Adobe Reader may be launched, you have to repeat the F1 action to enable Jade to open the PDF document itself at the specified hyperlink topic.

You can use the **ConnectSleepRetries** parameter in the [JadeHelp] section of the Jade initialization file to specify the number of seconds between the specified number of times Jade attempts to retry launching the Adobe Reader and opening a PDF file at the appropriate topic when you press F1 to access context-sensitive help from the Jade Platform development environment. For more details, see "Jade Help Section [JadeHelp]", in your *Jade Initialization File Reference*.

To access online help in the editor pane

- Press F1 when the caret is positioned on a:
  - Jade instruction
  - System or Window class, property, method, or event
  - Primitive type
  - Method option

Online help for the current item is then displayed, with the appropriate topic on the web page in your browser or in the PDF file displayed in the top left corner of the Adobe Reader document pane.

The help text provides the appropriate description, syntax or signature, and often a usage example.

To find details about the item under the caret

- Press F11

Details about the current item, including the creation timestamp and patch version of the entity, are then displayed in bubble help. For example, if the caret is positioned on the **i** variable, bubble help displays the following information.

```
Class: TestSchema:Person (2051)
Superclass: Object
Access: public
Type: real
Lifetime: all all-subclasses
Volatility: Volatile
Default: persistent
Maps: testschema
Created: 01 January 2019, 13:00:00
Patch Version: 1024
```

When details of a property that has a mapping method are displayed, press F12 to display a freestanding editor pane containing the mapping method source for that property.

See also "Using Bubble Help in the Editor Pane", in Chapter 4 under "Defining and Compiling Jade Methods and Conditions", "Displaying Bubble Help in Browser Lists" under "Using the Class, Primitive Types, or Interface Browser", in Chapter 3, and "Navigating around the Jade Platform Development Environment", earlier in this chapter.

» To view information about the cause and solution to a compiler error

- Press F1 when a compile error (that is, an error in the range 6000 through 6999) is displayed in Jade Platform development environment status line and the caret is positioned in whitespace in the editor pane (that is, there is no character on either side of the caret).

Online help for the error is then displayed on the web page in your browser or in the PDF file, providing you with the cause and recommended action for the solution of that compiler error.

Note This feature applies only to compiler errors. Syntax errors are displayed only in the status line.

Obtaining General Help

» To access Jade online help in general, perform one of the following actions

- Select the **Index** command in the Help menu
- Click the **General Help** toolbar button

Depending on the value of the **UseJadeWebHelp** parameter in the [JadeHelp] section of the Jade initialization file, the default page of the product information (the first page of the Jade online help directory document) is then displayed on a web page in your browser or the directory document (**Jade.pdf**) is then displayed in Adobe Reader, providing a summary of and hyperlinks to all documents in the Jade Platform product information library.

Use the functions available in Jade online help to find the required topics. For details, see "Jade HTML5 Online Help" or "Jade Platform Product Information Library in Portable Document Format", elsewhere in this chapter.

Obtaining Jade Version Information

Use the Jade Platform development environment Help menu **About Jade** command to access information about the Jade Platform.

» To access the Jade information

- Select the **About Jade** command from the Help menu

The About box is then displayed. This dialog is for display purposes only.

Creating Context Links to Your Own Application Help File

Creating context links from your Jade application to your own help file is simple. The easiest way to do this is by making use of named destinations that you create in your Adobe Portable Document Format (PDF) help document or index entries in your Microsoft Word file, or to a URL.

Notes Jade supports web-based Hypertext Markup Language (**.htm** or **.html**) files, Adobe Portable Document Format (.**pdf**) files, Windows help (.**hlp**) files, and compiled help (.**chm**) in your Jade applications.

If you supply online help for applications that you develop, it is your responsibility to provide the supporting software that enables the display of the appropriate format.

For details, see "Web-based HyperText markup Language (HTML) Online Help", "Adobe Portable Document Format (PDF) Online Help", or "Creating Context Links for a Windows Help File", later in this section.

Web-based HyperText Markup Language (HTML) Online Help

When help is invoked directly for web-based online help via the **Window** class **showHelp** method or via the user pressing the help key (F1), a URL is created and the default browser is invoked to display the URL.

When help is requested, if the **Application** class **helpFile** property specifies web HTML help, detected by the value of the **helpFile** property starting with a recognized URL scheme (that is, **http://** or **https://**), Jade attempts to construct a URL to pass to the default web browser.

Set the value of the **Application** class **helpFile** property to a base URL, as shown in the following code fragment examples.

```
app.helpFile := "http://www.example.com/prodhelp";

app.helpFile := "http://www.example.com/prodhelp/" & app.version.String & "/";
```

If the value of the **Window** class **helpKeyword** property starts with a recognized URL scheme, the URL is used; otherwise the value of the **helpKeyword** property is appended to the value of the **helpFile**property to become the URL to use, as shown in the following code fragment example.

```
mybtn.helpKeyword := "Form1/button1.htm";
mybtn.showHelp;
```

The [JadeHelp] section of the Jade initialization file can contain the following parameters, to enable you to configure the online help that you require for your Jade system. (For details about the other Jade initialization file parameters that enable you to configure your Jade help files, see "Jade Help Section [JadeHelp]", in your *Jade Initialization File Reference*.)

- **HelpSchemes**, which has the default value of **http://,https://**, enables you to specify other recognized URL schemes; for example, adding **file://** (that is, **http://,https://,file://**) enables you to load HTML help pages from your local disk.
- **HtmlHelpIndexUrl**, which has the default value of null, or blank. If this parameter is present, it must be a complete URL. If present and the **Index** command in the Help menu has been selected, help context, or help finder has been requested, this value will be used; for example:

  ```
  HtmlHelpIndexUrl = http://www.example.com/prodhelp/index.htm
  ```

- **HtmlHelpContentsUrl**, which has the default value of null, or blank. If this parameter is present, it must be a complete URL. If present, help has been requested, and no **helpKeyword** property value can be found, this value will be used; for example:

  ```
  HtmlHelpContentsUrl = http://www.example.com/prodhelp/contents.htm
  ```

- When handling automatic Help menu items, if a **helpContextId** or **helpKeyword** property is specified on the Help menu **Index** automatic menu item, the destination of the help is based on the value of the **helpContextId** or **helpKeyword** property. In addition, the **click** event method is not executed.

If you do not use the **Application** class **helpFile** property to dynamically set the help file at run time, no help file is opened when help is requested, regardless of the value specified in the **Window** class **helpKeyword** property.

Adobe Portable Document Format (PDF) Online Help

When help is requested, if the help file specifies a Portable Document Format (PDF) file (detected by the .**pdf** file suffix), Jade attempts to execute Adobe Reader to handle the file. Jade checks the Windows registry for the Acrobat Reader (**AcroRd32**) or for the **acrobat** executable program.

If Adobe Reader is not found, the help request is ignored and entries explaining the cause of the failure are output to the **jommsg.log** file. If Adobe Reader is located, it is initiated for the PDF help file defined in Jade. See also "Using Bookmarks for Navigation around a PDF File", later in this chapter.

For a **helpKeyword** help request, the **helpKeyword** property is passed to Adobe Reader as a **named destination**, which Adobe Reader uses to position the help file display. As there are no equivalent concepts in a PDF file of any other type of help request (for example, **helpContextId**, index request, and so on), only the first page of the PDF file is displayed for a help request other than using the **helpKeyword** property.

When preparing your documents for PDF online help, note the following points.

- The **Window** class **helpKeyword** property can contain a help file name before the keyword, separated by a semicolon. This help file (which can be a .**pdf**, .**hlp**, or .**chm** file) is specific to this **helpKeyword** property, and overrides the default value; for example:

```
btnHelp_click(btn: Button input) updating;
vars
begin
    if fldFolder.topSheet = shtSelect then
        btn.helpKeyword := "DevRef.pdf;selectinglibraryacxautomationdrg10";
    elseif fldFolder.topSheet =  shtLibrary then
        btn.helpKeyword := "DevRef.pdf;namelibrary_activex";
    elseif fldFolder.topSheet =  shtObjects then
        btn.helpKeyword :=
"DevRef.pdf;namingobjectclassesacxautomationdrg10";
    elseif fldFolder.topSheet =  shtInterfaces then
        btn.helpKeyword := "DevRef.pdf;naminginterfacesacxautomationdrg10";
    elseif fldFolder.topSheet =  shtConstants then
        btn.helpKeyword := "DevRef.pdf;namingconstantsacxautomationdrg10";
    endif;
    btn.showHelp;
end;
```

  Note Although it is more efficient to use a single help file, specified in the **Help File** text box on the **Application** sheet of the Define Application dialog, this feature is intended for situations in which multiple help files are required for a single application.

  If you do not specify a help file in this text box or you do not use the **Application** class **helpFile** property to dynamically set the help file at run time, no help file is opened when help is requested, regardless of the value specified in the **helpKeyword** property.

- When handling automatic Help menu items, if a **helpContextId** or **helpKeyword** property is specified on the Help menu **Index** automatic menu item, the destination of the help is based on the value of the **helpContextId** or **helpKeyword** property. In addition, the **click** event method is not executed.

For details about the Jade initialization file parameters that enable you to configure your Jade help files, see "Jade Help Section [JadeHelp]", in your *Jade Initialization File Reference*.

Acrobat does not convert index fields in your source documents to named destinations. You can manually create named destinations to correspond to topics in your source file. (For details, see the Adobe Reader online help.) Alternatively, you can write a macro to generate named destinations from headings, if required. Named destinations are case-insensitive, must be unique within the document, and cannot contain spaces.

Caution As named destinations are **Print** fields, an error or unexpected results can occur if you output a document containing these fields to a printer. You may therefore want to create a copy of your source file to meet you print requirements.

Alternatively, create the PDF file from your source document that contains the **named destination** fields and then print the PDF file.

To create a context link for a PDF help file

1. In your documentation for a specific topic (for example, for a **Products** list box control on a form that has a topic (a subsection) in your documentation for that control called "Products List Box"), insert a unique **named destination** field, which automatically becomes a help keyword when you generate your PDF help file. The example used in this procedure has a unique **saleproduct** named destination field.

   To manually add a **named destination** field if you have not written a macro to generate named destinations from headings, perform the following actions.

   a. Press Ctrl+F9.

      A pair of braces symbols **{ }** is then displayed.

   b. Inside the brace symbols, enter the following, where ***named-destination*** is the unique name for your **named destination** field.

      ```
      PRINT "[ /Dest /named-destination /DEST pdfmark"
      ```

      Notes This *is* the correct syntax; there is not a missing closing bracket (]) symbol.

      The named destination fails if there is a space after any virgule character (/) or the name of the named destination contains a space.

   You can add a **named destination** to any part of your Word source document.

   Unlike the insertion of a bookmark for subsequent creation of Windows online help when you create a source Word document for generation as a PDF help file, the named destination can be inserted in any part of the document, regardless of the formatting style applied to the text at the position you insert the named destination field.

2. Turn on the display of fields in your Word document (by checking the **Field codes** check box on the **View** sheet of the Options dialog, accessed from the **Options** command in the Tools menu). All of your named destination help fields are then displayed.

3. In your Jade Platform development environment, apply the **named destination** keyword to your list box object. In this case, you would:

   a. Edit the **Products** form in Jade Painter.

   b. On the **Common** sheet of the Properties dialog for the **Products** list box control, specify **saleproduct** in the text box for the **helpKeyword** property.

      Tip Although you can type the value into the **helpKeyword** property on the **Common** sheet of the Properties dialog, you can also copy only the portion of the field in the Word document between the **Dest /** and **/DEST** (that is, only the **sales product** part of the **PRINT "[/Dest /salesproduct /DEST pdfmark"** field) and paste it into the **helpKeyword** property.

Alternatively, if you cannot edit a control in Painter (for example, when the sheets of a folder control are overlaid at run time so you cannot apply help keywords to sheets in Painter), apply your keyword in your Jade code, usually in the **load** method for the form, as shown in the following example.

```
if products.topSheet = supplier then
    supplier.helpKeyword := "salesproduct";
elseif ... then
```

4. Apply help keywords in your Jade code when you have multiple sheet folders, as you do not normally want your **Help** button to display text relating to the whole dialog but only for the sheet that is on view.

   In this case, identify the sheet that is on top and then apply the keyword using code in the **click** event for the **Help** button, like that shown in the following example:

```
if <form-name>.topSheet = <sheet1-name> then
     <sheet1-name>.helpKeyword := "<named-destination-1>";
elseif <form-name>.topSheet = <sheet2-name> then
     <sheet2-name>.helpKeyword := "<named-destination-2>";
...
```

5. To make your menu commands context-sensitive (or "hot"), apply your named destination help keyword by using the Menu Design facility. Specify the help keyword for your menu command index entry in the **Help Id** text box on the **Text** sheet of the Menu Design dialog.

   Note Index entries must be unique. For example, if several of your forms have identical controls, you must further qualify each one; for example:

```
Print Option Button - invoice
Print Option Button - receipt
Print Option Button - address
```

As you cannot define the F1 key as a shortcut key on a menu item (Painter does not allow you, because Windows treats F1 key messages differently), you must implement this functionality with a **keyDown** event method on the top-level **Form** class in the application. Ensure that the key code is set to **null** before leaving this method so that Jade's default help processing is not actioned.

In addition, you should use the **registerKeys** method to specify that you are interested only in the F1 key so that the presence of the **keyDown** event method does not have too much impact on remote thin client users.

The PDF file must be located in the help binary directory and the exact name of the PDF file must be specified in the **Help File** text box on the **Application** sheet of the Define Application dialog for the application. However, this specification of the PDF file (for example, **OurSalesApp.pdf**) must be set up on the Define Application dialog and the file located in the binary directory before the appropriate command (for example, the **User's Guide** command on the Help menu in the application itself will launch the PDF file).

Using Bookmarks for Navigation around a PDF File

Use bookmarks to create your "go to" points (hotspots) for jump references used for navigation within PDF files.

When inserting bookmarks, note the following points.

- You can add bookmarks only to headings; that is, to a paragraph that is associated with a heading style (for example, **Heading 1**).
- Bookmarks are case-insensitive.
- Create a bookmark at a heading within your Microsoft Word document. You should append each bookmark with a unique document reference and chapter number (for example, **funtionkeysUG2**).

As the maximum number of characters for bookmarks is small, avoid underscore characters and other symbols. Simply ensure that the bookmark is unique within the document, is sufficient to make it clear the heading on which it is based, and that each bookmark has the appended document and section details.

- As the bookmark name cannot contain spaces, if you want to create a bookmark that reflects a multiple-word heading, string the words together into a single bookmark name; for example, **copycommandug3** (for the **Copy** command in Chapter 3 of a user's guide).
- If you have inserted a page break before a heading that is to be a bookmark for a topic jump (by using the **Break** command from the Insert menu), remove this page break and replace it by checking the **Page break before** check box on the **Line and Page Breaks** sheet of the Paragraph dialog.

  If you do not do this, when you print a document or reopen a saved document, a page break is inserted before *each* reference to that bookmark, which looks horrifying (and can take a long time to rectify).
- If you have a control with the same name in more than one dialog or a command with the same name on more than one menu, differentiate them by adding the name of that menu or dialog to your bookmark; for example, **copyeditbrowserug3** (for the **Copy** command from the browser Edit menu in Chapter 3 of a user's guide).

» To create bookmarks for "go to" points

1. In your Word source file, select the heading text to which you want to navigate.
2. Select the **Bookmark** command from the Insert menu. (Alternatively, press Ctrl+Shift+F5.) The Bookmark dialog is then displayed.
3. In the **Bookmark name** text box, specify a bookmark name that is unique within the document, and then click the **Add** button.

   If you want to immediately create a reference field to that bookmark from elsewhere in the document, copy the text to the clipboard (Ctrl+C when it is selected) in the **Bookmark name** text box before you click the **Add** button.

Caution It is preferable to display the bracket symbols (**[ ]**) that Word creates around a bookmark, by checking the **Bookmarks** check box on the **View** sheet of the Options dialog, accessed from the Tools menu.

Take care each time you move text to a position in front of a bookmark. If the bookmark brackets are not visible and you place your caret to the left of the text of your bookmark, the caret is actually positioned *inside* the bookmark. Unexpected (and unwanted) results will occur if you make an insertion at this point.

If links are required from other documents in the product information library, it is simpler to insert a hyperlink instead of a reference field that must subsequently be converted to a hyperlink during the pre-production phase of the document.

» To create references to bookmarked headings

To access a bookmarked help heading from within a PDF file, you must insert a reference field to that bookmark by performing the following actions in your Word source file.

1. Select the word or words in your text that you require as the "jump from" point. This is most often a reference to the heading to which you want to jump.
2. Select the **Field** command from the Insert menu.
3. In the **Field codes** text box, delete the equal sign (**=**) and then specify the following.

```
ref <bookmark-name>
```

Tip Rather than trying to remember the exact bookmark name, you may find it more convenient to split your document (by using the **Split** command from the Window menu), searching for the bookmarked heading that you want to reference, accessing the Bookmark dialog from the Insert menu for that heading, and then copying the bookmark to the clipboard so that you can paste it after the space that follows the word **ref** in the **Field codes** text box in the **Field** dialog.

4. Click the **OK** button to insert the bookmark.

Note To ensure that you can easily determine the location of reference fields within your Word document, select the **Always** value in the **Field shading** combo box on the **View** sheet of the Options dialog.

The field is then displayed with a gray-scale background.

5. Select the field and then apply a color of your choice (for example, the bright blue color) to the reference field by selecting the required color (for example, the **Blue** color) from the **Font Color** icon on the Formatting toolbar or by selecting the reference field and then selecting the appropriate colored square (for example, the bright blue square) displayed in the extended **Font** color drop-down list box on the Font dialog.

Creating Context Links for a Windows Help File

Index entries in Word documents automatically become help keywords in a Windows help file, and it is then simply a matter of applying those keywords as properties in Jade. For example, for a list box control on a form called **Products** that has a topic (a subsection) in your documentation for that control called "Products List Box", perform the following actions.

1. In your documentation for a specific topic, apply an index entry; for example, "Product List - displaying", which automatically becomes a help keyword when you generate your Windows help file.

2. In your Jade Platform development environment, apply the keyword to your list box object. In this case, you would:

   a. Edit the **Products** form in Jade Painter.

   b. On the **Common** sheet of the Properties dialog for the **Products** list box control, specify **Product List - displaying** in the text box for the **helpKeyword** property, exactly the same as the index entry you inserted in your documentation for that topic.

   Alternatively, if you cannot edit a control in Painter (for example, when the sheets of a folder control are overlaid at run time so you cannot apply help keywords to sheets in Painter), apply your keyword in your Jade code, usually in the **load** method for the form, as shown in the following example.

```
if products.topSheet = supplier then
    supplier.helpKeyword := "Product List - displaying";
elseif ... then
```

3. Apply help keywords in your Jade code when you have multiple sheet folders, as you do not normally want your **Help** button to display text relating to the whole dialog but only for the sheet that is on view. In this case, identify the sheet that is on top and then apply the keyword using code in the **click** event for the **Help** button, like that shown in the following example:

```
if <form-name>.topSheet = <sheet1-name> then
     <sheet1-name>.helpKeyword := "<index-entry-1>";
elseif <form-name>.topSheet = <sheet2-name> then
     <sheet2-name>.helpKeyword := "<index-entry-2>";
```

4. To make your menu commands context-sensitive (or "hot"), apply your index entry help keyword by using the Menu Design facility. Specify the help keyword for your menu command index entry in the **Help Id** text box on the **Text** sheet of the Menu Design dialog.

Note Index entries must be unique. For example, if several of your forms have identical controls, you must further qualify each one; for example:

```
Print Option Button - invoice
Print Option Button - receipt
Print Option Button – address
```

Chapter 3 Defining Classes in Your Own Schema

This chapter covers the following topics.

- Defining Your Own Schema
  - Defining a Schema
  - Accessing a Schema
  - Validating Schema Definitions
  - System Map Files
- Defining Applications
  - Setting an Application
  - What Happens Next
- Using the Class, Primitive Types, or Interface Browser
  - Opening a Class, Primitive Types, or Interface Browser
  - Using the Class List, Primitive Type List, or Interface List Window
  - Using the Properties List Window
  - Using the Methods List Window
  - Using the Editor Pane
- Adding Classes to Your Schema
  - Finding a Schema, Class, Interface, or Primitive Type
  - Displaying All Superschema Classes in Your Class Browser
- Defining Your Own Classes
- Customizing the Class Browser of the Current Schema
- Defining Dynamic Clusters
- Browsing Classes that Are in Use
- Displaying Process Usages of a Class
- Compiling All Methods Defined in a Class, Primitive Type, or Interface
- Searching for or Replacing an Element in a Schema
- Specifying Text for a Schema Element

- Printing a Selected Schema Element
- Performing a Reorganization

Defining Your Own Schema

The Schema Browser is displayed by default when you first sign on to the Jade Platform development environment.

A *schema* is the highest-level organizational structure in Jade, and represents the object model for a particular domain. A schema is a logical grouping of classes, together with their associated methods and properties. These effectively define the object model upon which your applications are based.

For details about importing packages of common frameworks, see Chapter 8 of your *Developer's Reference*, "Using Packages".

The appearance and functionality of applications in a schema can differ, but they all share the underlying object model defined by the schema. The **RootSchema** is provided by Jade, and is always at the top of the schema hierarchy.

The **RootSchema** provides essential system classes; for example, the **Object** class (the root of the class hierarchy), **Collection** classes, and the **File**, **Exception**, and **Form** classes. Because these classes are defined in the **RootSchema**, they can be accessed from all subschemas.

When you begin developing a system using Jade, you would normally begin by adding a schema to the **RootSchema** and you would then define all of your classes, properties, and methods within that schema. This ensures that your object model (class hierarchy) is clearly packaged and is kept distinct from the system classes.

For larger and more-complex development efforts, a hierarchy of user schemas may be necessary to adequately represent the object model. You can add a subschema to the **RootSchema**. You can also add new classes to a schema, methods, properties, and constants to a schema class, or methods and constants to an existing class defined in a superschema.

In addition, you can define an *interface*, which provides a set of methods that are guaranteed to be available on any implementing class. When a class implements an interface, it agrees to implement all of the methods defined by the interface. With this contract in place, the implementing classes can participate in useful type-safe callback mechanisms.

Classes that implement an interface can be grouped by that interface type (for example, a collection can have a membership of a specified interface), to capture the similarities of non-related classes without the need to artificially force a class relationship; that is, you allow a class to perform multiple roles outside of the role dictated by its class hierarchy. For details, see Chapter 14, "Adding and Maintaining Interfaces".

For details about grouping methods from any class in any schema into a named workspace, see Chapter 13, "Using Method Views to Bookmark Workflows".

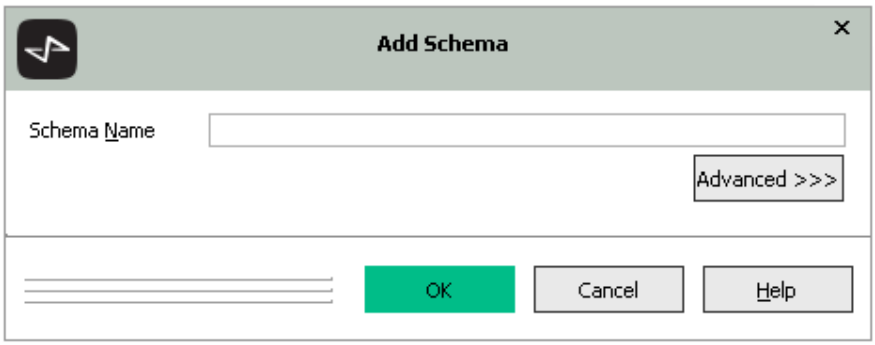
Defining a Schema

When you are developing a Jade system or you have installed Jade for the first time, you must first add your own schema to the system-supplied **RootSchema**.

» To add a schema

1. In the Schema Browser, select the **RootSchema**.
2. Select the **Add** command from the Schema menu. (You can add a schema only when the **Global** class in any superschema does not require reorganization.)

The initial form of the Add Schema dialog, shown in the following image, is then displayed.

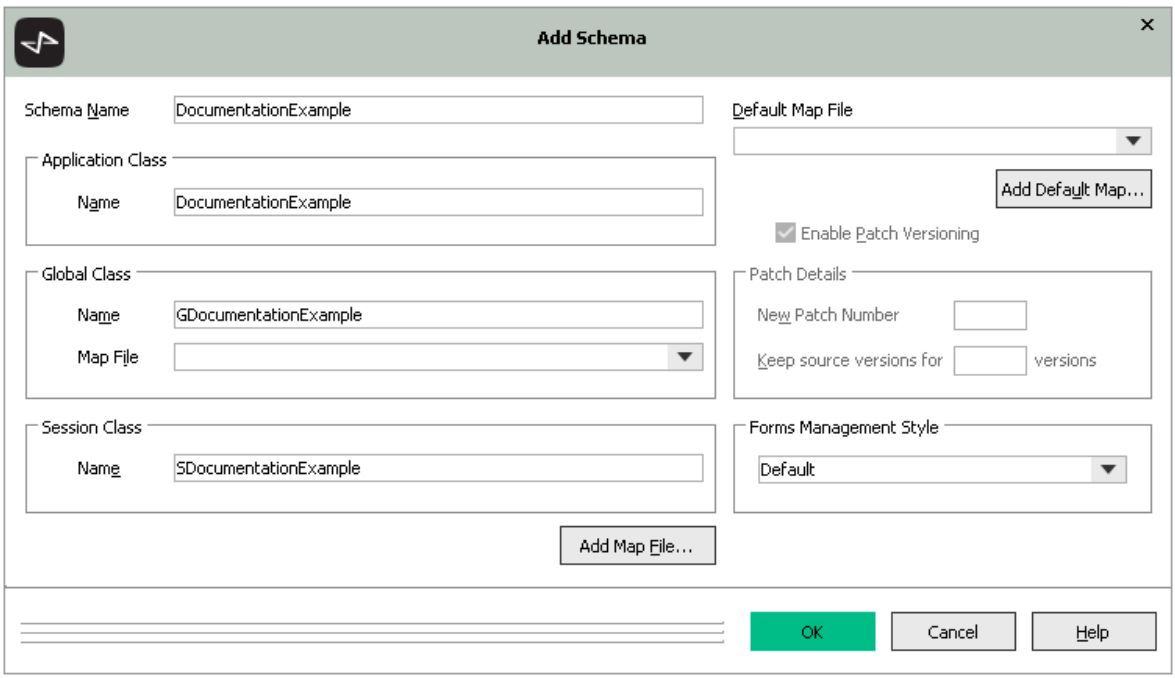


3. In the **Schema Name** text box, specify the name of the schema that you want to define. Your schema name must start with a letter, it must be unique, and can contain alphanumeric characters only. (Jade automatically converts the first letter to uppercase, if required.)

   If you do *not* want to change any default values, go to step 15 of this instruction.

4. If you want to change default values, click the **Advanced** button.

   The extended Add Schema dialog, shown in the following image, is then displayed.



The logical entity representing a database file, sometimes referred to as a *map* file in Jade, provides a convenient way of separating the user's view of data from some of the practical considerations associated with storing that data on disk. A schema database has its own set of files and class maps that are used when any of the applications of that schema is executed. A subschema database can reuse or override the file definitions of its superschema.

The default map file specifies the name of the database file that is used when reading or writing instances of a class for which no map file has been defined. The default map file that is created is based on the schema name specified in step 3 of this instruction.

The map file for the global instance is the default map file.

You can change map files (that is, the path in which your class map file is located) only by using the batch Jade Database utility. For details, see the **setFilePath**, **setFilePathAudited**, **clearFilePath**, and **clearFilePathAudited** commands, in Chapter 1 of the *Database Administration Guide*.

5. If you want a different default map file (for example, if you are adding a subschema to an existing user-defined schema) select an existing map file from the list box in the **Default Map File** combo box.

6. If you want to define a new map file that you can then select as the default map file in the **Default Map File** combo box, click the **Add Default Map** button.

   Note You cannot select a map file that is marked as partitioned and that already has a class map defined for it.

   The File Dialog is then displayed.

   a. In the **File Name** text box, specify a valid file name prefix for the default class map file; for example, **defmap**.

      The map file name cannot contain the **- \ / : * ? " < > | .** characters or spaces.

   b. Check the **Partitionable** check box if the database map file can be partitioned. (You cannot change the value of the **Partitionable** check box if the **DbFile** class **isPartitioned** method returns **true** or the file is used elsewhere.)

      A partitioned database file must have zero or one classes mapped to the file. A collection class cannot be mapped to a partitioned file.

      Note Because the default map file cannot be partitioned, this check box is disabled when you specify the default map file for a new schema.

   c. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

   d. Return to step 5 of this instruction if you want to select your specified file as the default map file. (See also "System Map Files", later in this chapter.)

7. The application class name defaults to the name of the schema. If you want to change the default application class name, specify the name in the **Name** text box in the Application Class group box.

   A schema can contain one or more applications, each of which is an instance of the **Application** class. A persistent instance of each application is created at development time. A transient instance (an instance of the application class that you define) is created each time the application is run. Jade automatically creates a default application for you, which has the same name as your schema.

   Persistent instances of the **Application** class are mapped to **_usergui.dat**. You cannot change this mapping, which identifies the database file in which persistent instances of the application are stored.

8. The global class defaults to the name of the schema prefixed with **G**, but you can change it if you want, by specifying that you require for your global class in the **Name** text box in the Global Class group box.

   Jade automatically creates a single persistent instance of your global class. There can be one instance of this class only. All applications in the schema have a reference to this single global instance (by using the **global** system variable).

   Note You cannot select a map file that is marked as partitioned and that already has a class map defined for it.

9. In the Global Class group box, select your map file for the global class in the **Map File** combo box if you want to change the map file for the global class to an existing map file name.

   Alternatively, you can add a new map file by using the **Add Map File** button. (For details about using the File Dialog that is then displayed, see step 6 of this instruction.)

10. The session class defaults to the name of the schema prefixed with **S**, but you can change it by specifying the name of your session class in the **Name** text box in the Session Class group box. This session class is created as a subclass of the **WebSession** class.

    For web-enabled applications, a transient instance of the web session class is created for each web session when a web-enabled application is run. Your logic can access this transient instance by using the **currentSession** system variable.

11. If global patch versioning is set for your database, check the **Enable Patch Versioning** check box if you want patch versioning applied to your new schema. This check box is disabled if global patch versioning has not been set for your database. For details, see "Enabling or Disabling Patch Versioning", in Chapter 2 of the *Runtime Application Guide*.

12. If patch versioning is enabled for your new schema (that is, the **Enable Patch Versioning** check box is checked), specify a new patch number in the **New Patch Number** text box if you want to assign a new patch version number for changes made by all developers in your new schema.

    By default, the new patch number is **1**.

13. In the **Keep source versions** text box, specify the number of changes within the current patch version number whose sources you want to keep if you do not want to keep the default number of changes (that is, 10).

    If the specified number of source changes is exceeded, only the latest changes up to your specified number are retained (for subsequent method source comparison, for example). If you anticipate that schema elements may be changed several times, consider setting this to a higher value. (No warning is raised if the number of changes within the patch version exceeds the specified number, so you may later find that the first source change is no longer retained.)

14. In the **Forms Management Style** combo box, select the forms management style that you require for your schema if you want a style other than the default multiple definitions and multiple translations style.

    The default style provides the maximum flexibility at the cost of the greatest maintenance effort.

    For more details, see "Forms Translation Styles", in Chapter 11.

15. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The specified schema is then displayed as a subschema of the selected superschema in the Schema Browser.

Note When you define a new schema, it automatically becomes the current, or selected, schema. The current schema and application are displayed at the right of the status line.

Removing a User-Defined Schema

The **Remove** command from the **Schema** menu enables you to remove (delete) a user-defined schema from the Schema Browser.

Note You can remove user-defined schemas only. You cannot remove a user-defined schema that has one or more subschemas or a system schema. (The **Remove** command is disabled if the schema selected in the Schema Browser has subschemas or it is a system schema.) As you cannot remove a schema that has subschemas, you must first remove any subschemas of that schema before you can delete the higher-level schema.

For details about removing elements from a user-defined schema (for example, classes, methods, or properties), see "Removing a Schema Element", later in this chapter.

» To remove a schema that has no subschemas

1. In the Schema Browser, select the schema that you want to remove.
2. Select the **Remove** command from the Schema menu.

   The Confirm Delete message box is then displayed.
3. To confirm that you want to remove the schema and all of its class instances, click the **OK** button. Alternatively, click the **Cancel** button to abandon the deletion.

The Schema Browser is then updated to reflect the removal of the schema. (There may be a momentary delay while this updating occurs.)

Caution If classes in the schema have destructors that delete dependent objects, errors may be raised if the schema deletion has already deleted those objects.

For details about deleting a user-defined schema from a deployment database using the optional **deleteSchema** parameter, see the *Schema Load User's Guide*.

Accessing a Schema

If you want to view or maintain a schema other than the current schema, in the Schema Browser you simply select the schema that you want to view. The schema that you select then becomes the *current* schema. (The schema of the window or browser that currently has focus is the *current* schema; for example, although the **LockTest** schema may be selected in the Schema Browser, the **TestSchema** schema is the current schema if the **TestSchema** Class Browser currently has focus.)

By default, existing open windows for the previous current schema are not closed, as Jade enables you to have concurrent open windows for different schemas in a development environment session. You can therefore view components of other schemas and work concurrently in two or more schemas in the same Jade database. For example, you can browse the classes, properties, or methods in an existing schema while you define a new schema.

The Schema Browser highlights an incomplete schema with a default background color of red. (You can use the **Window** sheet of the Preferences dialog to set this background color to a color of your choice.) If you select an incomplete schema in the Schema Browser, the Schema, Browse, and Jade menus are disabled.

If you have selected an incomplete schema and you then attempt to open a Class, Primitive Type, Map, or Application Browser by clicking on the relevant toolbar button, a message is displayed, advising you that the schema is incomplete.

Although you can have a browser of each type open for one or more schemas at any time in a work session, only one Schema Browser can be open for the whole database, as the schema itself is development environment-dependent.

In addition, you can have any number of the following types of browser open concurrently for each schema, if required:

- Class Browser
- Primitive Types Browser
- Interface Browser
- Methods Browser

Notes You can also have concurrent open Summary of Patches and Translator windows for different schemas in a development environment session. You can search for a specific user-defined schema if you have more schemas than are displayed in the Schema Browser. For details, see "Finding a Schema, Class, Interface, or Primitive Type", later in this chapter.

Use the Schema menu **Close Windows for Schema** command to close all open windows for the current schema; that is, the schema of the window that currently has focus.

To select a schema as the current schema

- In the Schema Browser, select the schema that you want to be the current schema. Alternatively, change focus to any open browser window of the required schema.

The selected schema is then the current schema, and is selected in the Schema Browser.

What Happens Next

A default application with the name of the schema but with no start-up form is created for you when your schema is defined. When you have created your first form for the application, it becomes the default start-up form when you run your application.

During development, you can nominate another form as the start-up form, by using the **Change** command from the Application menu in the Application Browser. You can also change the start-up form in your code, by setting the **startupForm** property of the **Application** class.

After you have defined your own schema:

- Define your own applications if you want to provide a different "view" of the object model, with different forms allowing the data in that model to be displayed, and possibly updated. For details, see "Defining Applications", later in this chapter.
- Alternatively, access the Class, Primitive Types, or Interface Browser or the Jade Painter to start defining your classes, interfaces, methods, properties, constants, and forms for your application if you want to specify the start-up form for the application at a later stage. For details, see "Using the Class, Primitive Types, or Interface Browser", later in this chapter, and "Creating a Form", in Chapter 5.

Validating Schema Definitions

You can validate the definition of the following elements in your current schema.

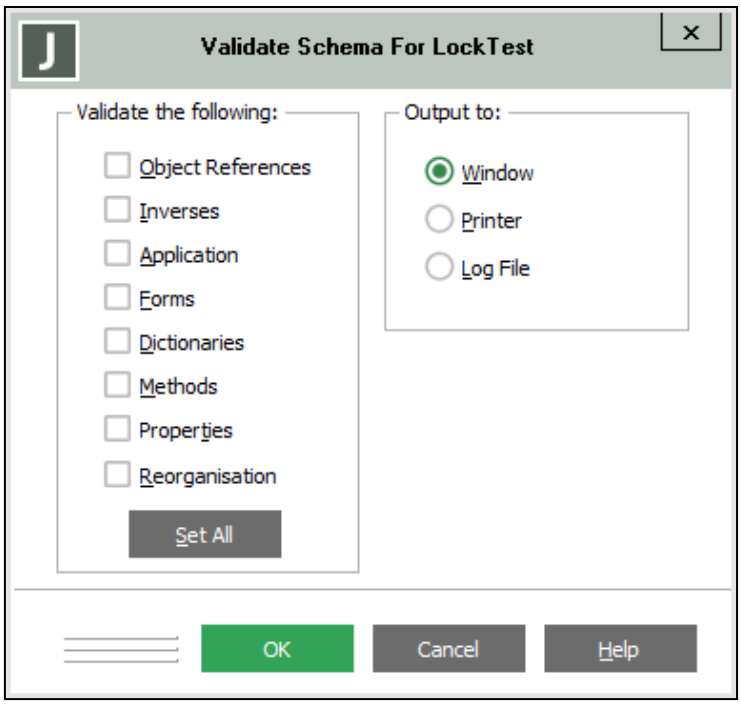
- Object references
- Inverse relationships
- Application
- Forms
- Dictionaries (except for dynamic dictionaries)
- Methods
- Properties
- Reorganization

To validate the current schema

1. Select the **Validate** command from the Schema menu for the current version of a versioned schema. (The **Validate** command is disabled for the latest schema version when the current schema in marked for reorganization.)

   As classes in the schema may require reorganization, you can validate these classes if the schema itself is not marked for reorganization.

   The Validate Schema dialog, shown in the following image, is then displayed.



2. Check the appropriate check box of each definition in the current schema that you want to validate, as follows.
   - Object References, to output a list of all invalid object references.
   - Inverses, to output a list of all invalid inverse reference definitions.
   - Application, to output a list of all invalid application definitions.
   - Forms, to output a list of all invalid form definitions and report on the following.
     - A control or menu has been deleted from a form but not all associated event methods of the control or menu have been deleted or it does not have an underscore character (_); that is, a form has an event method that does not have an associated property and it is not a **Form** event.
     - A control or menu has been deleted from a form but not all associated event methods of the control or menu have been deleted; that is, a form has an event method that does not have an associated property.
     - A control or menu has been deleted from a form but not all associated event methods of the control or menu have been deleted; that is, a form has an event method that does not have an associated property but a non-control or menu property also exists.
     - Form has invalid control parents.

- **Form** subclass has no corresponding form instance.
- Form instance has no class.
- Form control or menu property has no control or menu instance.
- Invalid form property of control or menu.

- Dictionaries, to output a list of all invalid dictionary definitions.
- Methods, to output a list of the class and name of all methods in the current schema that are in error or are uncompiled.
- Properties, to output a list of all properties in a class that have duplicate feature numbers.
- Reorganisation, to validate and report on the following.
  - If a schema has been marked as versioned but there are no classes versioned.
  - If there are versioned classes but the schema has not been marked as versioned.

  If you open the Validate Schema dialog from the latest version of a versioned schema, the **Reorganisation** check box is disabled.

3. If you want to validate all schema definitions, click the **Set All** button. All schema definitions are then checked for validation.
4. If you do not want your selected definition validations output to your monitor, select the **Printer** option button to output the results to your default printer or the **Log File** option button to output your selections to a log file.

   The log file, named **valscm.log**, is located in the binary (**bin**) directory in which the Jade executable file (**jade.exe**) is located.

   You can view the log file by using a text editor such as Notepad. The output file is not cumulative; that is, the file is overwritten the next time it is used.
5. Click the **OK** button to confirm your selection. Alternatively, click the **Cancel** button to abandon this selection.

The validation of all required schema definitions is then initiated. A progress dialog displays the selected definitions as they are validated. After a momentary delay, a message box is displayed, advising you that the validation has completed successfully and no errors were detected. If errors were detected during validation, the End of Validate dialog displays the schema or schemas that contain errors.

If your validation output was directed to the monitor (the default option), the Jade Interpreter Output Viewer window displays the validated definitions. For details about using the Jade Interpreter Output Viewer to view definition validations output to your monitor, see "Using the Jade Interpreter Output Viewer", in Chapter 1 of the *Runtime Application User's Guide*.

The following is an example of validation output to the Jade Interpreter Output Viewer window (which is the same as the information output to the default printer and to the **valscm.log** file).

```
=====================
Validating References
=====================
454 object references were validated.
There were no errors.

===================
Validating Inverses
===================
```

```
No errors

=======================
Validating Dictionaries
=======================
No errors

================
Validating Forms
================
Control toolBar in Background has no corresponding property
Control tCustomers in Background has no corresponding property

==================
Validating Methods
==================
Method Background::load has errors
Method Background::toolBar_mouseDown has errors
Method Background::toolBar_mouseUp has errors

=======================
Validating Reorg Status
=======================
No errors

Schema validate completed.
There were 5 errors.
Elapsed time = 00:00:03
```

If your validation output was directed to your printer or the log file, focus is then returned to the Schema Browser.

Notes If a selected definition requires modification (for example, a method requires compilation or the schema requires reorganization), you must perform the appropriate actions to effect the required change.

Errors are detected when references, dictionaries, or reorganization are validated only when your schema is corrupted, for some reason.

System Map Files

The logical entity representing a database file, sometimes referred to as a *map* file in Jade, provides a convenient way of separating the user's view of data from some of the practical considerations associated with storing that data on disk. (For details about viewing and maintaining class map files, see "Using the Class Maps Browser", in the following subsection.)

The read-only system map files listed in the following table store persistent system objects.

Map File	Description
_jadeapp.bin	Contains Jade Platform development environment application data
_jadedef.bin	Default map file for Jade Platform development environment objects
_sysdef.bin	Default map file for system objects

Map File	Description
_sysdev.bin	Contains system meta data objects defined in the **JadeSchema** schema and **JadeToolsSchema** schema; that is, the Jade Platform development environment, code coverage, and XML Metadata Interchange (XMI)
_sysgui.bin	Contains GUI data for system schemas
_sysint.bin	Contains system internationalization data
_system.bin	Contains meta data objects defined in the **RootSchema** schema
_systools.bin	Contains meta data objects defined in the **JadeMonitorSchema** schema
_sysxrf.bin	Contains system schema cross-references

The system files are as follows.

- Jade Platform development environment system files are **_sysdev**, **_jadeapp**, and **_jadedef**
- Tools system files are **_systools** and **_monitor**
- Deployment system files are **_system**, **_sysgui**, **_sysxrf**, **_sysint**, and **_sysdef**

When deploying a Jade system, you can optionally:

- Mark development system files offline, if you want to prevent users from using the Jade Platform development environment.
- Mark development system files offline and do not deploy them, to prevent users from using the Jade Platform development environment. When the Jade Platform development environment is required, put these files in the deployed system and mark them as online.

  Note The separation of runtime and development files is not intended to stop unlicensed users running the Jade Platform development environment but to enhance development mid-release without affecting run time environments. However, if a Jade system had a production (non-development) licence applied, Jade will not allow you to apply a free developer licence. If the Jade system already has a development licence, users can apply the additional development system files.

- When providing a system to customers, mark the development system files offline and do not include them in the installer. This prevents customers from using the Jade Platform development environment and other tools.
- When applying Jade patch releases, customers do not need to deploy the development or tools system files, thereby minimizing risk to their deployed system.

Notes Marking **_sysdev**, **_jadeapp**, and **_jadedef** system files offline prevents the Jade Platform development environment from being used.

Marking **_systools** and **_monitor** system files offline prevents the Jade Monitor, code coverage, and XML Metadata Interchange (XMI) from being used. For details, see the **jdbutilb** batch Jade Database utility **markOffline** command and **markOnline** command in Chapter 1 of the *Database Administration Guide*.

The default database map files listed in the following table store persistent objects in your Jade schemas.

Map File	Description
_control.dat	Database control file

Map File	Description
_environ.dat	Maintains environmental objects (for example, system, session, node, and process objects)
_monitor.dat	Used by the Jade Monitor
_rootdef.dat	Default map file for user objects (note that this is a user *data* file)
_stats.dat	Used for storing statistics information
_userdev.dat	Contains user development-time non-schema data
_usergui.dat	Contains user GUI data (for example, skins information and ActiveX controls)
_userint.dat	Contains user internationalization data
_userscm.dat	Contains user schema definitions
_userxrf.dat	Contains user schema cross-references

Using the Class Maps Browser

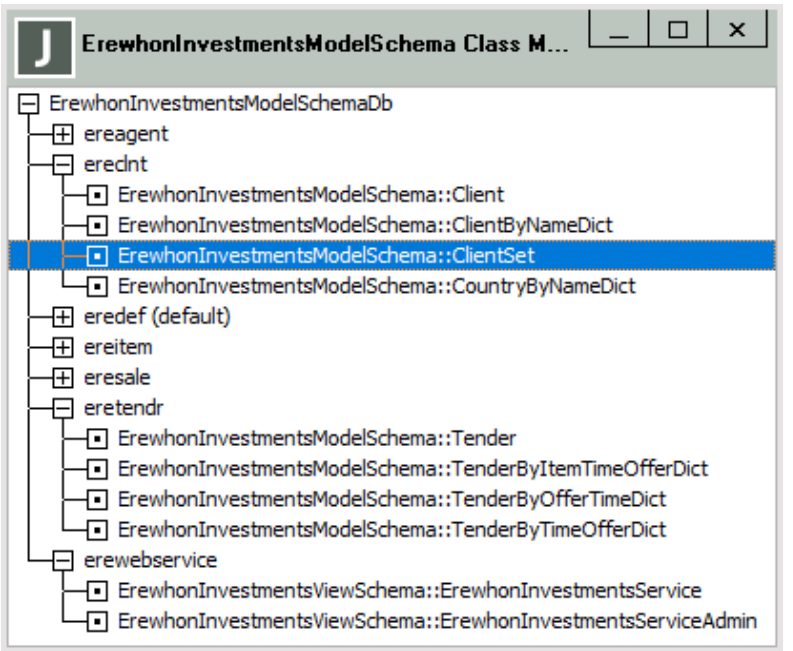
Use the Class Maps Browser to add or maintain the class map files in which persistent objects in the schema (that is, instances of the schema classes) are stored.

When you define a class, you can specify a map file for that class.

To open a Class Maps Browser window, perform one of the following actions

- Select the **Maps** command from the Browse menu.
- Click the Browse Maps toolbar button.
- Press Ctrl+M.

The Class Maps Browser, shown in the following image, is then displayed. (You can change your default browser options, if required, by using the **Browser** sheet from the Options menu **Preferences** command.)



Note Only one Class Maps Browser window can be open at any time.

The Map Files List window displays the names of all class map files for the current schema. The Map Files List window is displayed in the form specified by your Browser options (that is, it is user-specific).

The Jade default display of the Map Files List window is as follows.

- System map files are displayed in red.
- Class names are sorted.
- Each class name is prefixed with the name of the schema in which it is defined.
- User-defined map files are displayed in black.
- Files are displayed in alphabetical order.

» To view all classes mapped to a specific map file

- Click the collapsed node icon of the appropriate map file.

The selected map file node is then expanded, and all classes mapped to the selected file are displayed in the hierarchy tree.

Maintaining Map Files

Use the commands in the MapFiles menu from the Class Maps Browser to browse and maintain your class map files. The map file specifies where instances of schema classes are stored.

When you create a new class, specify the file in which instances of that class are to be stored. Before you can specify the map file for a class in the Define Class dialog, the map file must already exist.

When you create a new schema, specify the file in which the instances of the **Global** class are to be stored. Before you can define a new schema, the map file for these classes must also exist.

Tips To quickly add a map file for your **Global** classes when adding a schema, click the **Add Map File** button in the Add Schema dialog. (For details, see "Defining a Schema", earlier in this chapter.)

When you install Jade, system class map files are located in the **system** directory of your current Jade release, by default. For ease of administration, you may want to also store your user-defined class map files in the same data directory.

If you want to override the default mapping (that is, instances of exclusive collections are mapped to the same file as the owner of the collection) when there are no existing exclusive instances of the collection class, see "Tuning Collection Classes", later in this chapter.

The MapFiles menu commands are listed in the following table.

Command	Description	For details, see…
Add	Displays the File Dialog	Adding or Changing a Map File, in the following subsection
Change	Displays the File Dialog	Adding or Changing a Map File, in the following subsection
Remove	Deletes the selected map file	Removing a Schema Element, later in this chapter

Adding or Changing a Map File

When you select the MapFiles menu **Add** command or **Change** command from the Class Maps Browser, the File Dialog is then displayed. This dialog is also displayed when you click the **Add Map File** button on the on the advanced view of the Add Schema dialog or on the **Class** sheet of the Define Class dialog.

You can change map files (that is, the path in which your class map file is located) only by using the batch Jade Database utility. For details, see the **setFilePath**, **setFilePathAudited**, **clearFilePath**, and **clearFilePathAudited** commands, in Chapter 1 of the *Database Administration Guide*.

To add a new map file or change the name of an existing map file

1. In the **File Name** text box, specify a valid file name prefix for the default class map file (for example, **defmap**) or change the name of the existing class map file, if required. The name must be unique to the schema to which it is being added or updated.

   When the class map file is created, it is allocated a .**dat** suffix.

2. Check the **Partitionable** check box if the database map file can be partitioned. (You cannot change the value of the **Partitionable** check box if the **DbFile** class **isPartitioned** method returns **true**.)

3. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The specified class map file is then displayed in the schema hierarchy of the Class Maps Browser. New classes in the schema can now be mapped to this file, or existing classes can be mapped to the file, if required.

Defining Applications

Within your schema, you can have one or many applications. While each application in a schema can differ in appearance and functionality, it shares the same underlying object model as other applications in that schema; that is, the object model defined by the schema.

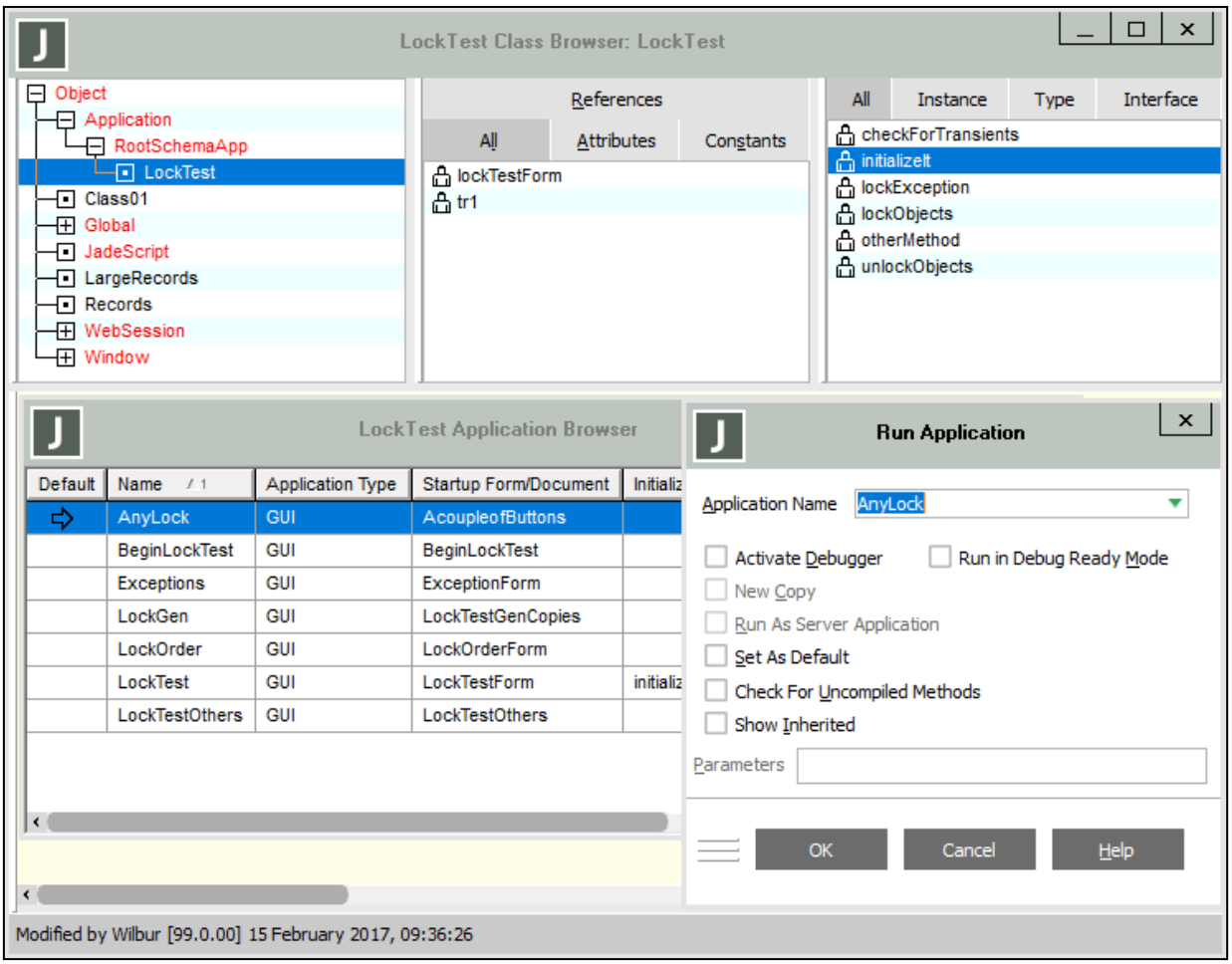
The **Application** class provides a superclass for all user application instances. Each user application is defined as an instance of the **Application** class. The **Application** class defines standard properties and methods for the running of any application.

Each time a new schema is defined, an instance of the **Application** class is created for that schema. When a schema is loaded from a file and there is application data (in the .**ddb** or .**ddx** file), Jade creates an instance of the **Application** class for each application defined in this file.

If the **initialize** or **finalize** method is not set for an application but it is defined in the application or any superclass other than the RootSchema **Application** class, the Application Browser displays that method in the respective **Initialize Method** or **Finalize Method** column of the Application Browser. This enables you to see from the Application Browser the default **initialize** and **finalize** methods that will be invoked when running the application.

Tip Double-clicking on a method name in the **Initialize Method** or **Finalize Method** column displays the method source for that method.

The following example shows the **Application** class node expanded in the Class List of the Class Browser, with the Application Browser and Run Application dialog illustrating the connection between the **Application** class and the user applications defined in that class.



The Application Browser provides a summary of application attributes, which are displayed in a table that has the following columns.

1. **Default**, with an arrow in the row of the default application.
2. **Name**, containing the application name.
3. **Application Type**, containing the application type.
4. **Startup Form/Document**, containing the name of the startup form, if specified.
5. **Initialize Method**, containing the name of the application **initialize** method, if defined.
6. **Finalize Method**, containing the name of the application **finalize** method, if defined.
7. **About Form**, containing the About form of the application, if specified.
8. **Web App Type**, containing the web application type; that is, **HTML Documents**, **Web Services**, **Rest Services**, or **Jade Forms.**

9. **Version**, containing application version number, if specified, to be displayed in the default About box for the application.
10. **Default Locale**, containing the default locale to be used when the application is run under a locale that is not supported by the schema of the application.
11. **Icon**, containing the icon, if specified, to be used for your application instead of the default icon.
12. **Font**, containing the typeface and attributes, if specified, of the font for the forms in the application to be used instead of the default font.
13. **Help File**, containing the name and location, if specified, of the application help file.

Tip Right-click in the table on the Application Browser to display the Application menu commands.

Clicking on the heading of a column sorts the table by that value (for example, **Application Type**). If the column is not the **Name** column, the table is sorted using the selected column and then the **Name** column as the second sort column. Clicking on the existing sorted column heading toggles ascending sort order.

Double-click on the name of:

- The startup form or startup document for an application, to open a new hierarchy browser window for that class.

  If you do not have development environment security access to view the class, the request is rejected.

- An initialize or finalize method for an application, to open a new method source window for that method, to enable you to edit the method.

  If you do not have development security access to view the method, the request is rejected. If you do not have change access, any change is rejected.

The **LockTest** schema shown in this example has user-defined applications, which in the development environment are persistent instances of the **Application** class. These user-defined applications inherit any constant, property, or method defined in the **Application** class or that you defined in the **LockTest** subclass.

View persistent user-defined applications instances by selecting the Classes menu **Inspect Instances** command or from the Application Browser. View shared transient instances for a node by selecting the Classes menu **Inspect Shared Transients** command.

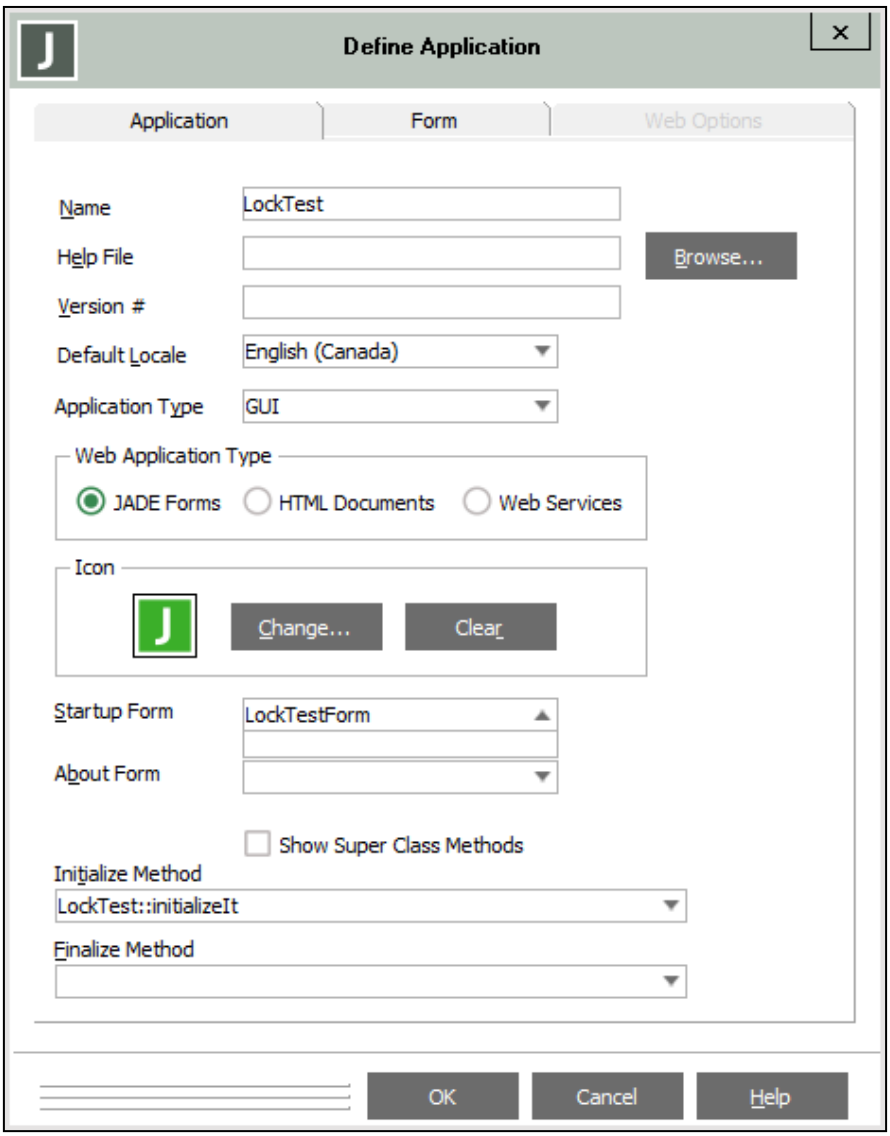
To run an application, select the required application in the Run Application dialog. A transient instance of the class is then automatically made available to the run time copy of the application.

To add an application to your current schema

1. Perform one of the following actions to open an Application Browser window.
   - Select the **Applications** command from the Browse menu
   - Click the **Browse Applications** toolbar button
   - Press Ctrl+L

   The Application Browser is then displayed.
2. From the Application menu of the Application Browser, select the **Add** command.

The **Application** sheet of the Define Application dialog, shown in the following image, is then displayed, to enable you to specify general application details.



3. In the **Name** text box, specify the name of your new application. An entry in this text box is mandatory. The name must start with an uppercase character and it must be unique to the schema to which you are adding it.

4. In the **Help File** text box, specify the name and location of the application help file, if required. For example:

```
c:\jade\testapp.hlp
```

Use the **Browse** button to search for your help file, if required. When you click the **Browse** button, the common File dialog is then displayed, to enable you to select the appropriate file or location. The help file can be an Adobe Acrobat Portable Document Format (.**pdf**) file, Hypertext Markup Language (.**htm** or .**html**) files, a Windows help (.**hlp**) file, or a compiled help (.**chm**) file.

Note If you do not specify a help file in this text box or you do not use the **Application** class **helpFile** property to dynamically set the help file at run time, no help file is opened when help is requested.

5. In the **Version #** text box, specify the application version number, if required.

6. In the **Default Locale** combo box, select the default locale (language) for your new application, if required. The locales that are listed are those supported by the current schema. If you do not specify a default locale, the current locale is set to the first locale supported by the schema.

   When the default locale is set in the Jade Platform development environment, the *Locale not supported* message box is not displayed in the runtime application.

   The default locale is used when the application is run under a locale that is not supported by the schema of the application. The application cannot be run under a locale that is not supported, as no form and string translations will exist. If you try to run an application under a locale that is not supported by the schema, the current locale is set to the application default locale.

7. Use the **Application Type** combo box to select the type of application that you require, as follows.

   - GUI

     The default application type, which provides Windows and print facilities, and so on.

   - GUI, no forms

     Specifies a graphical user interface application that has no form display.

   - Non-GUI

     Specifies an application that can run in a client node or a server node. The **initialize** method of a non-GUI application should not use any GUI facilities and should not invoke printing services. A non-GUI application runs on the node on which the **startApplication** method of the **Application** class is executed (that is, if this method is executed on a server node, the application is started on that server node).

     Alternatively, you can start a non-GUI application by using the **ServerApplication** parameter in the [JadeServer] section or the [JadeAppServer] section of the Jade initialization file, which starts the application when the server node is initialized, or you can check the **Run As Server Application** check box in the Jade Platform development environment Run Application dialog.

   - Rest Services

     Specifies that the application can be accessed from the Internet, if required. The **Web Options** sheet is then enabled.

     The defined for the application is the first web page that is displayed when the application is invoked from the web browser. Application features such as the start-up form, Multiple Document Interface (MDI) forms, and three-dimensional controls are ignored for REST-enabled applications.

   - Rest Services, Non-GUI

     Specifies that the application can be accessed from the Internet as a background task, if required. The **Web Options** sheet is then enabled.

     Application features such as the start-up form, Multiple Document Interface (MDI) forms, and three-dimensional controls are ignored for REST-enabled applications. In addition, as applications of type **Rest Services, Non-Gui** do not display the Web Application Monitor window, this type of application can be run in the background.

   - Web-enabled

     Specifies that the application can be accessed from the Internet, if required. The **Web Options** sheet is then enabled.

The start-up form defined for the application is the first web page that is displayed when the application is invoked from the web browser. Application features such as Multiple Document Interface (MDI) forms and three-dimensional controls are ignored for web-enabled applications.

- Web-enabled Non-GUI

  Specifies that the application can be accessed from the Internet as a background task, if required. The **Web Options** sheet is then enabled. The start-up form defined for the application is the first web page that is displayed when the application is invoked from the web browser.

  Application features such as Multiple Document Interface (MDI) forms and three-dimensional controls are ignored for web-enabled applications. In addition, as applications of type **Web-Enabled Non-GUI** do not display the Web Application Monitor window, this type of application can be run in the background.

For more details, see the **Application** class **applicationType** property in Chapter 1 of the *Encyclopaedia of Classes*.

Notes Applications of type **No Forms**, **Non_GUI**, **Rest Services, Non-Gui**, and **Web-Enabled Non-GUI** terminate only after the Jade **terminate** instruction is executed.

The **Application** class **startApplication**, **startAppMethodWithString**, and **startAppMethod** methods start only **Non-GUI**, **Rest Services, Non-Gui**, and **Web-Enabled Non-GUI** applications if they are invoked from a server method or server application. (An exception is raised if they are invoked from a server method or a server application to start an application of a type other than non-GUI.) On a client node, they start all types of application.

---

8. If you selected the **Web-enabled** or **Web-enabled Non-GUI** application type in the previous step, the Web Application Type group box is then enabled. If you do not want to dynamically generate HTML based on Jade forms, select the:

   - **HTML Documents** option button if you want HTML generated based on HTML documents in your web application. (For details, see "Specifying Your HTML Thin Client Access Options", in Chapter 1 of the *Web Application Guide*.)
   - **Web Services** option button if you want your application to support web services. (For details, see "Defining a Web Service Provider Application", in Chapter 11 of the *Developer's Reference*.)
   - **Rest Services** option button if you want your application to support REST services. (For details, see "REST-Based Web Services", in Chapter 11 of the *Developer's Reference*.)

   By default, HTML is dynamically generated in web applications based on Jade forms; that is, the **Jade Forms** option button is selected.

9. In the Icon group box, click the **Change** button if you want to display the common File Open dialog, to enable you to select an icon (.**ico**) file for your Jade application icon display.

10. In the **Start-up Form** combo box, select the form that is to be displayed when the new application is started.

    Notes The first form that you create for the application becomes the default start-up form when you run your application. (For details, see "Adding a New Form", in Chapter 5.)

    If your application type is **Web-Enabled** or **Web-Enabled Non-GUI** and you do not select or specify a value in this combo box, the **HTML Documents** sheet on the **Web Options** sheet is enabled so that you can define an HTML home page.

    ---

11. In the **About Form** combo box, select the form that is to be displayed in response to a user of the runtime application selecting the **About** command in the Help menu.

12. Check the **Show Super Class Methods** check box if you want applicable methods from the current schema and all superschemas displayed in the following **Initialize Method** and **Finalize Method** combo boxes. When this check box is unchecked, applicable methods from the current schema are displayed in the following combo boxes.

    This check box is initially unchecked for a new application definition or when you are changing an application and the **Initialize Method** and **Finalize Method** combo boxes are empty or the methods are defined in the current schema. If an application being changed has a defined **initialize** or **finalize** method that is defined in another schema, the check box is checked and disabled.

    If you check this check box to toggle the checked status, the **Initialize Method** and **Finalize Method** combo boxes are reloaded according to the new status.

    When you select an **initialize** or **finalize** method from a superschema, the check box is disabled so that you cannot uncheck the check box, which would prevent the display of the currently selected method.

13. In the **Initialize Method** combo box, select the start-up method that is to be invoked when the application is started (for example, the **initialize** method), if required. This must be a method of the **Application** class.

    Note The **Application** class **initialize** method is called automatically if you do not explicitly specify a start-up method in the **Initialize Method** combo box.

14. In the **Finalize Method** combo box, select the method that is to be invoked when a close request has been made for your application. This must be a method of the **Application** class. When the selected **finalize** method is invoked, the application then terminates, and cannot handle new form requests.

15. Select the **Form** sheet if you want to specify your form preferences. (For details, see "Specifying Your Form Preferences", in Chapter 5.)

    Alternatively, perform one of the following actions.

    - Select the **Web Options** sheet if you want to specify your options for Jade applications accessed from the Internet. For details, see "Enabling Your Jade Application for HTML Thin Client Access", in Chapter 1 of the *Web Application Guide*, or "Defining a Web Service Provider Application", "REST-Based Web Services", or "ISO 8601 Date and Time Support" in Chapter 11 of the *Developer's Reference*.
    - Click the **OK** button.

Your specified application is then created as an instance of the **Application** class and is therefore not displayed in the Class List of the Class Browser. It is displayed in the Application Browser, however, and can be viewed from the Class Browser by inspecting instances of the **Application** class.

Notes Only one Application Browser for the current schema can be open at any time. If an Application Browser is already open for that schema, it is brought to the top when you click the **Browse Application** toolbar button or you select the **Applications** command from the Browse menu.

You can have concurrent open Application Browsers for different schemas in the current development environment session.

Setting an Application

Use the **Set** command from the Application menu to set the selected application to the current active application. Alternatively, you can use the:

- **Set As Default** check box in the Run Application dialog

  If the default application is set to a superschema application on the Run Application dialog, that superschema

application is the default application.

- Application name combo box at the far right of the status line in the background window

Set an application for:

- Running an application from the Jade Platform development environment; that is, by performing one of the following actions.
  - Select the **Run** command from the Application menu.
  - Click the **Run Application** toolbar button.

  Note As you can run an application only in the current schema context, the **Run Application** toolbar button and **Run** command are disabled in latest schema browser windows.
- The default application when executing or debugging methods in the **JadeScript** class. Running a **JadeScript** method always starts a new application copy.
- The default application to be extracted for a partial (selective) schema extract.

» To set an application as the current application, perform one of the following actions

- From the Application Browser:
  a. Select the application that you want to be the current application.
  b. Select the **Set** command from the Application menu.

  Tip Double-click on one of the first three columns in the table on the Application Browser (that is, the **Default**, **Name**, or **Application Type** column), to quickly set it as the current application.
- From the Run Application dialog:
  a. In the **Application Name** text box, select the application that you want to run, if it is not the current (default) application.
  b. Check the **Set As Default** check box.
- From the background window:
  a. Click on the application name displayed at the far right of the status line.

     A combo box is then displayed, listing all of the application names for the currently selected schema, with the current default application selected.
  b. Select the application you require as the new current application default.

  To cancel the combo box display, press the Esc or Tab key, or move focus to another window.

The selected application is then displayed in the Application Browser with an arrow pointing to the left of the application name, indicating that it is the current application. (When you define a new schema, the application instance created by Jade is automatically set as the current application.) That application remains the current application until you set another application.

Notes You can remove user-defined applications from a schema, providing that at least one application remains.

You cannot remove an application when it is the currently set application and a user is currently using that application (for example, if you are running Jade in single user mode).

What Happens Next

When you have defined your application:

1. Access the Class Browser or Primitive Types Browser.
2. Define the classes, forms, methods, and properties that you require for your application.

For Details About…	See…
Accessing and using the browsers and defining classes	The following sections
Defining methods, conditions, properties, and constants	Chapter 4
Defining forms	Chapter 5
Specifying your web access options	Chapter 1 of the *Web Application Guide* or Chapter 11 of the *Developer's Reference*

Using the Class, Primitive Types, or Interface Browser

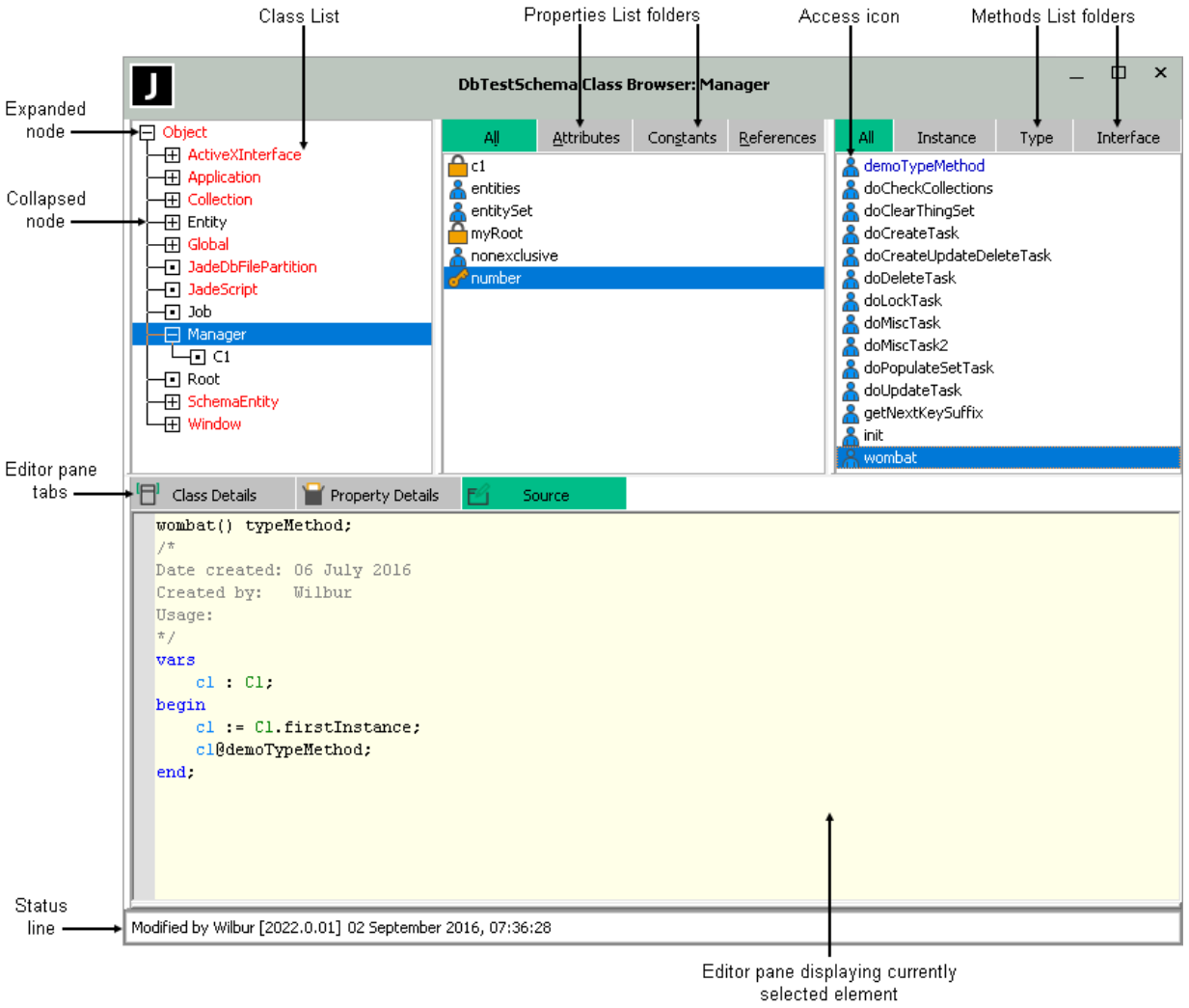
Use the Class Browser to add a class to or maintain classes and their associated methods, properties, and constants, the Primitive Types Browser to add or maintain primitive type methods and constants, or the Interface Browser to add or maintain interface methods and constants. (For details about interfaces, see "Adding and Maintaining Interfaces", in Chapter 14.)

The Class Browser, Primitive Types Browser, and Interface Browser are displayed in the form specified by your Browser options.

By default, the **Object** class is selected when a Class Browser is opened. The **Object** node is expanded (that is, its subclasses are displayed), while all other nodes are collapsed.

As the display of the browsers is user-specific, you can change your default browser options by using the Browser Options from the Options menu **Preferences** command. (For details, see "Maintaining Browser Options" under "Setting User Preferences", in Chapter 2.)

The components of the Class Browser are shown in the following image.



By default, the Class, Properties, and Methods List windows and the editor pane are displayed in the Class Browser.

Note From version 2020.0.01 and higher, a horizontal scroll bar is displayed in hierarchy browser list boxes if any list entry is truncated horizontally.

If a method selected in the Methods List is inherited from a superclass, this reimplementation information is displayed in the status line, as shown in the following example.

```
Modified on 04 May 2007, 09:15:14[6.2.11] (reimplemented)
```

The Primitive Types Browser contains the Primitive Types, Constants, and Methods List windows and the editor pane, all of which are displayed by default when the browser is first accessed.

The Interface Browser contains the Interfaces, Constants, and Methods List windows and the editor pane, all of which are displayed by default when the browser is first accessed.

Note You can have any number of Class Browsers, Primitive Types Browsers, and Interface Browsers open concurrently for each schema, if required. This enables you to easily view components of other schemas and to work concurrently in two or more schemas in the same Jade database. For example, you can browse the classes, properties, or methods in an existing schema while you define a new schema. (Use the Schema menu **Close Windows for Schema** command to close all open windows for the schema of the window that currently has focus.)

Opening a Class, Primitive Types, or Interface Browser

To open a Class Browser, perform one of the following actions

- Select the **Classes** command from the Browse menu of the Class Browser, Primitive Types Browser, or Schema Browser
- Click the **Browse Classes** toolbar button
- Press Ctrl+B

A Class Browser window is then opened. When you perform this action using the Ctrl+B shortcut keys the new Class Browser is opened, with the class selected in the new browser defaulting to the same class that is currently selected when the action is performed from a hierarchy browser.

Note More than one Class Browser for a schema can be open at any time. You can also have concurrent open Class Browsers for different schemas in your current development work session.

To open a Primitive Types Browser, perform one of the following actions

- Select the **Primitive Types** command from the Browse menu of the Class Browser, Primitive Types Browser, or Schema Browser
- Click the **Browse Primitive Types** toolbar button
- Press Ctrl+T

A Primitive Types Browser window is then opened.

Note More than one Primitive Types Browser for a schema can be open at any time. You can also have concurrent open Primitive Types Browsers for different schemas in your current development work session.

To open an Interface Browser, perform one of the following actions

- Select the **Interfaces** command from the Browse menu from the Class Browser, Primitive Types Browser, Interface Browser, or Schema Browser
- Click the **Browse Interfaces** toolbar button
- Press Ctrl+N

An Interface Browser window is then opened. For details, see "Adding and Maintaining Interfaces", in Chapter 14.

Note More than one Interface Browser for a schema can be open at any time. You can also have concurrent open Interface Browsers for different schemas in your current development work session.

Displaying Bubble Help in Browser Lists

Bubble help is displayed for the entity over which the cursor is positioned in a list in the current Class Browser, Primitive Types Browser, or Interface Browser. For example, when you move the mouse so that cursor is positioned over a method in the Methods List of a browser, the signature of that method is then displayed in bubble help, as are summary details about a property when you position the cursor over a property in the Properties List of the Class Browser or a constant in the Constants List of the Primitive Types or Interface Browser.

You can display bubble help for classes, methods, properties, and constants in the current browser, if applicable. The details displayed in bubble help are those that are displayed in the editor pane when you select the entity in a browser list.

When a method displayed in the Methods List in the Jade Hierarchy Browser and Methods Browser is hovered over, the bubble help includes the schema and class of the method as well as the method signature.

Bubble help is displayed for items in browser lists by default. To hide the display of bubble help in the current browser, select the **Show Bubble Help** command from the View menu. (This command enables you to toggle the display of bubble help. A check mark is displayed to the left of the command in the View menu when bubble help is selected for display.)

Note Overriding the Jade Platform development environment default bubble help display applies only to the current browser.

The default value applies when you subsequently close and reopen that browser in your work session.

For details about displaying bubble help in the editor pane, see "Using Bubble Help in the Editor Pane", in Chapter 4.

Displaying Lists and the Editor in the Current Browser Window

The View menu contains commands that enable you to toggle the display of the Constants List, Properties List, Methods List, or the editor pane in the current browser.

A check mark is displayed to the left of the command in the View menu when that browser component is selected for display.

These commands override the settings of check boxes in the Windows group box of the Preferences dialog **Browser** sheet, which apply to all browsers in the Jade Platform development environment. The commands and the Preferences dialog check boxes that they override are listed in the following table.

Command	Check Box Overridden	Toggles the display of the…
Editor Window	Integrated Editor	Editor pane in the current browser
Properties Window	Show Properties	Properties List in the current Class Browser
Constants Window	Not applicable	Constants List in the current Primitive Types or Interface Browser
Methods Window	Show Methods	Methods List in the current browser

The integrated editor pane, the Properties List or Constants List, and the Methods Lists are displayed in browser windows by default.

Note Overriding the Jade Platform development environment default list display applies only to the current browser. The default value applies when you subsequently close and reopen that browser in your work session.

Toggling Property, Method, and Constant Access Type Display

The Methods List of the Primitive Types Browser and Interface Browser and the Properties, Constants, and Methods Lists of the Class Browser contain symbols that indicate the access type or status of constants, properties, and methods.

These symbols and the View menu commands that you can select to hide or display these icons in the current browser are listed in the following table.

No Version	Description	Command to Toggle Display
	Property, method, or constant is public	Show Public
	Property or method is protected	Show Protected
	Property is read-only	Show Read Only
	Property is a key	Not applicable
	Method contains one or more compilation errors	Toggling not applicable to method error display
	Entity in the Methods List is a condition	Conditions

These constant, property, and method types are listed in the appropriate browser list by default. A check mark is displayed to the left of the command in the View menu when that entity is selected for display. (For details about toggling the display of methods or conditions, see "Toggling the Display of Methods and Conditions", in the following subsection.)

The **Show Public**, **Show Protected**, and **Show Read-Only** (applicable only to properties) commands in the View menu override the settings of respective check boxes in the Access group box of the Preferences dialog **Browser** sheet, which apply to all browsers in the Jade Platform development environment.

Notes Overriding the Jade Platform development environment default display applies only to the current browser. The default value applies when you subsequently close and reopen that browser in your work session.

Although the icon symbols are displayed in browser list windows by default, you can use the **Do not show Icons in Browser Window** check box in the **Browser** sheet of the Preferences dialog to hide the display of all browser list icons or display all icons if they are currently hidden. For details, see "Maintaining Browser Options", in Chapter 2.

If a method, property, or constant is versioned, the left arrow or right arrow is superimposed onto the icon to indicate the version of that element; that is, a left arrow indicates the current version and the right arrow indicates the latest (uncommitted) version.

If you have a browser open against the:

- Current version of the schema, all icons are unadorned or they have a superimposed left arrow that indicates that the method or property is versioned.
- Latest version of the schema, all icons are unadorned or they have a superimposed right arrow that indicates that the method or property is versioned.

The current version icon indicates the element with which the system is currently running and the latest version icon indicates the latest (uncommitted) version of that element.

Browsers can display a composite view, which shows:

- Any versioned class is highlighted with the user-defined versioned background color, which is light green by default. The same background color is applied to the latest version of any method, property, or class constant.
- Both versions of any versioned method, property, or class constant.
- In the current version context, methods, properties, or constants that have been added to the latest version context.

  To provide additional feedback in the current version context, foreground colors in the browsers indicate one of three versioning states, as follows.

  - Changed color (which defaults to orange) indicates that a property or class constant has changed between the current and latest version.
  - Added color (which defaults to pink) indicates a new method, property, or constant added to current context.
  - Removed color (which defaults to gray) indicates a method, property, or class constant that exists in the current schema context and that has been removed from the latest schema context. This element is removed from the current schema when the class is next reorganized.

You can change the version display foreground and background colors, by using the **Window** sheet of the Preferences dialog, accessed from the **Preferences** command in the Options menu of browse windows. (For details, see "Maintaining Window Options", in Chapter 2.)

Toggling the Display of Methods and Conditions

The View menu contains the commands listed in the following table that enable you to toggle the display of methods or conditions in the Methods List of the current browser.

Command	Toggles the display of…
Methods	Jade and external methods in the Methods List of the current browser
Conditions	Conditions in the Methods List in the current browser

A check mark is displayed to the left of the command in the View menu when that entity is selected for display. Jade methods, external methods, and conditions are displayed in the Methods List by default.

Toggling the Display of Entities in Imported Classes

The View menu contains the command listed in the following table.

Command	Toggles the display of…
Show Imported Classes	Classes (and their properties and methods, when applicable) imported into the current schema

A check mark is displayed to the left of the command in the View menu when that entity is selected for display. Classes in imported packages are displayed in the Class List by default.

Changing the Sort Order of Displayed Classes

You can toggle the sort order of classes displayed in the Class List of the Class Browser. By default, classes are displayed in hierarchical order but you can display them in alphabetical order, if required.

Note As no class hierarchy is visible when classes are displayed in sorted (alphabetical) order, you cannot see the structure of the schema.

To display classes in alphabetical order in the Class List of the Class Browser

- Select the **Sorted Order** command from the View menu.

Jade then sorts all classes so that they are displayed in alphabetical order in the Class List window, and a check mark is displayed at the left of the command in the View menu to indicate that classes are currently displayed in alphabetical sort order.

For details about changing the display sort order so that it is effective for the current and all future work sessions until it is changed, see "Maintaining Browser Options", in Chapter 2. (As the default value is user-specific, it applies only to your work sessions, and not those of other users.)

Showing Inherited Methods, Conditions, Properties, and Constants

You can display all constants, properties, methods, and conditions inherited from all superclasses by the class selected in the Class List of the Class Browser or Interface Browser. The Properties List and Methods List of the Class Browser and the Constants List and Methods List of the Interface Browser display only the constants and methods of the selected class or interface, and not those inherited from superclasses.

By default, inherited objects are not displayed, and they can be displayed in subclasses only if they are not specified as *final* in the superclass. (For details, see the **final** option under "Controlling the Use of Elements in Other Schemas", in Chapter 1.)

Methods that are reimplementations of superclass methods display **(r)** on the end of the list item method name; for example:

```
displayCustomer (r)
```

Note This is not done, however, for event methods, which are always reimplementations of a superclass method.

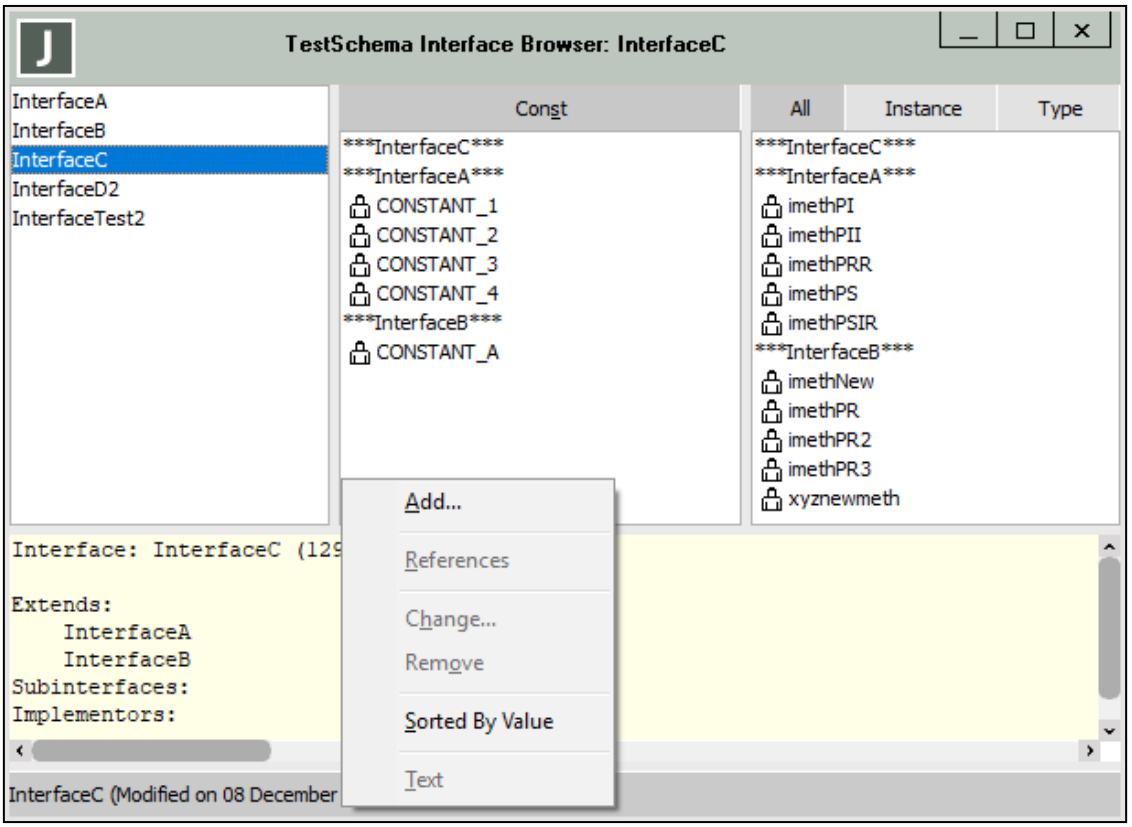
When a reimplemented superclass method is displayed in the editor pane, **(reimplemented)** is displayed in the status bar of the browser.

To display inherited objects in the Class Browser or Interface Browser

- Select the **Show Inherited** command from the View menu.

The Properties List (or Constants List) and the Methods List then display all objects provided by the selected class or interface as well as those provided by superclasses, if applicable. A check mark is displayed at the left of the command in the View menu to indicate that objects inherited by the class or interface are also displayed.

The objects are listed below the classes or interfaces in which they are provided, as shown in the following image.



For details about changing the display of inherited methods, properties, and constants so that it is effective for the current and all future work sessions until it is changed, see "Maintaining Browser Options", in Chapter 2. (As the default value is user-specific, it applies only to your work sessions, and not those of other users.)

For details about displaying entities inherited from superschemas, see "Displaying All Superschema Classes in Your Class Browser", later in this chapter.

Refreshing the Browser

You can refresh (update) the contents of a browser (for example, the Class Browser or Interface Browser) to reflect all changes made since the browser was opened.

Note As single user mode browser windows are dynamic, you will need to use this command only in a multiuser Jade environment to update the browser with changes made by other users.

To update the current browser

- Select the **Refresh** command from the View menu.

The browser is then updated to reflect any changes to the schema made by other users. There may be a momentary delay while this updating occurs.

Using the Class List, Primitive Type List, or Interface List Window

The Class List, Primitive Type List, or Interface List window is displayed in the form specified by your Browser options; that is, it is user-specific.

The Jade default display of the Class List, Primitive Type List, or Interface List window is as follows.

- System classes or primitive types are displayed in red.
- User-defined classes that are inherited from a superschema are displayed in blue.
- User-defined classes and interfaces that are defined in this schema are displayed in black.
- Classes and interfaces imported from a package are displayed with a green foreground color.
- Classes are displayed in hierarchical order, with class nodes other than the **Object** class node collapsed; that is, subclasses are not displayed when the Class Browser is first opened.
- Versioned classes, primitive types, or interfaces have a light green background and an unfilled left arrow on the icon indicating the current version with which the system is running or a bright green-filled right arrow on the icon indicating the latest version of that class, primitive type, or interface.

The actions that you can perform by using the mouse from the Class List, Primitive Type List, and Interface List window are listed in the following table.

Action	Result
Click	The properties and methods provided by the selected class populate the Properties List and Methods List windows of the Class Browser. The definition of the selected class is displayed in the editor pane. When a node icon is selected, a collapsed node is expanded or an expanded node is collapsed. The constants and methods provided by the selected primitive type or interface populate the Constants List and Methods List windows of the Primitive Types Browser or the Interface Browser, respectively. The options defined for the selected primitive type or interface are displayed in the editor pane.
Right-click	Displays a subset of the Classes, Types, or Interfaces menu, to enable you to maintain the selected class, primitive type, or interface.
Double-click	From the Class Browser only, expands all subclasses of the selected class, if applicable.

Use the scroll bar to scroll up and down a list window, if required.

For details about defining classes, see "Adding Classes to Your Schema", later in this chapter. For details about browsing all methods in the class selected in the Class List (by selecting the **Methods Browser** command from the Classes menu), see "Navigating to the Methods for a Specified Class", in Chapter 2.

Using the Properties List Window

The Properties List window is displayed in the form specified by your Browser options; that is, it is user-specific.

Note In the Primitive Types Browser and Interface Browser, the Properties List window contains only constants and it is displayed as the **Const** (Constants) List.

The Jade default display of the Properties List window is as follows.

- System properties or constants are displayed in red.
- User-defined properties or constants that are inherited from a superschema are displayed in blue.
- User-defined properties or constants that are defined in this schema are displayed in black.
- Properties or constants are displayed in alphabetical order.
- Properties or constants imported from a package are displayed with a green foreground color.
- All public, protected, and read-only properties or constants are displayed.
- Design-time dynamic properties are displayed in maroon and runtime dynamic properties in magenta.
- Versioned properties or constants have a light green background and an unfilled left arrow on the icon indicating the current version with which the system is running or a bright green-filled right arrow on the icon indicating the latest version of the property or constant.

Note By default, the Properties List window of the Class Browser displays only the properties of the selected class, and not those inherited from superclasses. To view inherited properties, select the **Show Inherited** command from the View menu or select the **Show Inherited** check box in the **Browser** sheet of the Preferences dialog. (For more details, see "Showing Inherited Methods, Conditions, Properties, and Constants", earlier in this chapter.)

The actions that you can perform by using the mouse from the Properties List window are listed in the following table.

Action	Result
Click	On a property or constant, displays the definition of the selected property or constant in the editor pane. On a folder, displays the selected type of property, constant, menu item, or control. The **Ref** folder displays the reference properties, the **Attr** folder the attribute properties, and the **All** folder displays all attribute and reference properties but *not* the constants for the selected class. The **Const** folder displays the constants of the selected class, primitive type, or interface.
Right-click	Displays the Properties menu or Constants menu, to enable you to maintain the selected property or constant.
Double-click	Displays a definition dialog (for example, Define Attribute dialog), to enable you to modify the definition of the selected property or constant.

Use the scroll bar to scroll up and down the Properties List window, if required. To toggle the display of the Properties List in the current browser (that is, override the **Show Properties** check box value in the **Browser** sheet of the Preferences dialog), select the **Properties Window** command from the View menu. For details about defining properties and constants, see Chapter 4.

Displaying Control Property Values in the Editor Pane

Instead of opening a form in the Jade Painter only to view additional information about a control; for example, its parent hierarchy, caption, or other design-time settings, you can view this information in the editor pane of the Class Hierarchy Browser.

When you click on a **Controls** property in the Properties List of the Class Hierarchy Browser or bubble help for a property and that property is a control or menu instance on a form, additional information is now displayed in the editor pane.

For a control, the display includes the control parent hierarchy within the form and the value of the **caption** (if the control has that property), **left**, **top**, **width**, **height**, **enabled**, and **visible** properties; for example:

```
Name: btnChgCtlPalPic (22)
Class: ClassDefinitionDialog ()
Type: Button
Access: public
Ordinal: 4
non-virtual embedded control

Control parent hierarchy on form:
    ClassDefinitionDialog::Form
        fldDefineClass::Folder
            shtCtrlOptions::Sheet
                grpPainterIcon::GroupBox

Caption: 'Change...'
Left: 48 Top: 59 Width: 80 Height: 59
Enabled: true
Visible: true
```

For a menu property, the display includes the menu parent hierarchy on the form or that the menu item appears on the menu line of the form, and the value of the menu **caption**, **level**, **enabled**, and **visible** properties; for example:

```
Name: mnuRefactorIDeclareVar (216)
Class: JadeWindow ()
Type: MenuItem
Access: public
Ordinal: 251
non-virtual embedded menu

Menu parent hierarchy on form:
    mnuEditMenu Caption: &Edit
        mnuRefactor Caption: R&efactor

Caption: '&Declare Local Variable'
Level: 3
Enabled: true
Visible: true
```

Applies to Version: 2018.0.01 and higher

Using the Methods List Window

The Methods List window is displayed in the form specified by your Browser options; that is, it is user-specific.

Note The Methods List window of browser windows has four folders: **All**, **Instance**, **Type**, and **Interface**. By default, the **All** folder is displayed; that is, all user-defined instance methods, all type methods, and all interface methods are displayed.

In addition, the Methods List can contain the **Form Events**, **Control Events**, and **Menu Events** folders. For details, see "Notes about the Methods List Window", in the following topic.

The Jade default display of the Methods List window is as follows.

- System methods are displayed in red. (You cannot modify system methods.)
- User-defined methods that are inherited from a superschema are displayed in bright blue.

- User-defined instance methods that are defined in this schema are displayed in black.

  An *instance* method is a method whose receiver at run time is an instance of the class, primitive, or interface type.

- User-defined type methods that are defined in this schema are displayed in dark blue.

  A *type* method is a method declared on a class, primitive, or interface type without having to have an instance of the type. At run time, the receiver is the type on which the method is declared.

- Methods, both Jade and external, are displayed in alphabetical order.

- Methods that are reimplementations of superclass methods display **(r)** on the end of the list item method name; for example:

  ```
  displayCustomer (r)
  ```

  Note This is not done, however, for event methods, which are always reimplementations of a superclass method.

- Methods that have one or more interface mappings display **(i)** on the end of the list item method name; for example:

  ```
  addCustomer (i)
  ```

  Note If the method is also a reimplementation, it displays both **(r)** and **(i)** on the end of the list item method name.

- Methods imported from a package are displayed in green.

- All public, protected, and read-only methods are displayed.

- Conditions have the **Condition** symbol displayed to the left of each condition name.

- The caption of the tab of the first folder varies, depending on the entities that are displayed. The caption is displayed as:

  - **Form Events**, when the class is a form
  - **Events**, when a control property is clicked
  - **All**, in all other cases; that is, all user-defined instance, type, and interface methods are displayed. (event methods are not included)

- When you select the **Interface** folder in the Methods List window, any methods with interface mappings defined in the selected class are displayed as a two-level hierarchy.

  The first level is the name of each interface implemented by the class, and the second level is a list of methods with mappings belonging to that interface. If the class method and interface method names differ, the interface method name is displayed in parentheses after the name of the class method. Clicking on the interface name item is ignored (that is, it is disabled).

- Versioned methods have a light green background and an unfilled left arrow on the icon indicating the current version with which the system is running or a bright green-filled right arrow on the icon indicating the latest version of the method. (No arrow is displayed on method icons when a schema is not versioned.)

The actions that you can perform by using the mouse from the Methods List window are listed in the following table.

Action	Result
Click	On a method, displays the Jade method code for that method in the editor pane. On a folder, displays the selected type of method (that is, all, instance, or type methods).
Right-click	Displays the Methods menu, to enable you to maintain a method or add a new method.
Double-click	Displays a new free-standing editor window for the current method.

Use the scroll bar to scroll up and down the Methods List window, if required. For details about defining methods, properties, constants, and conditions, see Chapter 4.

To toggle the display of the Methods List in the current browser (that is, override the **Show Methods** check box value in the **Browser** sheet of the Preferences dialog), select the **Methods Window** command from the View menu. (For details about method options, see "Maintaining Source Management Options", in Chapter 2.)

Notes about the Methods List Window

By default, the Methods List window of the Class Browser or Interface Browser displays only the methods and conditions of the selected class or interface, and not those inherited from superclasses.

To view inherited methods and conditions in the Class or Interface Browser, select the **Show Inherited** command from the View menu or select the **Show Inherited** check box in the **Browser** sheet of the Preferences dialog. If a method selected in the Methods List is inherited from a superclass, this reimplementation information is displayed in the status line, as shown in the following example.

```
Access is public. Created on 07 November 2001, 07:21:15 (reimplemented)
```

To view methods and conditions in the Class Browser that are inherited from superschemas, select the **Superschemas** command from the View menu. For details about using the View Superschemas dialog, see "Displaying All Superschema Classes in Your Class Browser", later in this chapter. To view methods implemented in specific interfaces, select the **Show Interface Method Mappings** command from the View menu. For details, see "Viewing Interface Method Mappings", in Chapter 14.

Note You cannot reimplement conditions that are inherited from a superschema or a superclass.

From version 2018.0.01, methods that are reimplementations of superclass methods display **(r)** on the end of the list item method name; for example:

```
displayCustomer (r)
```

Note This is not done, however, for event methods, which are always reimplementations of a superclass method.

From version 2018.0.01, clicking on the **Form Events**, **Control Events**, or **Menu Events** tab displays only form, control, or menu item property event methods, respectively.

The **Form Events** and **Control Events** tabs are hidden if the class selected in the Class List is not a **Form** class or subclass. In addition, the **Control Events** tab is hidden if the currently selected property is not a control or menu instance.

To edit a:

- Form's event methods, click the **Form Events** tab in the Methods List and then select the required event method.

- Control or menu item event method, click the associated property in the Properties List so that the **Control Events** or **Menu Events** tab is displayed in the Methods List. Click the tab to display the associated sheet, and then select the required event method. This sheet remains visible while the associated property remains selected.

You can switch between the method tabs without affecting the property that is selected and you can edit non-event methods, form event methods, and control or menu item event methods by changing the selected sheet in the Methods List. The only time that a selected control or menu item is affected is if:

- Another class is selected.
- A different property is selected.
- A different property sheet is selected.

Note Selecting a different property in the Properties List causes the Methods List to be reloaded only if the property is a control or menu item event.

Jade supplies many methods and properties that are inherited automatically by all forms and controls in the Class Browser. These supplied methods are dimmed (disabled) in the Methods List window until you supply an implementation for that method.

» **Perform one of the following actions on a selected method or condition**

- Click, to display the code for the method or condition in the editor pane.
- Right-click, to display the Methods menu that enables you to maintain the selected method or condition.
- Double-click, to display a new free-standing editor pane for the current method or condition.

Using the Editor Pane

The editor pane is displayed in the form specified by your editor options; that is, it is user-specific. For details, see "Maintaining the Editor Display", "Maintaining Editor Options", and "Maintaining Source Management Options", under "Setting User Preferences", in Chapter 2.

Use the editor pane to:

- Define new methods or conditions in the selected class, primitive type, or interface (for details, see "Defining and Compiling Jade Methods and Conditions", in Chapter 4).
- Maintain existing methods and conditions.
- Compile methods and conditions.
- Execute methods in the **JadeScript** class of the Class Browser (if selected).
- Change or rename an entity (for example, a property, local constant, variable, or method parameter) selected within the body of a method in the editor pane (for details, see "Renaming or Changing an Entity", in Chapter 4).
- The emoji selection window (displayed using the Windows key + period (**.**), or dot, character key combination) can be used to paste a selected emoji or selection of emojis into the editor pane in a Unicode Jade system. For more details, see "Unicode Surrogate Pair Characters, including Emojis", in Chapter 1 of the *Developer's Reference*.

Use the scroll bar to scroll up and down the editor pane, if required.

To toggle the display of the editor pane in the current browser (that is, override the **Integrated Editor** check box value in the **Browser** sheet of the Preferences dialog), select the **Editor Window** command from the View menu.

You can display multiple sheets in the editor pane. Multiple sheets enable you to navigate away from a method that you are currently editing (to view property or class details, for example) without losing your position in the source or being prompted to save your changes every time. For details, see "Displaying Multiple Sheets in the Editor Pane", later in this chapter.

Jade separates the keywords that are valid in a method from those that are valid only in a schema file so that the editor pane and AutoComplete feature use the list valid for the context; that is, method keywords rather than schema file keywords. Schema keywords are used only when displaying a schema file when a schema load fails because of syntax errors in the schema file being loaded. (For details about the AutoComplete feature, see "Using Jade AutoComplete Functionality", in Chapter 2.)

The Jade installation default display of Jade elements in the editor pane is listed in the following table.

Element	Default Display Color
Background	Light yellow
Instructions (keywords)	Blue
Method elements (for example, **vars**, **on**, and **epilog**)	Blue
String and numeric literals	Red
Global constants	Purple
Primitive, system, and user types	Green
Imported packages	Green
Interfaces	Green
Special characters	Black
Debug (breakpoint) line	Yellow
Debug step line	Light blue
Identifiers (for example, instance variables)	Black
Comments and indent guides	Dark gray
Edge marker	Light blue
Margin and selected text background	Light gray
Caret background	Light green
Linemarks	Light turquoise
Selected text (selection foreground)	Transparent
Caret	Black
Dollar identifiers (for example, in translatable strings and user formats)	Green
Method (AutoComplete)(instance methods)	Brown
Property (AutoComplete)	Light blue
Additional Selections (background color of other text in the method that matches the current selection; for example, opening and closing parenthesis and bracket pairs)	Lime green

Element	Default Display Color
TypeMethod (AutoComplete)	Faded blue
Local Parameter	Bronze if a light theme is in use and medium gray if a dark theme is in use (when AutoComplete is enabled)
Local Variable	Medium blue (when AutoComplete is enabled)
Local Constant	Rose-pink (when AutoComplete is enabled)

Splitting the Editor Pane View

You can split the editor pane into two horizontal views of the same method; that is, you can split the editor pane horizontally into two panes and then control each pane separately.

You can scroll these views independently, to view and edit different parts of the active document at the same time.

Any changes that you make in one view are reflected in the other.

To open or manipulate the second view, perform one of the following actions

- Position the mouse cursor on the split bar just above the status line so that it changes to the resize pointer and then click and drag the split bar to where you want it positioned.
- Use the Shift+Ctrl+W shortcut keys to open or close (that is, toggle) the lower view.
- Use the Ctrl+W shortcut keys to open the lower view if it is not already open or to switch focus between the two views if the lower view is already open.

Adding a Method from a Free-Standing Method Editor Pane

The **New Jade Method**, **New External Method**, and **New Condition** commands in the Methods menu are enabled when the focus is set to a free-standing method editor pane (for example, when you double-clicked on a method in the Methods List of the Class Browser). When you select one of these commands, the corresponding method definition dialog (for example, the Jade Method Definition or Condition Definition dialog) is then opened.

When you add the method, a new free-standing editor pane for that method is opened and focus is set to it so that you can define the new method to meet your requirements. Alternatively, you can reuse the same form to display the source of a method. For details, see "Reusing the Form Displaying Method Source", elsewhere in this chapter.

If you add multiple methods from the definition dialog, a free-standing editor pane is opened only for the last method that you added.

Multiple update actions associated with a method while the source has been modified but not saved retain the lock until all updated copies of the method source are saved, compiled, or discarded. In addition, if you attempt to edit another copy of the same method in the same Jade environment when the method has already been changed but is unsaved in another window, the following message is displayed in the message box.

```
This method has been already been modified by you and is unsaved in another
window.
```

Click:

- **Yes**, to proceed with the change

  The source is changed, and you can continue editing the method. Note, however, that the other changed method source will *not* be updated and you must handle the fact that the source has been changed and is different.

Note If the method is saved or compiled, the status line of the window with the other unsaved version of the method displays *Method source was changed in another window - the displayed changed source is out of date*.

- **No**, to activate that window and discard the change

  Focus moves to the other window in which the method has been modified.

- **Cancel**, to discard the change

  Focus then returns to the current window.

Reusing the Form Displaying Method Source

You can specify whether you want to reuse the same form to display the source of a method, when possible, rather than creating a new form for each such request. The option to reuse the same method source window is used when:

- A method source window is restored when the Jade Platform development environment is initiated.
- You double-click an **initialize** or **finalize** method entry on the Application Browser.
- You double-click on a method in a Class Browser.
- You double-click on a method in the Methods List of a Class Browser.
- You double-click on a method displayed in the methods list of the Method Status List Browser or you double-click a method entry while holding down the Shift key.
- You press F12 when in a method source window.
- You press F11 when a method identifier is selected in a method.

  The bubble help includes the creation timestamp and patch version of the entity that F11 selected.
- You double-click on an entry in a breakpoint list browser.
- You double-click on a method in a Library list browser.
- You copy a method using the Methods menu **Copy** command.
- You add a new method while the method source window has focus.

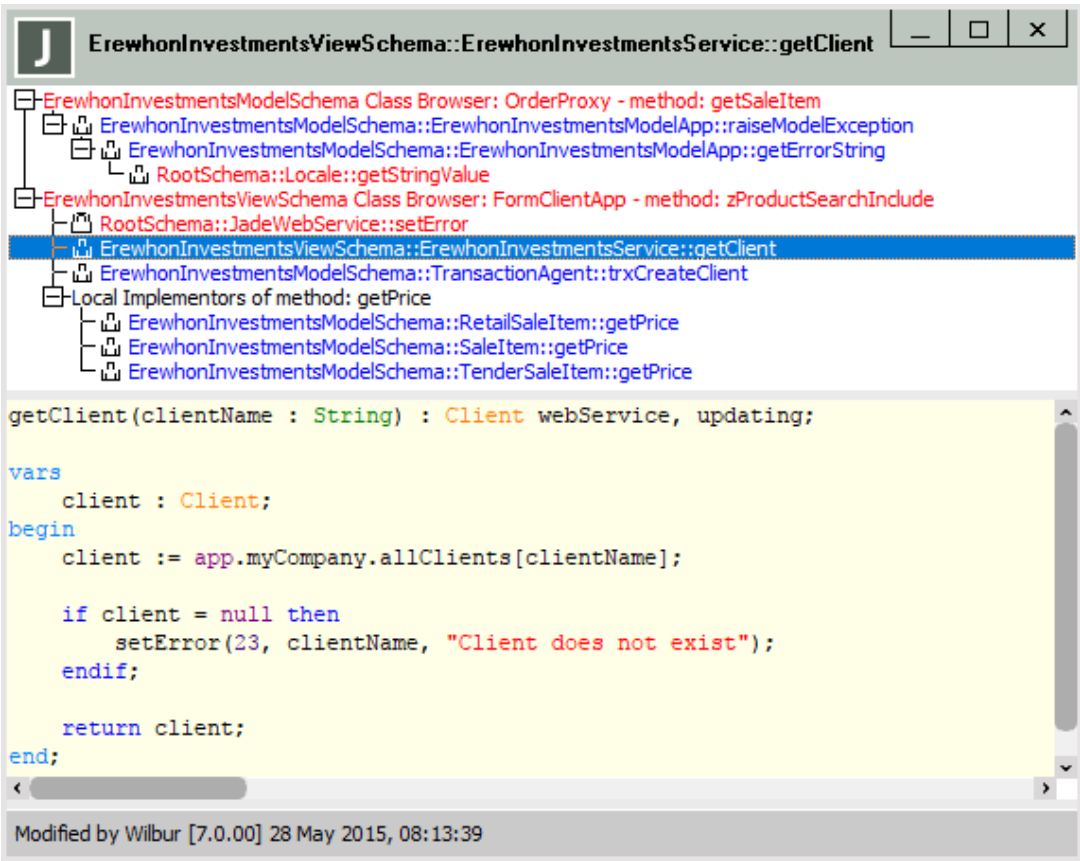
» To reuse the same form to display method source, perform one of the following actions

- Check the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet of the Preferences dialog, documented in "Maintaining Source Management Options", in Chapter 2.
- Select the **Use Same Method Source Window For All** command in the View menu, to toggle the reuse of the method source window.

  Note As this command controls only any request made from the current editor pane, you can use it to determine whether the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet of the Preferences dialog is checked.

From version 2016.0.01, when the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet of the Preferences dialog is checked, pressing F11 on a method name in the editor pane logic adds the list of local implementors of that method to the methods list in the History List of the Reused Method Source form instead of displaying a separate window to display the list of implementors. A parent entry of those methods called **Local Implementors of Method: *method-name*** is added to that list box.

When you have specified that you want to reuse the same form to display method source and you press F12 in the editor pane, Jade initially creates a new form to display the method source, and a list box that displays the history of how you got to that form. Each subsequent request to show another method uses the same window and adds the additional navigational history to the list box, as shown in the following example.



Click on an entry in the list box to display the method source for that entry, unless the currently displayed method has been changed, in which case the Save Method message box prompts you to save and compile your changes, to discard them, or continue editing the method. Conversely, if the method source in a shared window has been changed, when the new method source request is made, a message box is displayed informing you that the window cannot be shared (until the changes are saved or restored), and asking whether you want a new method source window to be opened. You can then cancel the request or click **OK** to open another window.

Note If the current method has been changed and you press F12, the development environment prompts you to save and compile your changes, to discard them, or to continue editing the method before the new method source window is displayed. This avoids two changed separate copies of the same method being open, with potentially different sources, leading to confusing results when moving from form to form.

If you close a form that is displayed in the method source browsing history list box, that entry is removed from the list box when the form is next activated. Similarly, if a method that is displayed in the method source browsing history list box is deleted, that entry is removed from the list box.

The method source browsing history list box entries are updated to use the current form caption when the form is activated.

Notes The History List can contain both versioned and non-versioned methods. The appropriate title and skin are displayed when each method is made current.

Only the currently selected method is restored after your work session is closed and then restarted and the option to restore windows is enabled. (The history is *not* restored.)

You can open a method source in a separate window that can be floated without first unchecking the **Reuse Same Method Source Window For All** check box preference, by double-clicking a method entry while holding down the Shift key in the:

- Class Hierarchy Browser, Class References Browser, Method Status List Browser, or Senders Browser (search results list), to open a separate method source window for the method regardless of the setting of the reuse option.
- History List of the Reused Method Source form, to open a separate method source window for the method and to remove the method from the History List.

If there are no methods left in the History List, the Reused Method Source form is closed. In addition, the Methods menu **Remove from List** command is enabled for the Reuse Method Source window when a method entry is selected in the History List. Clicking that command removes the selected method from the History List. If there are no methods left in the History List, the Reused Method Source form is closed.

» **To reuse the same method source form for each browser origin, perform one of the following actions**

- Check the **Reuse Same Method Source Window For Each Origin** check box on the **Source Management** sheet of the Preferences dialog, documented in "Maintaining Source Management Options", in Chapter 2.
- Select the **Use Same Method Source Window For Each Origin** command in the View menu, to toggle the reuse of the method source window.

  Note As this command controls only any request made from the current editor pane, you can use it to determine whether the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet of the Preferences dialog is checked.

This results in one History List of the Reused Method Source form for each browser origin; for example, pressing F11 on a method name in a method in the editor pane opens an extended source window that is populated only with actions in the original browser or in the displayed extended source window (the History List of the Reused Method Source form). Performing actions that cause use of a source window in another browser opens a new extended source window rather than using any other open source window not initiated by that browser.

Note If the **Reuse Same Method Source Window For Each Origin** value is checked and the method about to be displayed is open in another window and has been modified, the **Reuse Same Method Source Window For Each Origin** option is ignored and that window will be opened instead.

Clicking the *form* entry in the methods list in the History List of the Reused Method Source form returns you to the originating form. (As the original method can be modified, displaying another copy of the method source is not desirable.)

Applies to Version: 2016.0.01 and higher

Tips for Using the Editor Pane

You can obtain a description of any identifier in a method, including the creation timestamp and patch version of the entity, by simply positioning the caret within that identifier and then pressing F11. If the identifier:

- Represents a method, the source of that method is then displayed in a separate editor window. For details about using the *same* source window, see "Reusing the Form Displaying Method Source".
- Is a local variable, interface, property, class, or constant, a dialog is then displayed, providing a brief description of the selected object.

When details of a property that has a mapping method are displayed, press F12 to display a freestanding editor pane containing the mapping method source for that property.

If you position the caret on a Jade instruction, primitive type, or a system class, method, or property and then press F1, online help for the current item is displayed. The help text provides the appropriate description, syntax or signature, and often a usage example.

Note You can create multiple carets and select multiple (non-contiguous) ranges in the editor pane. For details, see "Editing Multiple Selections and Rectangular Selections", in Chapter 2.

You can make a rectangular selection, by pressing:

- Alt+left-click+dragging the mouse
- Shift+Alt+arrow keys

You can copy the rectangle selection area to the Windows clipboard or to the Jade editor clipboard. For details about copying and pasting rectangular selections, see "Editing Multiple Selections and Rectangular Selections", in Chapter 2.You can also use this functionality to add tab characters into multiple lines, by pressing Shift+Alt+↓ to select the lines (without selecting any characters) into which the tab characters are added and then pressing Tab to enter tabs on each of the selected lines. Similarly, to remove tabs, press Shift+Alt+↓ to select the lines (without selecting any characters) from which the tabs are removed and then press Backspace, to move text back to the prior tab position.

If no text is selected and the cursor is over an identifier, all matching occurrences of that identifier (full-word and case-sensitive) are highlighted. The current selection is displayed using the additional selection color (that is, the **Selection background** color defined on the **Editor** sheet of the Preferences dialog).

From version 2018.0.01, Jade highlights parentheses (**( )**) and bracket (**[ ]**) pairs when editing a Jade method or schema file when the cursor is positioned before the starting or closing **( )** parenthesis or **[ ]** bracket character, making it quicker to resolve the omission of the closing parenthesis or bracket. The back ground of the two parentheses (**( )**) and bracket (**[ ]**) bracket characters is colored using the **Additional Selections** value specified on the **Editor** sheet of the Preferences dialog. The default value is bright green. If the cursor is positioned before a parenthesis or bracket character that does not have a matching starting or closing bracket, the background of the character is highlighted in red. (There is no user preference available for this.)

When text is selected, the result is determined by the value of the **Only highlight whole words matching selection** check box on the **Editor Options** sheet of the Preferences dialog. If the check box is:

- Unchecked (the default value), any other text matching the case-sensitive selection is also highlighted using the **Additional Selections** color specified on the **Editor** sheet of the Preferences dialog (lime green, by default), unless the selection contains only *space-type* characters; for example, selecting the word **to** highlights any occurrence of **to** in the editor pane.
- Checked, any selection causes other text matching text in the editor pane to be highlighted if the selection is a full-word identifier and there are other occurrences of that word in the editor pane. This match is case-sensitive and full-word; for example, selecting the word **caption** highlights any other occurrence of the full word **caption**.

If you press F12 when the caret is positioned anywhere in the editor pane for a method, a new freestanding editor window containing all of the code for the current method is then displayed. In addition, you can obtain further assistance by pressing:

- Ctrl+1, Ctrl+2, or Ctrl+3 when the caret is positioned after an opening bracket ([) operator immediately after a valid object type, the comma-separated keys of that class are inserted after the caret position if the class is a **Dictionary** class or subclass, the text **indx]** is inserted if the class is an **Array** class or subclass, or the text **startPos : forLength]** is inserted if the class name is a **String** or **Binary** primitive type.
- Ctrl+1, Ctrl+2, or Ctrl+3 when the caret is positioned after an opening parenthesis symbol (**(**) in the signature of a class method that is automatically generated when an interface method is defined, the correct signature for the mapped interface method is then inserted following the change of signature in the interface method causing the mapped class method to be in error.
- Ctrl+1 when the caret is positioned at the end of a class to display a list of properties for that class or its superclasses, to enable you to select one for insertion after the caret.
- Ctrl+2 when the caret is positioned at the end of a class or primitive type to display a list of methods for that class, superclasses, or primitive type, to enable you to select one for insertion after the caret.
- Ctrl+3 when the caret is positioned at the end of a class to display a list of constants for that class, to enable you to select one for insertion after the caret.
- Ctrl+1 or Ctrl+2 when the caret is positioned at the end of a method name or immediately inside the opening parenthesis of a method call, to display the signature for that method. (This display remains positioned below the method until you press Esc, to enable you to specify the required method signature.)

  If the caret is positioned elsewhere (for example, after a comma between parameters), press Ctrl+5 to display the signature of the last method before the caret position.
- Ctrl+4 when the caret is positioned anywhere in an editor (method source) window, to display a combo box filled with the local variables and parameters of the method.

  When you select a displayed value, it is inserted into the method source after the caret position. (The combo box is not displayed if there are no local variables or parameters for the current method.)

For details, see "Using Bubble Help in the Editor Pane", in Chapter 4.

Printing Editor Text

When you select the **Print** command from a browser Methods menu, the Print Editor Buffer dialog is then displayed. The title bar of this dialog displays the name of the selected method and the class, primitive type, or interface in which it is defined.

You can select the printer to which output is directed, you can preview print output, or you can direct it to a rich text format (.**rtf**) file. In addition, you can print only the code that is selected in the editor pane or the whole method. (By default, all of the method is printed.)

To print the current method

1. In the Methods List, select the method that you want to print. If you want to print part of the method only, select the part that you want to print in the editor window.
2. Select the **Print** command from the Methods menu. The Print Editor Buffer dialog is then displayed.
3. In the Options group box, select the **All** option button to print the whole method or select the **Selected** option button to print the selected portion of the method.

4. In the Print Option group box, select the **Print Preview** option button to see a preview of your print output or select the **RTF File** option button to direct your print output to a rich text format (.**rtf**) file. By default, output is directed to your default printer.

5. Click the **Print Setup** button to change the printer to which output is directed and to change print setup details, if required. Print setup details are retained for subsequent print requests.

6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Displaying Multiple Sheets in the Editor Pane

Multiple-sheet functionality is available in the editor pane of the hierarchy browser forms in the Jade Platform development environment.

Multiple sheets in the editor pane enable you to navigate away from a method that you are currently editing (to view property or class details, for example) without losing your position in the source or being prompted to save your changes every time.

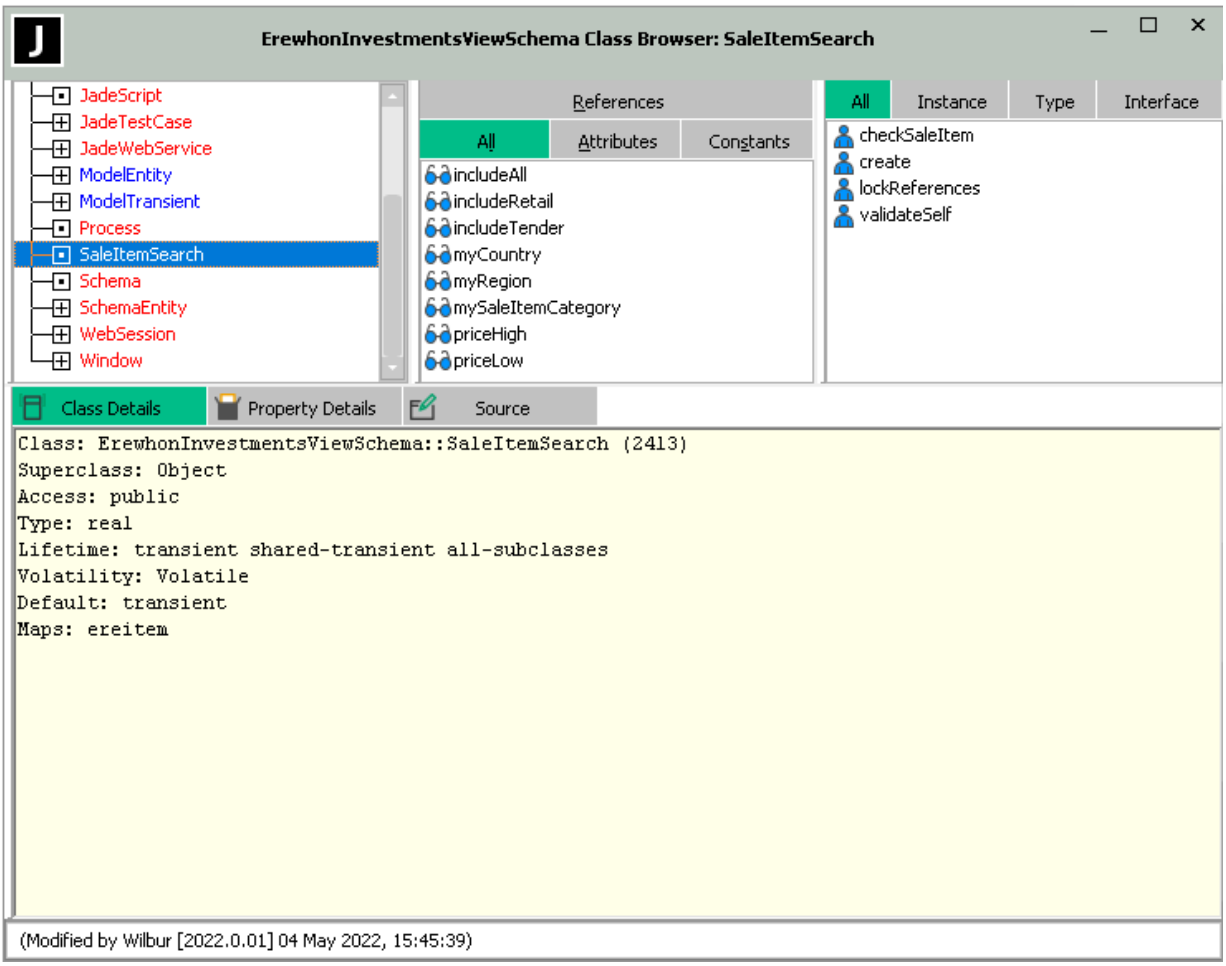
Note You are prompted to save your changes if clicking on a control or menu item property can result in the Methods List window reloading (to show control event methods, for example).

The multiple-sheet functionality is turned on, by default. If you do not want to display multiple sheets in the editor pane, use the Preferences dialog to turn off (or hide) the multiple sheets and restore the editor pane to a single pane. The Preferences dialog **Browser** sheet provides the Class Browser Options group box that controls display options for multiple-sheet functionality. For details, see "Maintaining Browser Options", later in this chapter.

The browser forms that have multiple-sheet functionality available include the following.

- Class Browser
- Interface Browser
- Primitive Types Browser
- Class References Browser
- Exposure Browser
- Export Packages Browser

An example of the Class Browser with multiple-sheet functionality enabled is shown in the following image.



Depending on the browser form type, the following tabs are displayed when multiple-sheet functionality is enabled.

- Class Details, Interface Details, Type Details, or Collection Details

  Displays details about the class selected in the Class List window, interface selected in the Interface List window, primitive type selected in the Primitive Type List window, respectively.

- Property Details or Constant Details

  Displays details about the property selected in the Properties List window or the constant selected in the Constants List window, respectively.

- Source

  Displays details about the method selected in the Methods List window.

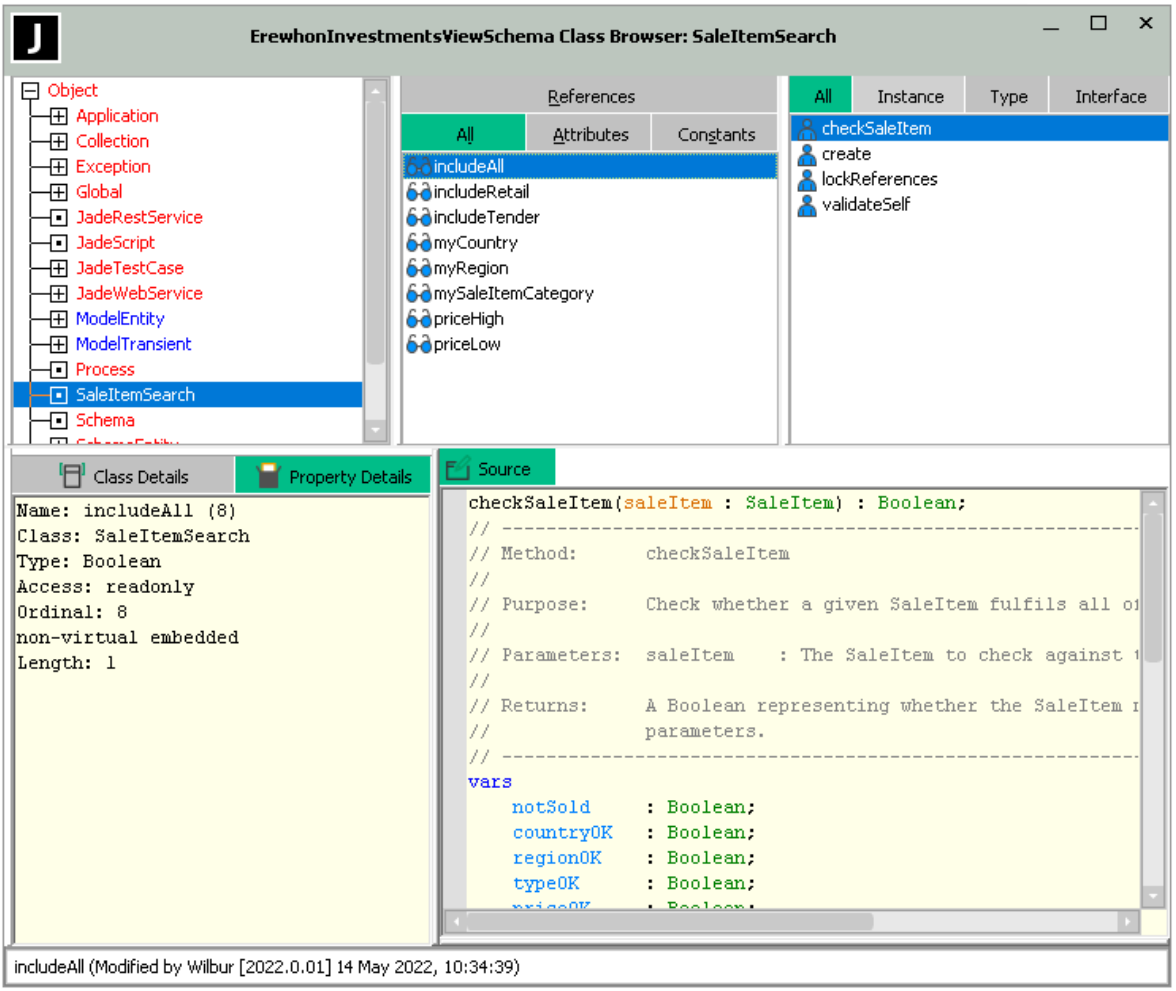
When you have multiple-sheet functionality enabled, right-clicking in the multiple-sheet tab area displays a menu that has the following commands.

- **Align Tabs Top**, which displays the tabs at the top of the editor pane. In addition, the Class Browser Options group box options are updated on the **Browser** sheet of the Preferences dialog.

- **Align Tabs Bottom**, which displays the tabs at the bottom of the editor pane. In addition, the Class Browser Options group box options are updated on the **Browser** sheet of the Preferences dialog.
- **Hide Tabs**, which turns off the multiple-pane functionality; that is, the tabs are no longer displayed. In addition, the Class Browser Options group box options are updated on the **Browser** sheet of the Preferences dialog.
- **Combine Panes** or **Separate Panes**, which enable you to toggle between combined and separate sheets.

Note You can also double-click on the tabs to toggle between combined and separate sheets.

When the sheets are combined, the **Source** sheet is separate from the two details sheets, as shown in the following image.



Applies to Version: 2022.0.01 and higher

Adding Classes to Your Schema

Although a schema inherits all classes and their associated properties, methods, and constants from its superschemas, these inherited classes are not displayed by default in the Class Browser of the subschema. Instead, for the sake of clarity, only classes local to the schema are displayed, as well as any superschema classes to which subclasses, methods, or constants have been added. For this reason, when you open the Class Browser for a newly defined schema, only the **Object** class is displayed initially. For example, after using the F4 functionality to add the **Window** class to the Class Browser (when there are no subclasses of **Window** in the current view schema), the **Window** item icon is a leaf.

You must use the F4 search functionality to add additional subclasses of the **Window** class (for example, the **Form** class or a **Control** subclass).

Only classes specifically added using F4 are displayed, unless you specifically add a subclass to the view schema.

You can immediately begin adding subclasses to the **Object** class and defining properties and methods of those classes. However, if you want to add a subclass, method, property, or constant to some other class that is defined in a superschema, you must first make that superschema class visible in the Class Browser of your new schema or schema view. For more details, see "Controlling the Use of Elements in Other Schemas", in Chapter 1.

To add a subclass to an existing class in a superschema, you may first have to find the class that you want to subclass.

Finding a Schema, Class, Interface, or Primitive Type

The **Find** command is useful when you:

- Want to locate a user-defined schema in the Schema Browser. (When searching for a schema from the Schema Browser, the RootSchema is also displayed in the **Select Required Entry** list box in the Find Schema dialog.)
- Have more classes in a schema than are displayed in the Class List window. (The search automatically locates the specified class inside a collapsed branch, if applicable.)
- Have not expanded the class branch in which the subclass is located.
- Want to locate a class in a superschema of the current class.
- Want to locate a primitive type in the Primitive Type List of the Primitive Types Browser.
- Want to locate an interface in the Interface List of the Interface Browser.
- Want to locate a class in the Schema View of the Class Browser.
- Want to locate a type by specifying only uppercase initials (for example, specifying **CA** to find the **CustomerAssociates** class.
- Want to locate a class in the current schema and subschemas by class number.

» To locate a schema, a class in the current schema and all of its superschemas, a primitive type, or an interface

1. Perform one of the following actions from the Schema, Class, Primitive Types, Interface, or Schema Views Browser.
   - Press F4.
   - Select the **Find** command from the Schema, Classes, Types, or Interfaces menu.

The Find dialog is then displayed. (For other entity types, the **Current Browser** and **New Browser** buttons are replaced by the **OK** button.)

This dialog is called the Find Schema dialog when you access it from the Schema Browser and the Find Type dialog when you access it from the Class, Primitive Types, Interface, or Schema Views Browser.

All user-defined schemas, all classes relevant to the currently selected schema (for example, classes inherited from superschemas), all primitive types, or all interfaces are displayed in the **Select Required Entry** list box. Classes and interfaces listed in green indicate a class or interface imported into the schema in a package. For details, see Chapter 8, "Using Packages", in the *Developer's Reference*.

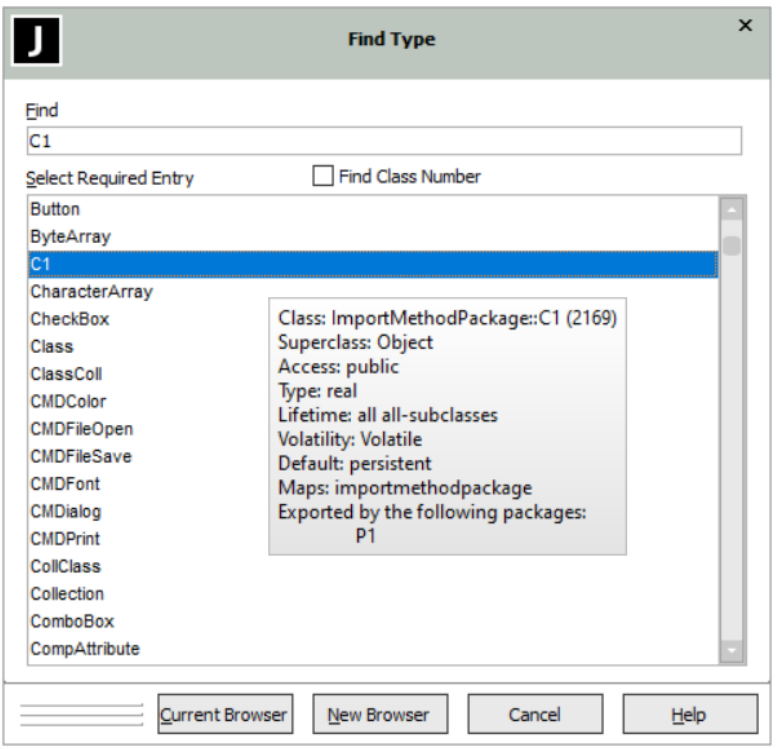
2. To select the schema that you want to locate in the Schema Browser, the class that you want to locate in the Class List of the Class Browser or Schema Views Browser, the primitive type that you want to locate in the Primitive Type List of the Primitive Types Browser, or the interface that you want to locate in the Interface List of the Interface Browser, perform one of the following actions.

   - Select the appropriate schema, class, primitive type, or interface from the **Select Required Entry** list box. The schemas, classes, primitive types, or interfaces in this list box are displayed in alphabetical order.
   - Type at least the first few characters of the schema, class, primitive type, or interface name in the **Find** text box. The list display starts with the schema, class, primitive type, or interface that matches your specified value. For example, if you have classes named **ProcessStackArray**, **ProdMaint**, **Product**, and **ProductDict**, if you enter **pro** in the **Find** text box, **ProcessStackArray** is then selected in the **Select Required Entry** list box. However, if you enter **produ**, the **Product** class is then selected.
   - When you enter text into the **Find** text box, only those entities that contain the specified text somewhere in the type name of the entry are made visible in the **Select Required Entry** list box. For example, if you specify **cust** when searching for a class, only those classes that contain **cust** somewhere in the class name are displayed. (The selection is case-insensitive.)

     The first entry that begins with the specified text is selected. If no entry starts with the specified text, no entry is selected. Use the up and down arrow keys to select and move through the entries in the list.

     The **Select Required Entry** list box behaves the same as the **AutoComplete** list box. For details, see "AutoComplete List Box Behavior", in Chapter 2. (This behavior also applies when searching for exported package features, global constants, interfaces, schemas, and the partial extract dialog.)

Hovering over any entry in the list box with the mouse displays the text returned by the **display** method for the associated object in bubble help, as shown in the following image.



- Specify the class or interface number in the **Find** text box. The number of the class selected in the Class List of the Class Browser or interface selected in the Interface List of the Interface Browser is displayed in parentheses in the first line of bubble help or the first line in the editor pane (for example, *Class: Graph (2291)*, where **2291** is the class number).
- Specify uppercase initials of the type; for example, **JWSC** to find the **JadeWebServiceConsumer** class. The following rules apply.
  - All text entered in the **Find** text box must be uppercase.
  - The first character that you enter must match the first character of a type to be listed in the **Select Required Entry** list box.
  - All other entered characters must match the uppercase characters in the name in the **Select Required Entry** list box.
  - Any entries that contain the specified text (case-insensitive) are also displayed in the **Select Required Entry** list box.

  Note When you click the **Current Browser** or **New Browser** button, the **Select Required Entry** list box lists only entries that match the specified search text.

3. If you want to specify a class number to search for the associated class in the current schema and all superschemas so that specifying numeric values does not affect the search, check the **Find Class Number** check box. The check box is not visible when performing a search in some hierarchy browser forms such as the Schema Browser and the Global Constants Browser.

This check box is unchecked, by default. When the check box is unchecked, specifying text in the **Find** text box filters the list of displayed class names, so that it includes the situation where you specify numbers only.

When you check this check box, the **Find** text box is cleared and allows only numeric digits to be specified. Pressing Enter searches the current schema and all superschemas for that class number. If that class number:

- Is found, the hierarchy browser is refreshed to select and display that class.
- Cannot be found, a message box is displayed.

4. When you have selected the required schema, primitive type, or interface, press the Enter key. Alternatively, click the **Current Browser** or **New Browser** button when you have selected the required class.

   In the Class Browser of the schema in which the class is defined, when you select a class from the list of all classes on the **Select Required Entry** list box and then click the:

   - **New Browser** button, a new Class Browser in the schema in which the class is defined is opened, with that class selected.

     Note You can browse to that schema only if the Jade development security library rules enable you to do so.

   - **Current Browser** button, that class is selected in the in the *current* Class Browser.

   In a Schema View, the Find Type dialog contains all classes relevant to the currently selected schema. If you select a class that is not in the current Schema View, a message box prompts you to confirm that you want to add the class to the Schema View when you click the **Current Browser** or **New Browser** button. When you select a class from the list of all classes on the **Select Required Entry** list box and then click the:

   - **New Browser** button, a *new* Class Browser for the same Schema View is then opened, with that class selected.
   - **Current Browser** button, that class is selected in the current Class Browser of the same Schema View.

The selected schema, class and any superclasses, primitive type, or interface are then displayed in the respective browser for the current schema. For example, if you selected the **MemberKeyDictionary** class, the **Collection**, **Btree**, and **Dictionary** superclasses are also displayed, as the **MemberKeyDictionary** class inherits some of its behavior from its superclasses. (The **List** and **Array** collection classes are not included and neither is the **Set** collection class, as the **MemberKeyDictionary** class inherits no functionality from these.)

Setting a Class as the Root Class

Use the View menu **Set Root** command from the Class Browser to set the class selected in the Class List to the root class.

To set your root class

1. Select the appropriate class in the Class List window
2. Select the **Set Root** command from the View menu

The selected class is then displayed as the root class in the Class List window; that is, the **Object** class is no longer displayed as the root class for your current schema. (The **Object** class is the default schema root class.) Alternatively, use the View menu **Reset Root** command from the Class Browser to reset the root class in the Class List window to the **Object** class.

To reset your root class to the Object class

- Select the **Reset Root** command from the View menu

The **Object** class is then displayed as the root class for the schema in the Class List.

Displaying All Superschema Classes in Your Class Browser

Although a subschema inherits all classes and primitive types and their associated properties, methods, and constants from its superschemas, these inherited classes are not displayed by default in the Class Browser of the subschema. Instead, for the sake of clarity, only classes local to the subschema are displayed, as well as classes from an imported package or any superschema classes to which subclasses, methods, or constants have been added. For this reason, when you open the Class Browser for a newly defined subschema, the **Object** class only is initially displayed.

A class that is not defined in the schema currently being viewed is displayed in the editor pane of the Class Browser with a prefix of the name of the schema in which the class is defined; for example:

```
Class: ErewhonInvestmentsViewSchema::GErewhonInvestmentsViewSchema (2257)
```

You can immediately begin adding subclasses to the **Object** class, and defining properties and methods of those classes. However, if you want to add a subclass, method, or constant to some other class, you must first make that class visible in your Class Browser window.

You can make all classes in a superschema visible by using the **Superschemas** command in the View menu, or selectively by using the **Find** command from the Classes menu. (For details, see "Finding a Schema, Class, Interface, or Primitive Type", earlier in this chapter.)

To display all superschema classes and methods

1. Select the **Superschemas** command from the View menu in the Class Browser for your schema. The View Superschemas dialog is then displayed. The superschema of the current schema is displayed in the **Show classes and methods defined in:** combo box, by default.
2. Scroll down the list box until you locate the superschema whose classes and methods you want to display in your Class Browser.
3. When the appropriate superschema is selected, click the **OK** button.

After a few seconds, the Class Browser for your schema is populated with all of the classes and methods in the specified superschema.

By default, all system (Jade-defined) classes are displayed in red, all classes inherited from a superschema are displayed in blue, user-defined classes in the current scheme are displayed in black, and classes in an imported package in green.

For details about displaying methods, conditions, properties, and constants inherited from superclasses in the same schema, see "Showing Inherited Methods, Conditions, Properties, and Constants", earlier in this chapter. See also "Controlling the Use of Elements in Other Schemas", in Chapter 1, and Chapter 8 of the *Developer's Reference*, "Using Packages".

Defining Your Own Classes

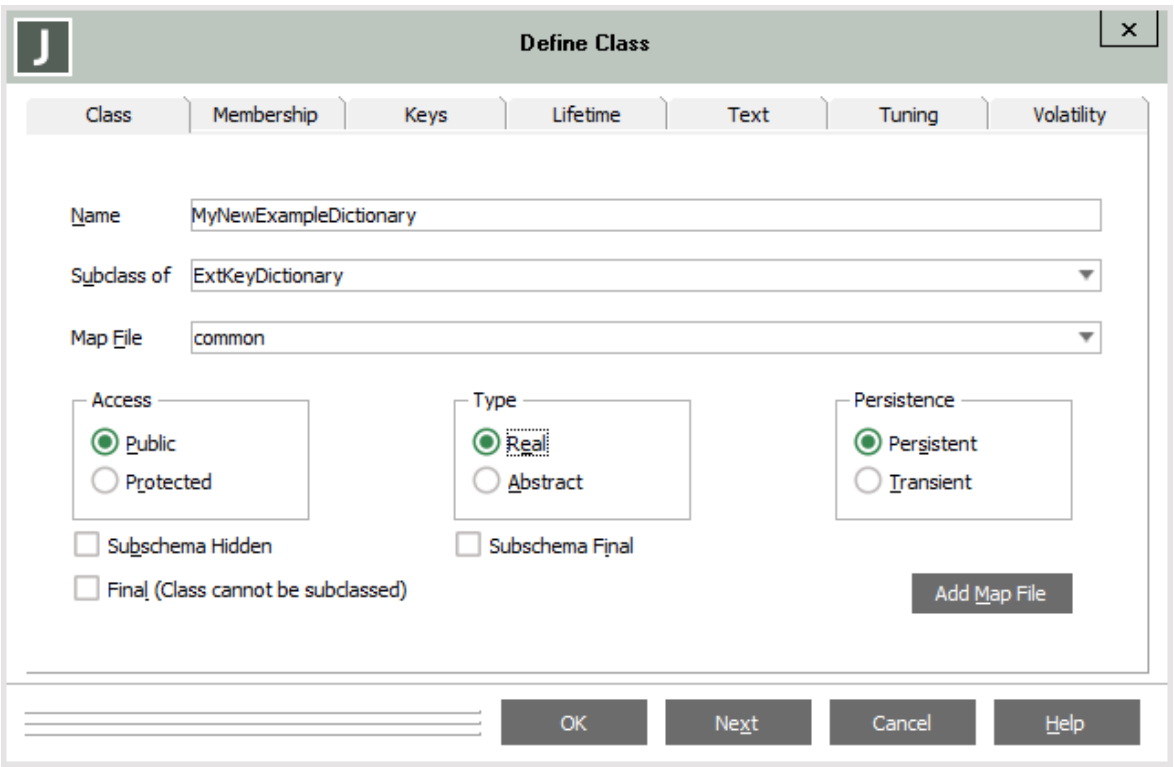
In Jade, every class must be defined as the subclass of some other class (its superclass). To define a new class, you must therefore first select an appropriate superclass (which may be the **Object** system class if there are no more-suitable candidates) and then define your new class as a subclass of your selected class.

Note A subclass inherits the behavior of its superclasses. For details about controlling how classes, methods, and properties implemented in your schemas are used by developers in other schemas, see "Controlling the Use of Elements in Other Schemas", in Chapter 1.

To add a class

1. In the Class Browser window, select the class that you intend to be the superclass of your new class. For details, see "Defining a Class", in the following subsection.
2. Select the **Add** command from the Classes menu.

The Define Class dialog, shown in the following image, is then displayed.



The Define Class dialog also contains the following sheets.

- Membership (for details, see "Defining Membership of a Collection Class", later in this section)
- Keys (for details, see "Defining Keys for a Dictionary Class", later in this section)
- Lifetime (for details, see "Specifying Class Lifetimes", later in this section)
- Options (for details, see "Defining Options for a Control Class", later in this section)
- Text (for details, see "Specifying Text for a Class", later in this section)
- Tuning (for details, see "Tuning Collection Classes", later in this section)
- Volatility (for details, see "Maintaining Class Instance Volatility", later in this section)

- Web Services (for details, see "Creating a Web Service Class" or "Creating a SOAP Header Class", in Chapter 11 of the *Developer's Reference*)

The **Membership** sheet is enabled only if you are adding a subclass to a **Collection** class or subclass.

The **Keys** and **Tuning** sheets are enabled only if you are adding a subclass to a collection **Dictionary** class, the **Options** sheet is enabled only if you are adding a subclass to a **Control** class, and the **Web Services** sheet is enabled only when you subclass the **JadeWebService** class or one of its subclasses. The **Text**, **Lifetime**, and **Volatility** sheets are always enabled.

Defining a Class

When the Define Class dialog is first opened, the **Class** sheet is displayed, to enable you to define your new class.

Note As the **Protected** option in the Access group box is reserved for future use, only public access to classes is supported; that is, selecting the **Protected** option button has no effect. Class access is public; that is, class instances can be created or deleted by another class by default. For details about controlling how classes implemented in your schemas are used by developers in other schemas, see "Controlling the Use of Elements in Other Schemas", in Chapter 1.

To define a class

1. In the **Name** text box, specify the name of the class that you want to define. The name must be unique within the schema to which it is being added.

   The class name must start with a letter, and the first four characters should not be **Jade** (which is used to prefix Jade system classes). It can include numbers and underscore characters, but cannot include punctuation or spaces. The first letter of the name is converted to an uppercase character, if it is lowercase.

2. In the class list of the **Subclass of** combo box, select the class of which you want your new class to be a subclass. All classes in the current schema are listed in alphabetical order, to enable you to select the required class. By default, the class selected in the Class List window of the Class Browser is displayed in the text box.

   Tip For details about filtering the entries displayed in the list box associated with the **Subclass of** combo box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.

3. In the map file list of the **Map File** combo box, select the database file to which instances of the class are to be stored.

   Note The **Map File** combo box is disabled when you add a subclass to a control. All control subclasses are mapped to the **_usergui.dat** file, and cannot be changed.

   For details about the system map files that store persistent objects in your Jade schemas, see "System Map Files", earlier in this chapter.

   All map files in the current schema are listed in alphabetical order, to enable you to select the required file. By default, the last map file selected is displayed in the text box. (The text box is empty if it is a new schema to which no map files have yet been added.) The map file must already exist. (Define a map file by using the **Add** command from the MapFiles menu in the Class Maps Browser.)

4. Select the **Abstract** option button in the Type group box if you do not want the class to be of **Real** type. By default, classes have a **Real** type so that an instance of a class can be created. As class instances cannot be created for abstract classes, you can use them to factor out behavior common to a number of classes; for example, the **Dictionary** subclass of the **Collection** class. (See also "Class Lifetime Options", in Chapter 1.)

Notes You can change the type of a dictionary subclass that is abstract, has at least one key, and has subclasses from **abstract** to **real** only if it does not have abstract methods.

Only changes to keys are propagated to dictionary subclasses.

5. Select the **Transient** option button in the Persistence group box if you do not want the default lifetime of instances of the class to be persistent.

   By default, the default lifetime of class instances is persistent. Instances of a transient class exist only during the application session; for example, the **File** class.

   You can create a transient instance of a persistent class and a persistent instance of a transient class. Use the **Lifetime** sheet to restrict creation of class instances. (For details, see "Specifying Class Lifetimes", later in this chapter.)

6. Check the **Subschema Hidden** check box if you want to specify that the class is available only in the local schema. This check box is unchecked by default; that is, the class is available in subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

   Classes that are subschema-hidden are not available for use in any subschemas. They cannot be referenced by subschema code, and they are not displayed in subschema Class Browsers.

7. Check the **Subschema Final** check box if you want to specify that the class can be extended or reimplemented in its local schema but not in a subschema. A class that is subschema-final has no restrictions on it in its local schema but the class cannot be subclassed in subschemas.

   Note This option does not restrict methods and constants being added to subschema copies of the class.

   This check box is unchecked by default; that is, the class can be extended and reimplemented in subschemas. For details and an example, see "subschemaFinal Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

8. Check the **Final (Class cannot be subclassed)** check box if you want to specify that the class cannot be subclassed (that is, it is *final* in the current schema) and the class is restricted in this schema and its subschemas.

   Note This option does not restrict methods and constants being added to subschema copies of the class.

   This check box is unchecked by default; that is, it cannot be subclassed. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

9. Click the **Add Map File** button if you want to add a new map file for the current schema. The File Dialog is then displayed, to enable you to define your new map file. For details, see "Adding or Changing a Map File", earlier in this chapter.

   When you have specified the new map file and clicked the **OK** button on the File Dialog, the class map file is displayed in the schema hierarchy of the Class Maps Browser and is selected in the **Map File** combo box on the **Class** sheet of the Define Class dialog.

   Notes The map file addition is an independent update that remains defined even if you cancel the class definition.

   If the **Map Files** combo box is disabled, the **Add Map File** button is also disabled. It is disabled if the class has a predefined map file or the class has instances.

10. Click the **OK** button. (Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another class.)

If you added a subclass to a **Collection** class, you must then select the **Membership** sheet. If you added a subclass to a **Dictionary** class, you must then select the **Keys** sheet.

You can select the **Lifetime** sheet if you want to restrict the lifetime of instances of the class and its subclasses, the **Tuning** sheet if you want to tune a **Collection** subclass, or the **Volatility** sheet if you want to change the volatility of instances of the class.

Notes If you want to enter free-format descriptive text for the class, select the **Text** sheet.

If a class template has been defined for all new classes in your Jade development database or for your own Jade classes, the class template is displayed below the class details in the editor pane when you click the **OK** button on the Define Class dialog. For details, see "Maintaining Text Templates", in Chapter 2.

Defining Membership of a Collection Class

The **Class** sheet of the Define Class dialog is displayed after you have specified your class options for a collection subclass and you then select the **Membership** sheet of the dialog.

To specify the membership of your collection subclass

1. In the list of classes or primitive types in the **Membership** combo box, select the class or primitive type that is to be contained in the new collection class.

   All classes and primitive types in the current schema are listed in alphabetical order, to enable you to select the required class or primitive type.

2. If you are adding an array with membership **Binary**, **String**, or **StringUtf8** primitive type, the **Length** text box and **Maximum Length** check box are enabled.

   The specified length value cannot exceed the maximum supported length for the membership type, as follows.

   - **Binary** - 16,000 bytes
   - **String** - 15,999 characters
   - **StringUtf8** - 8,000 UTF-8 characters

   Note When decreasing the length of **String**, **StringUtf8**, or **Binary** keys in dictionaries, all key values must be less than or equal to the new key length. Instances of **String** or **Binary** keys that are longer than a decreased key length are not truncated during the reorganization process, but cause the reorganization to fail and an exception to be raised.

3. If you select the **Decimal** primitive type for membership of an array class, the **Scale Factor** text box is enabled.

   When you check this check box to define the scale factor for entries in the array, values are rounded to the number of decimal places specified by the scale factor when they are added to the array.

4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another class.

If you are adding a subclass to a dictionary class, you must then select the **Keys** sheet. Alternatively, you can select the **Lifetime** sheet if you want to restrict the lifetime of instances of the class and its subclasses, the **Tuning** sheet if you want to tune a **Collection** subclass, or the **Volatility** sheet if you want to change the volatility of instances of the class.

If you are not adding a subclass to a dictionary class, the specified class is then displayed as a subclass of the selected superclass in the Class Browser window.

Defining Keys for a Dictionary Class

The **Keys** sheet of the Define Class dialog is displayed after you have specified your class and membership options for a subclass of the dictionary class and you then select the **Keys** sheet of the dialog.

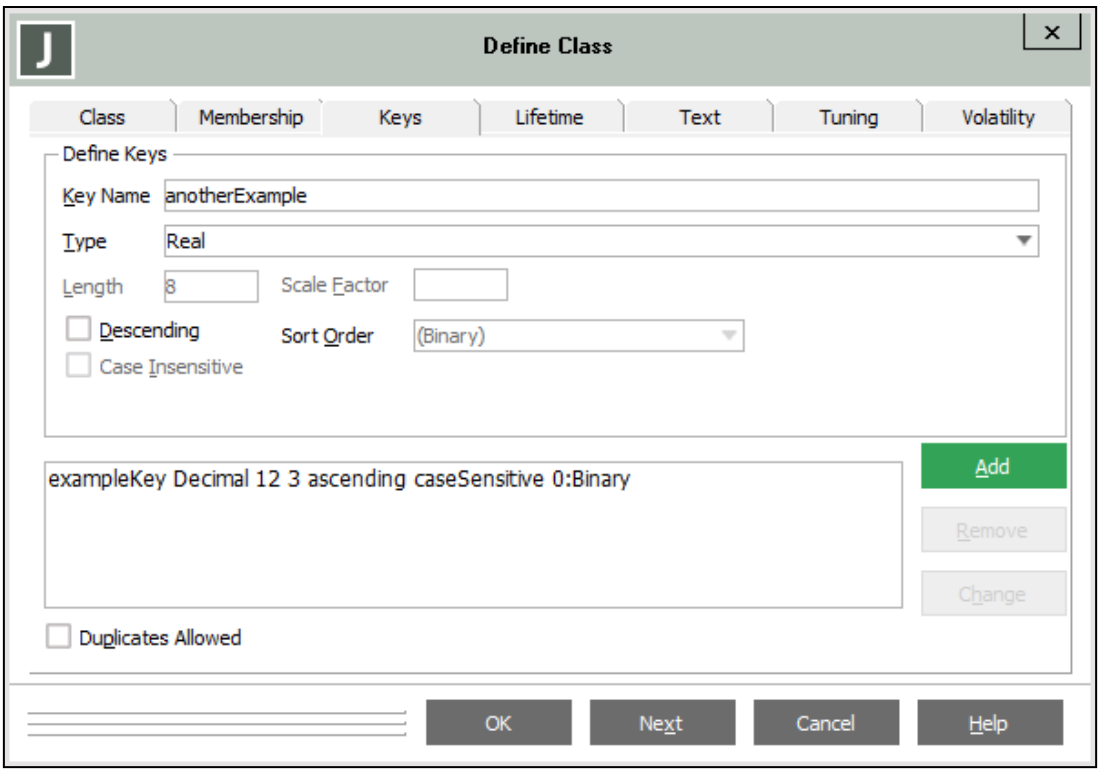
Notes You must specify your membership options before you can specify key options.

Mapped properties (that is, properties that have a mapping method) from a **RootSchema** class cannot be used as dictionary keys.

Only changes to keys are propagated to dictionary subclasses.

The **Keys** sheet that is displayed for an External Key Dictionary differs slightly to that displayed for a Member Key Dictionary. For details about key paths, see "Maintaining Dictionary Key Paths", later in this chapter.

The following image shows an example of the **Keys** sheet.



To specify the keys for a dictionary subclass

1. For a **MemberKeyDictionary** class, select the property that is to be a key of the dictionary subclass in the **Keys** list box. The listed properties are those of the class specified in the **Membership** sheet. (A property that is a key of a dictionary class is displayed in the Properties List of the Class Browser with the key icon to the left of the property.)

   Tip To reorder keys, hold down the Ctrl key while you use the mouse to drag and drop a key to a new position in the Keys List window.

   For an **ExtKeyDictionary** class, specify the required key name in the **Key Name** text box (beginning with a lowercase character) and then select the appropriate key type from the **Type** combo box.

2. If you are defining an External Key Dictionary class and you specified a **Binary** or **String** primitive type for class membership on the **Membership** sheet, specify the length of the string or binary keys in the **Length** text box, if required. The default value is **30**, but you can specify any value in the range **1** through **512**, inclusive.

   Conversely, if you selected a **Decimal** primitive type for membership of an external key dictionary in the **Type** combo box in step 1, specify the length of the decimal collection members in the **Length** text box, if required. The default value is **12**.

3. If you are defining an External Key Dictionary class and you selected a **Decimal** primitive type for class membership in the **Type** combo box in step 1, specify the scale factor of the decimal collection members in the **Scale Factor** text box, if required. The default value is zero (**0**).

4. Check the **Descending** check box if you want the selected key to be sorted in descending order. By default, keys are sorted in ascending order.

5. If you specified a **String** primitive type, select the required sort order in the **Sort Order** combo box, if required. The default sort order is binary, but you can select the sort order of any locale available in the system. You are not restricted to the locales supported by the current schema.

   Select the **(Latin1)** locale instead of **(Binary)** when you also require case-insensitivity. The **(Latin1)** locale implements the ISO 8859-1 standard, which you can use as an alternative to the **(Binary)** sort order.

   For string comparisons of key values when locales and case-sensitivity are a consideration, call the **Dictionary** class **stringKeyCompareGeq**, **stringKeyCompareGtr**, **stringKeyCompareLeq**, or **stringKeyCompareLss** method.

6. Check the **Case Insensitive** check box if the key is not dependent on specific capitalization. By default, keys are case-sensitive. This check box is enabled only if you selected a **String** primitive type in step 1.

   When you select the **(Binary)** value in the **Sort Order** combo box in step 5, the **Case Insensitive** check box is disabled (and unchecked, if it was checked).

7. Click the **Add** button to add the selected key to the new dictionary subclass. Alternatively, click the:

   - **Change** button, if you want to change the case sensitivity or the descending or ascending sort order.

     The **Change** button is enabled when you select a dictionary key, change one or more of the values that apply to that key, and then click the key attributes displayed in the list above the **Duplicates Allowed** check box.

   - **Remove** button, if you want to remove a key that you previously selected.

   Tip You can also double-click to add or remove a selected key.

8. Check the **Duplicates Allowed** check box if duplicate entries are to be permitted for the dictionary. By default, duplicate entries are not allowed.

   In a dictionary that allows duplicate keys, entries are inserted in *<key><oid>* order. Duplicate key entries therefore occur in object creation order within instances of the same class.

   If you have subclasses included in your collection, the order within the dictionary is not necessarily in strict creation order, but in creation order within the instances of each subclass.

   Regardless of the setting of this check box, you cannot add the same object to the same dictionary twice with the same key. For manually maintained member key and external key dictionaries, you can get the same object in the dictionary twice if it is added with different key values. However, this would generally be considered a logical integrity issue. Member key dictionaries with inverses cannot be set up in this way, as the inverse maintenance will automatically remove the old entry with the old key value.

9. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

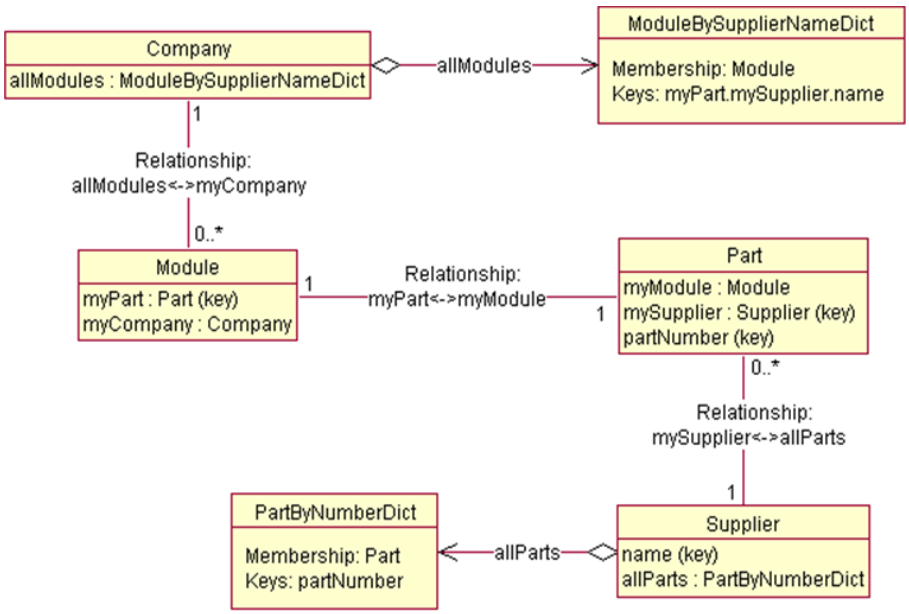
The specified class is then displayed as a subclass of the selected superclass in the Class Browser window. Alternatively, you can select the **Lifetime** sheet if you want to restrict the lifetime of instances of the class and its subclasses, the **Tuning** sheet if you want to tune a **Collection** subclass, or the **Volatility** sheet if you want to change the volatility of instances of the class.

Maintaining Dictionary Key Paths

A *key path* is a mechanism that enables you to define an automatically maintained dictionary key that is not an embedded property of the members of the dictionary, but is instead derived from the member objects.

When you define a key path, you specify a chain of references starting from the member class and finishing at an end point. At run time, the references are traversed to arrive at the end point that yields the key value.

In the example shown in the following image, **myPart**.**mySupplier**.**name** is the key path of the **ModuleBySupplierNameDict** dictionary.



In this example, each company has a dictionary of all of its modules (**allModules** of type **ModuleBySupplierNameDict**). This dictionary is keyed by the supplier name of the module's part using a key path (**myPart**.**mySupplier**.**name**).

- Each module knows:
  - The company to which it belongs, by means of **myCompany** (which is the inverse of **Company**::**allModules**).
  - The part that it contains, by means of **myPart**.

  Modules are used to add additional semantics to parts.
- Each part knows:
  - Its associated module, by means of **myModule** (which is the inverse of **Module**.**myPart**).
  - Its supplier, by means of **mySupplier**.

- Each supplier knows all of the parts that it supplies, by means of **allParts** (which is the inverse of **Part**.**mySupplier**).

Automatic Key Maintenance

Dictionary keys are automatically maintained when the dictionary has one or more explicit inverse references; that is, when it participates as an end point in a bi-directional relationship. For example, if **allModules** and **allParts** did not have their respective inverses **myCompany** and **mySupplier**, automatic key maintenance would not be performed for them.

When a dictionary property participates in a relationship (as **allModules** and **allParts** do in this example), if any of the non-key path keys are changed the dictionary is automatically updated to reflect the new key values. Automatic key maintenance is also performed for key path keys. In the example used in this discussion, this means that when a module has been added to **allModules**, if any component of the key path **myPart**.**mySupplier**.**name** is changed, **allModules** is automatically updated to reflect the new key value. This automatic key maintenance is performed only because **allModules** participates in a relationship (that is, it has **myCompany** as an inverse).

To enable Jade to automatically maintain keys for all properties of a key path, each class on which a key path property is defined must have an inverse reference back to the prior key path property. (This is required so that if any point along a key path is changed, Jade can find its way back to the related collection or collections to update key values.) An exception is raised at run time if a key path component does not have an inverse reference to its prior component.

As the deletion of objects is atomic, transactions are aborted if an exception is raised during automatic maintenance actions. If the attempt to commit a transaction after an exception occurs during a delete action (for example, user code in a **delete**, or destructor, method), the transaction is aborted if any user-defined exception handler does not do so. After the exception, no objects will have been deleted.

Using the example shown in the image, the **myPart**.**mySupplier**.**name** key path **myPart** must have the inverse **myModule** and **mySupplier** must have the inverse **allParts**. If either of these inverses were missing, the first attempt to use the **ModuleBySupplierNameDict** collection in a relationship (**allModules**, in this case) would result in a 1099 exception *(Key path component does not have an inverse to its prior component)* at run time.

Resolving Exceptions

If your runtime application encounters 1099 exceptions, the following options are available to you.

- Add the required inverse references. In the example used in this discussion, you would add the **myModule** inverse to **myPart** and the **allParts** inverse to **mySupplier**. If inverses must be added to a system that is already deployed, database reorganization may be required. Ensure that you take a full backup of your system before applying any schema modifications.
- Avoid using the key path collection in a relationship. In the example used in this discussion, **allModules** no longer participates in a relationship if you removed the **myCompany** inverse. As such, no automatic key maintenance is performed and the inverses back along the key path chain are not required. If inverses must be removed in a system that is already deployed, database reorganization may be required. Ensure that you take a full backup of your system before applying any schema modifications.

  Note Because no automatic key maintenance is performed, it is the responsibility of your application to ensure that keys are not changed after an object has been added to a key path collection. If the keys are changed, your application must properly manage the maintenance of the collection members.

- In the [JadeClient] section of the Jade initialization file on your client node and the [JadeServer] section of the Jade initialization file on your server node, add the following entry.

```
AllowKeyPathsWithoutInverses=true
```

Setting this parameter to **true** tells Jade not to raise an exception if it encounters a key path without reverse inverses and not to perform automatic key maintenance for key path properties other than the first property. (This enables your application to run without modification.)

Note With this initialization file parameter set to **true**, it is the responsibility of your application to ensure that keys are not changed after an object has been added to a key path collection. If the keys are changed, your application must properly manage the maintenance of the collection members.

For example, when a module has been added to **allModules** and you want to change a property on its key path, your application must take care of removing the module from **allModules** before the change, changing the key properties, and then adding the module to **allModules** again. Jade performs this process automatically if you elect to use automatic key path maintenance; that is, you do not set this initialization file parameter to **true**.

Setting Key Path References to Null

When using key paths, take care when setting intermediate references in the key path to null (either directly by assigning to the property or indirectly by deleting the referenced object). For example, assume that **allModules** in the example used in this discussion contains five different modules, with each module related to a different part and with each part supplied by a different supplier, as listed in the following table.

Module	myPart	myPart.mySupplier	myPart.mySupplier.name (key value)
module1	part1	supplier1	"Supplier One"
module2	part2	supplier2	"Supplier Two"
module3	part3	supplier3	"Supplier Three"
module4	part4	supplier4	"Supplier Four"
module5	part5	supplier5	"Supplier Five"

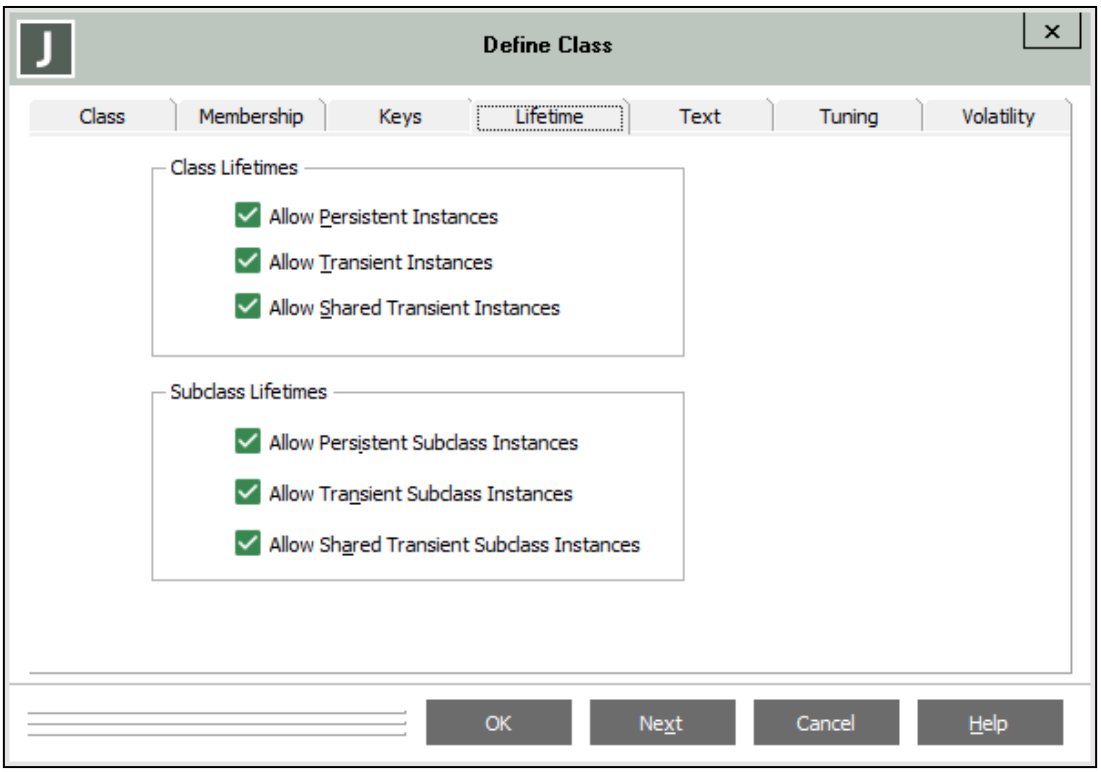
Imagine that for the first **Part** instance (**part1**), you set its supplier reference to null or you delete the supplier (which implicitly sets **mySupplier** to null because of the **Supplier**.**allParts** inverse). The **mySupplier** property is part of the key path and it is being updated, so **allModules** must automatically be updated to change the key value of the entry for **module1**. Because **myPart**.**mySupplier** is now null (meaning that **myPart**.**mySupplier**.**name** cannot be evaluated), the key value in the dictionary for **module1** is set to null.

Now imagine that you set **mySupplier** on the second **Part** instance (**part2**) to null. The same thing happens, only it now results in a duplicate key value in **allModules** because **module1** already exists with a null key. This will be a problem only if **allModules** does not allow duplicates.

Tip When working with key paths and dictionaries that do not allow-duplicates, ensure that key path references are null for one member object at a time only.

Specifying Class Lifetimes

The **Lifetime** sheet of the Define Class dialog, shown in the following image, is displayed when you select the **Lifetime** sheet of the dialog. For details about restricting the lifetime of instances of a class and its subclasses, see "Class Lifetime Options" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.



Note You cannot define a class when all three class or subclass options are unchecked (that is, at least one of the check boxes in the Class Lifetimes group box and one of the check boxes in the Subclass Lifetimes group box must be checked).

To define the lifetime of instances of the class and its subclasses

1. If you do not want the class to have persistent instances, uncheck the **Allow Persistent Instances** check box. (This check box is disabled if the **Allow Persistent Instances** check box was unchecked in the superclass to which you are adding this class.)

   When this is unchecked, an exception is raised at every attempt to create a persistent instance of the class. A compiler error is raised if you attempt to construct a persistent object from within a method (that is, you call **create <*object-name*> persistent**). Persistent instances of subclasses are restricted according to the setting of the **Allow Persistent Subclass Instances** check box, in step 4.

2. If you do not want the class to have non-shared transient instances, uncheck the **Allow Transient Instances** check box. (The **Allow Transient Instances** check box is disabled if the **Allow Transient Instances** check box was unchecked in the superclass to which you are adding this class.) When this is unchecked, an exception is raised at every attempt to create a non-shared transient instance of the class.

   A compiler error is raised if you attempt to construct a transient object from within a method (that is, you call **create <*object-name*> transient**).

Non-shared transient instances of subclasses are restricted according to the setting of the **Allow Transient Subclass Instances** check box, in step 5.

3. If you do not want the class to have shared transient instances, uncheck the **Allow Shared Transient Instances** check box. (The **Allow Shared Transient Instances** check box is disabled if the **Allow Shared Transient Instances** check box was unchecked in the superclass to which you are adding this class.)

   When this is unchecked, an exception is raised at every attempt to create a shared transient instance of the class. A compiler error is raised if you attempt to construct a shared transient object from within a method (that is, you call **create *<object-name>* sharedTransient**). Shared transient instances of subclasses are restricted according to the setting of the **Allow Shared Transient Subclass Instances** check box, in step 6.

4. Uncheck the **Allow Persistent Subclass Instances** check box if you want to prevent the creation of persistent instances of subclasses. The **Allow Persistent Instances** and **Allow Persistent Subclass Instances** check boxes will then be disabled in subclass definitions so that these values cannot be changed and an exception will be raised at every attempt to create a persistent instance of a subclass.

   When the **Allow Persistent Subclass Instances** check box is checked, the value can be changed in subclasses (that is, persistent instances and persistent subclass instances can be allowed or disallowed).

5. Uncheck the **Allow Transient Subclass Instances** check box if you want to prevent the creation of transient instances of subclasses. The **Allow Transient Instances** and **Allow Transient Subclass Instances** check boxes will then be disabled for subclass definition so that these values cannot be changed and an exception will be raised at every attempt to create a transient instance of a subclass.

   When the **Allow Transient Subclass Instances** check box is checked, the value can be changed in subclasses (that is, transient instances and transient subclass instances can be allowed or disallowed).

6. Uncheck the **Allow Shared Transient Subclass Instances** check box if you want to prevent the creation of shared transient instances in subclasses.

   The **Allow Shared Transient Instances** and **Allow Shared Transient Subclass Instances** check boxes will then be disabled for subclass definitions so that these values cannot be changed and an exception will be raised at every attempt to create a shared transient instance of a subclass.

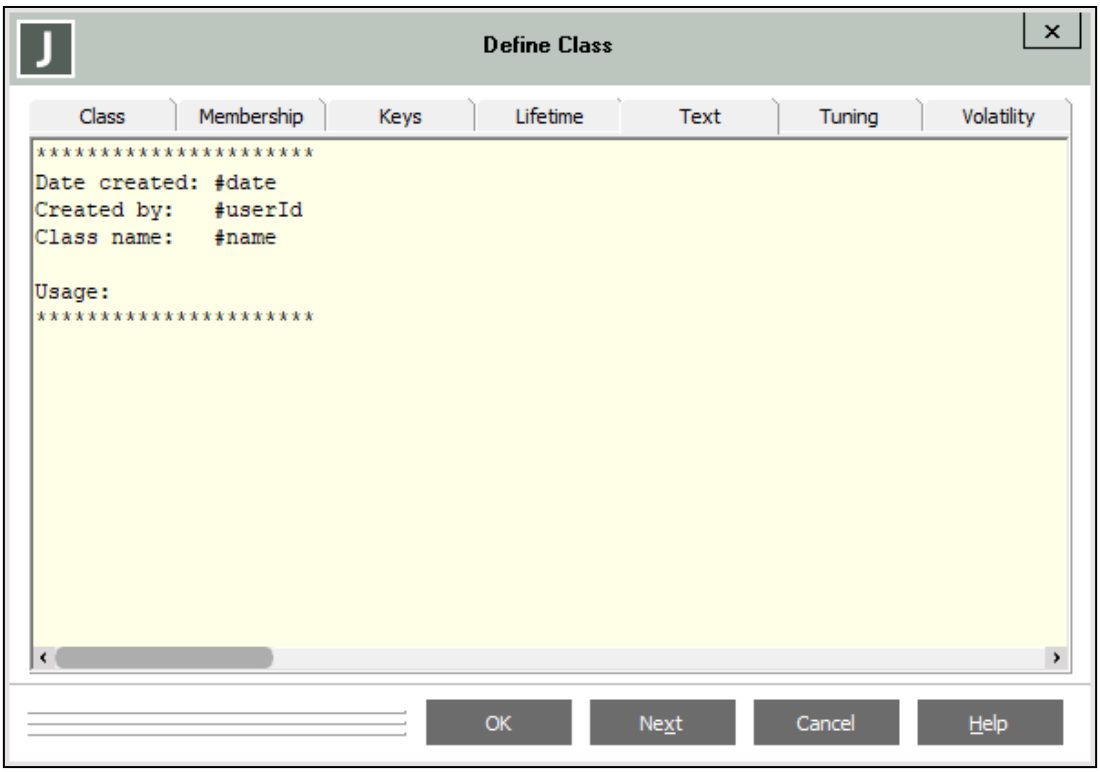
   When the **Allow Shared Transient Subclass Instances** check box is checked, the value can be changed in subclasses (that is, shared transient instances and shared transient subclass instances can be allowed or disallowed).

7. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another class.

Alternatively, you can select the **Tuning** sheet if you want to tune a **Collection** subclass or the **Volatility** sheet if you want to change the volatility of instances of the class.

Specifying Text for a Class

The **Text** sheet of the Define Class dialog, shown in the following image, is displayed when you select the **Text** sheet of the dialog.



To specify descriptive text for your class

1. In the editor pane, enter the descriptive text for your class. If you have defined a text template for your classes, this is displayed by default in the **Text** sheet. (For details, see "Maintaining Text Templates", in Chapter 2.)

   Note You can also enter supplementary text, which is for display only, by selecting the **Text** command from the Classes menu. The **Text** command accesses a free-standing window rather than a separate sheet of the Define Class dialog. For more details about entering text by using the free-standing editor window, see "Using the Free-Standing Editor Window to Define Text", later in this chapter. See also "Specifying Text for a Schema Element", later in this chapter.

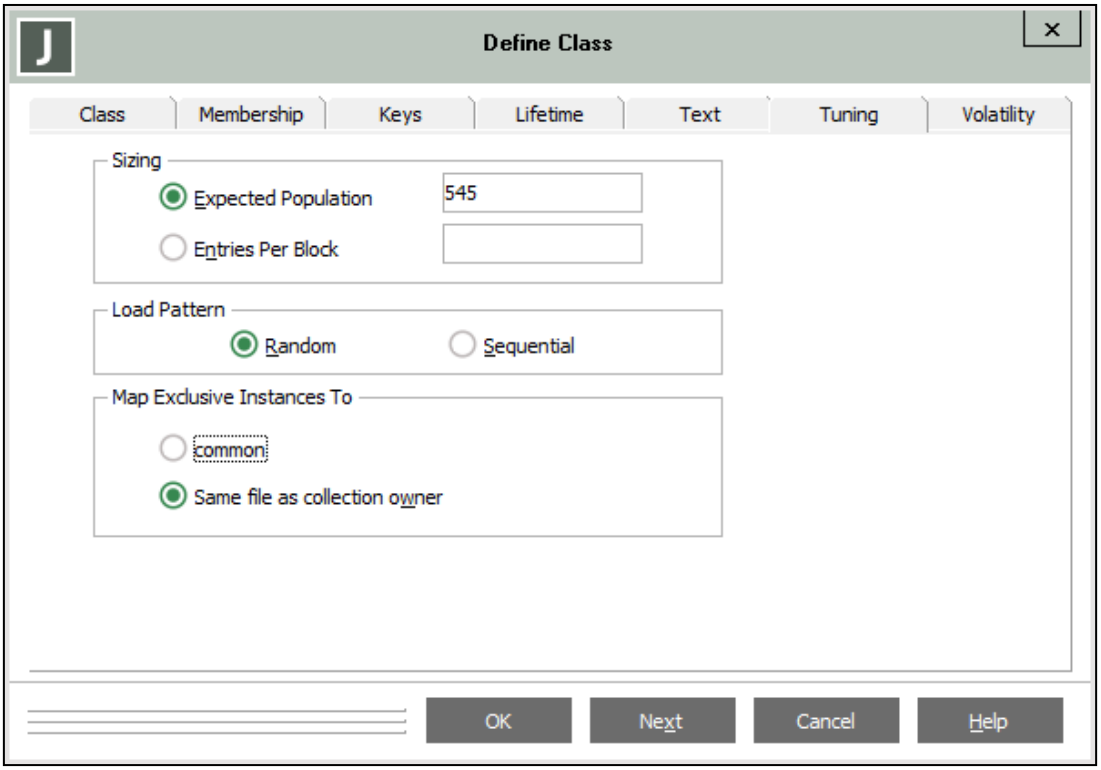
   The editor determines the behavior of the text editor window. Text is not automatically wrapped to fit the current window size. When you want text to start in the next line, press Ctrl+Enter. (If you want text to wrap in text and source windows, see "Maintaining Editor Options", in Chapter 2.)

2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your class selections or the **Next** button to define another class.

The specified class is then displayed as a subclass of the selected superclass in the Class Browser window and the descriptive text is displayed at the end of the class details in the integrated Class Browser editor pane when the class is selected in the Class List. It is also printed beneath the class name when the class is printed (by selecting the **Print Selected** command from the File menu) and the **Text** check box in the Print dialog is checked.

Tuning Collection Classes

The **Tuning** sheet of the Define Class dialog, shown in the following image, is displayed after you have specified your class options for a collection subclass and you then select the **Tuning** sheet of the dialog. (This sheet is enabled only if you are adding or maintaining a **Collection** subclass.)



This sheet enables you to specify the size of collection blocks and the collection load factor at the collection type level. For more details, see Chapter 4, "Collections Behavior and Tuning", in the *Developer's Reference*.

Note Block size tuning applies to all collection types and load factor tuning is specific to **Btree** types (that is, to sets and dictionaries).

To tune a collection

1. In the Sizing group box, select the **Entries Per Block** option button if you want to specify the size of collection blocks in terms of the number of entries in each block.

   If you specify an estimate of the expected population (that is, the expected size of the collections), Jade computes an optimal block size for you.

2. In the text box at the right of the **Expected Population** or **Entries Per Block** option button, specify the expected population of the collection or the maximum number of entries for each collection block, respectively.

   The default values of zero (**0**) indicate that Jade uses the default values for the collection type. The specification of the block size is particularly useful in Btree structures, as the block size can determine the number of levels or the height of a **Btree** required to contain a specific population (number of entries).

   Minimizing the number of levels in a **Btree** has been shown to improve the overall performance for typical use cases with large collections, by reducing both the search and update time for Btree operations.

3. The load pattern that you select depends on the key order entries, and enables Jade to determine the ratio of entries (as a percentage factor) that are moved to a new block when a **Btree** block splits.

   The Load Pattern group box is disabled for arrays, as it is not applicable. For dictionaries, the default value is **Random**. A random load pattern provides optimal loading when entries are added in random key order whereas a sequential pattern produces a higher load factor, providing a denser loading.

4. By default, instances of exclusive collections are mapped to the same file as the owner objects. (For details about browsing and maintaining class map files, see "Maintaining Map Files", earlier in this chapter.)

   The Map Exclusive Instances To group box enables the mapping of collection classes when the default **Same file as collection owner** option button is selected.

   In addition, the top option button in the Map Exclusive Instances To group box is disabled for **Collection** classes mapped to a partitionable database file, and the selection of the **Same file as collection owner** option button is enforced.

5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Maintaining Class Instance Volatility

The **Volatility** sheet of the Define Class dialog is displayed when you select the **Volatility** sheet of the dialog. This enables you to maintain the volatility of all instances of a class; that is, to optimize locking based on the frequency at which class instances are updated.

By default, class instances are volatile; that is, class instances are often updated. For details, see "Cache Concurrency", in Chapter 6 of the *Developer's Reference*.

Note The Schema Inspector displays class and object volatility by default, which enables you to check whether objects and collections are set to stable or frozen without having to write code to determine the volatility state of an object or collection.

To maintain the volatility of all instances in the class

1. In the Instance Volatility group box, select the:

   - **Stable** option button if you want to specify that class instances are updated infrequently
   - **Frozen** option button if you want to specify that class instances are not updated

   Tip Acquiring shared locks for stable objects incurs much less overhead, at the expense of extra cost when acquiring an exclusive lock. However, as stable objects are updated infrequently, exclusive locks are not often required.

   The main benefit of using frozen objects is for iterating collections. The implicit shared lock requests associated with iterating are ignored, avoiding the overhead that would otherwise be involved.

2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

You can also use the **Object** class methods summarized in the following table to define the volatility state of a specific object.

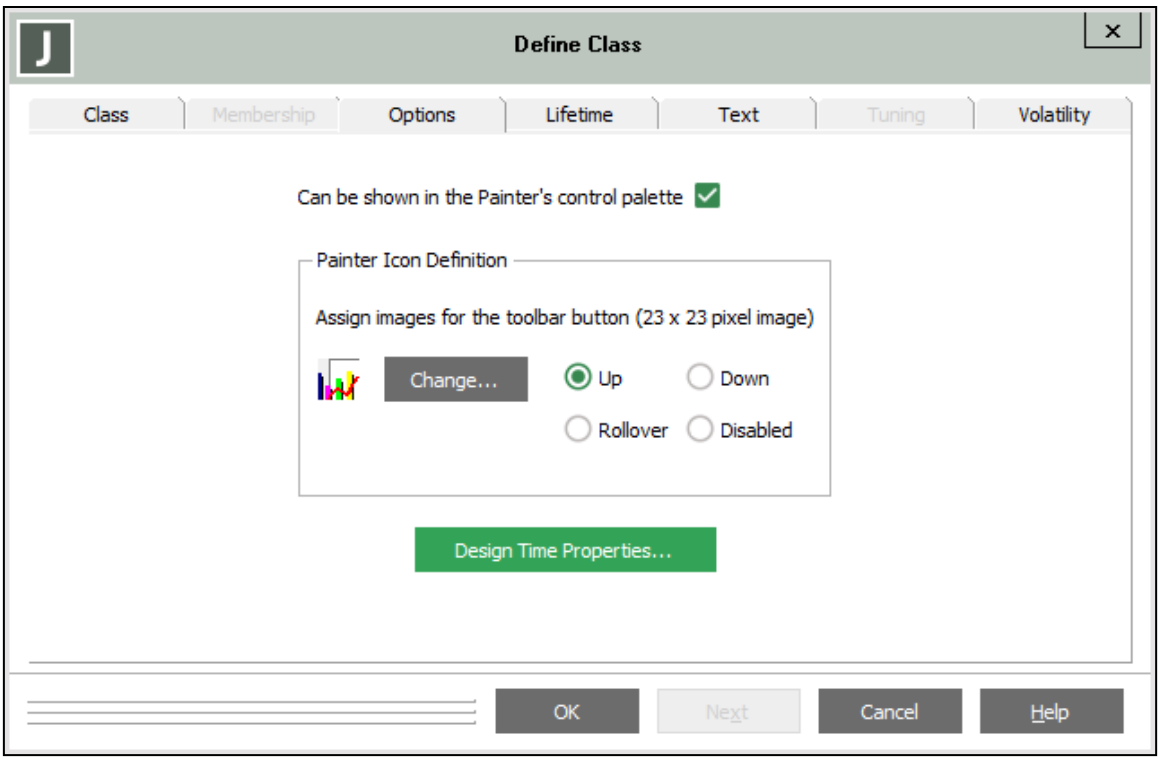
Method	Description
changeObjectVolatility	Changes the volatility of a persistent object
makeObjectFrozen	Makes the specified persistent object frozen

Method	Description
makeObjectStable	Makes the specified persistent object stable
makeObjectVolatile	Makes the specified persistent object volatile

For details, see Volume 2 of the *Encyclopaedia of Classes*.

Defining Options for a Control Class

The **Options** sheet of the Define Class dialog, shown in the following image, is displayed after you have specified your class options for a **Control** subclass in the **Class** sheet, and you then select the **Options** sheet of the dialog.



If you have defined a control image to be displayed in the Painter **Control** palette for the current class, that image is displayed in the **Options** sheet. The class options enable you to change the image of user-defined subclassed controls and to optionally display them in the Painter **Control** palette, if required.

The picture for a control subclass must be an image file supported by the **JadeMask** class **pictureMask** property; that is, a .**bmp**, .**gif**, .**ico**, .**jpg**, .**png**, or .**tif** image that is not larger than 23 pixels by 23 pixels. Smaller images are centered on controls, if required.

To define user-defined control options

1. Uncheck the **Can be shown in the Painter's control palette** check box if you do not want the image for your control displayed in the **Control** palette of the Jade Painter. By default, the image is displayed in the Painter **Control** palette; that is, this option is checked.

   If you define or change the picture for a class, it defines or changes the picture seen by all users of that control. If a picture is not specified for the **Control** class and this option is checked, a blank button is displayed in the **Control** palette of the Jade Painter (although the bubble help still indicates the type of control).

2. The Painter Icon Definition group box enables you to select the picture that is to be displayed on the tool of the Painter **Control** palette for this control for each of the four states associated with them; that is, up, down, roll over, and disabled.

   Click the **Up**, **Rollover**, **Down**, or **Disabled** option button and then click the **Change** button to specify a toolbar icon or to change an existing icon for the selected image state (which defaults to the **Up** state). The common File Open dialog (titled Select the <*state*> image file name) is then displayed, to enable you to select an existing graphics file whose image you want displayed for the **Control** subclass in the selected state.

3. Select the existing image file from the appropriate location.

4. Repeat steps 2 and 3 for each of the states associated with the image that you require.

   Tip Although you do not have to define images for each state associated with the control picture, a down state image provides users with visual confirmation that the icon has been clicked.

5. Click the **Design Time Properties** button if you want to specify the control properties that are listed in the Properties dialog during your Painter sessions. (A value assigned to a design-time property can be saved and used as the default value when a runtime instance of the control is created.)

   The Design Time Properties dialog is then displayed. For details, see "Selecting Your Design-Time Properties" under "Subclassing a Jade Control Class", in Chapter 5 of the *Developer's Reference*.

6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The specified class is then displayed as a subclass of the selected superclass in the Class Browser window.

When you have defined a class, you can then define methods, properties, or constants for that class. For details, see Chapter 4. In addition, your specified details for that class are displayed in the editor pane when the class is selected in the Class List of the Class Browser.

Displaying all References to the Selected Class

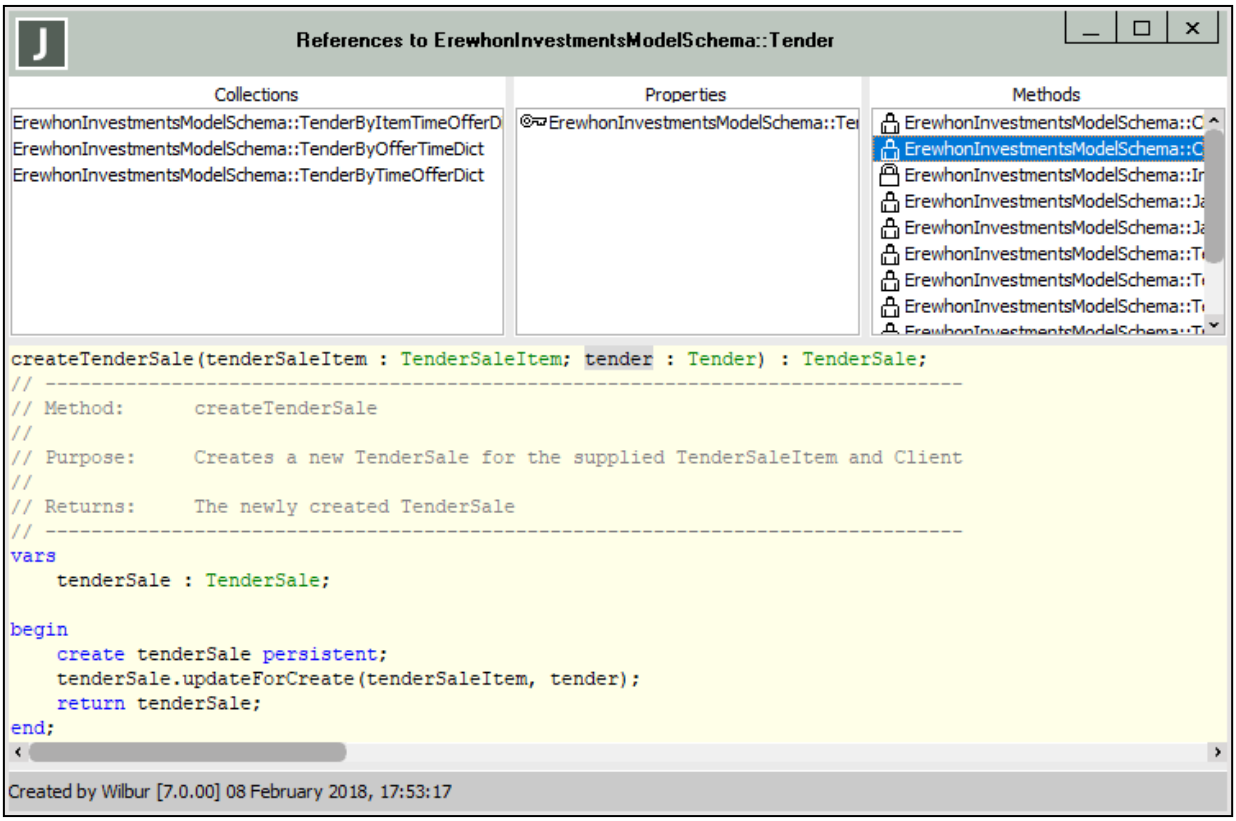
The References Browser for a selected class or primitive type lists all collections, properties, and methods that reference the selected class or primitive type, and enables you to view specific references to the class or primitive type.

For details about displaying all references to an interface, see "Displaying all References to the Selected Interface", in Chapter 14.

Using the References Browser

The **References** command from the Classes menu or Types menu to display all references in the schema to the class currently selected in the Class List of the Class Browser or the primitive type currently selected in the Primitive Type List of the Primitive Types Browser.

The References Browser for the selected class or primitive type is then displayed, as shown in the following image.



This browser lists all collections, properties, and methods that reference the selected class or primitive type, and enables you to view specific references to that class or primitive type.

To view a reference to a class or primitive type

- In the References Browser, select the collection, property, or method whose reference to the selected class or primitive type you want to view.

The details of the selected collection, property, or method are then displayed in the editor pane.

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods", in Chapter 4.

You can maintain a method displayed in the editor pane of the References Browser.

Customizing the Class Browser of the Current Schema

A *schema view* enables you to optionally customize the Class Browser of the current schema to display only the required classes in the Schema Views Class Browser.

You can select the display of classes from any schema in the hierarchy; for example, selected classes from the current schema and any of its superschemas.

Tip You can use the Class Browser of your schema view interchangeably with the default Class Browser. To quickly navigate from the default Class Browser to a Class Browser for a defined schema view, right-click on the **Browse Classes** toolbar button (if the selected schema view has been set as the current view).

For details about extracting and loading a schema view, see "Extracting a Schema View" or "Loading Your Schema", in Chapter 10.

To open a Schema Views Browser window

- Select the **Schema Views** command from the Browse menu

A Schema Views Browser window is then opened.

If you have not yet defined a schema view, nothing is displayed in the Schema Views Browser.

Note The Schema Views Browser enables you to maintain your schema views only. To view the classes in a schema view (and their associated properties and methods), you must open a Class Browser for the current schema view.

Adding a Schema View

From the Schema Views Browser, use the **Add** command from the Views menu to add a schema view to the current schema.

To add a schema view

1. Select the **Add** command from the Views menu. The Add Schema View dialog is then displayed.
2. In the **Name** text box, specify a name for your schema view.

   The schema view name must not already exist, it can contain numeric characters, but it cannot contain spaces. Jade converts the first character to uppercase, if required.
3. Click the **OK** button when you have defined your new schema view. Alternatively, click the **Cancel** button to abandon your selection.

When you click the **OK** button, the Schema Views Browser then displays your newly defined schema view.

Notes Only one Schema Views Browser for the current schema can be open at any time. If a Schema Views Browser is already open for that schema, it is brought to the top when you select the **Schema Views** command from the Browse menu.

You can have concurrent open Schema Views Browsers for different schemas in your current development work session.

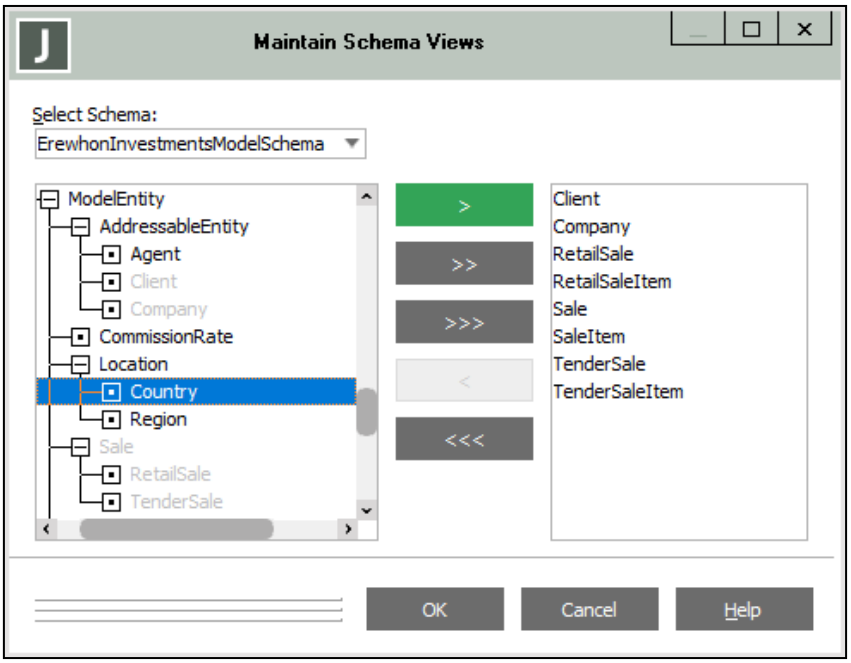
Maintaining a Schema View

From the Schema Views Browser, use the **Change** command from the Views menu to maintain the current schema view.

Tip You can resize the Maintain Schema Views dialog to enable you to display a greater number of classes, by clicking on the form border and then using the mouse to drag the form to the required position when the cursor changes shape to a double-ended arrow.

To change a schema view

1. Select the **Change** command from the Views menu. The Maintain Schema Views dialog, shown in the following image, is then displayed.



2. In the **Select Schema** combo box, specify the name of the schema whose classes you want included in your schema view or select the required name from the list.

3. In the **Class Hierarchy** list box, select the class that you want to include in your schema view and then click the **>** button to copy the selected class to the **Schema Views Class** list.

   Alternatively, you can use the Ctrl key to make multiple class selections, you can copy the selected class and subclasses by clicking the **>>** button, or you can copy all classes in the schema by clicking the **>>>** button.

   When classes have been selected for inclusion in your schema view and are displayed in the **Schema Views Class** list on the right of the dialog, they are disabled in the candidate objects list box.

Tip You can locate a specific class in the **Class Hierarchy** list box at the left of the dialog by pressing the F4 key or by right-clicking in the list box and selecting the **Find** command from the popup menu that is then displayed. The Find Type dialog is then displayed, to enable you to specify or select the class that you want to locate. All classes in the current schema and all of its superschemas are displayed in the **Select Required Entry** list box.

To select the class that you want to include in your schema view, select the appropriate class from the list box or type at least the first few characters of the class name in the **Find** text box (the list display starts with the class matching the value that you specify).

When you have selected the required class, click the **OK** button. The selected class and any superclasses are then displayed in the candidate object list box.

If the class being searched for is in the schema selected in the **Select Schema** combo box of the Maintain Schema Views dialog, the class that you searched for is then selected in the **Class Hierarchy** list box. If the class for which you are searching is in a different schema, this schema is then displayed in the **Select Schema** combo box. The **Class Hierarchy** list box on the left of the Maintain Schema Views dialog is populated with classes from the new schema. The class that you searched for is selected in the **Class Hierarchy** list box.

4. Repeat step 3 for all classes that you want to include in your schema view.
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

When you click the **OK** button, focus then returns to the Schema Views Browser.

» To remove a class from the schema view

1. In the **Schema Views Class** list box, select the class that you want to remove from your schema view and then click the **<** button to move the selected class to the **Class Hierarchy** list.

   Alternatively, you can use the Ctrl key to make multiple class selections or you can remove all classes by clicking the **<<<** button.
2. Repeat step 1 for all classes that you want to remove from your schema view.
3. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

When you click the **OK** button, focus then returns to the Schema Views Browser.

Opening a Customized Schema View

To maintain schema view classes and their associated properties and methods, you must open the Class Browser for that schema view.

» To open a schema view, perform one of the following actions

- Select the **Open** command from the Views menu
- Right-click on the Browse Classes toolbar button if the selected schema view has been set as the current view

  For details about setting schema views, see "Setting a Schema View", later in this chapter.

Removing a Class from a Schema View

From the Class Browser of a schema view, the **Remove from Schema View** command from the Classes menu enables you to remove the selected class from the schema view. (This command is visible only when the Class Browser is open for a schema view.)

Note You can remove a class from a schema view only when the class that is selected is a leaf; that is, it has no displayed subclasses in the schema view.

To remove a class from a schema view

1. In the Class Browser, select the class that you want to remove from the schema view.
2. Select the **Remove from Schema View** command from the Classes menu.

   A message box is then displayed, to enable you to confirm that the specified class is removed from the specified schema view.
3. Click the **Yes** button to confirm that the selected class is to be removed from the schema view. (The class is *not* deleted; it is removed only from the schema view.)

   Alternatively, click the **No** button to abandon the deletion.

The Class Browser of the schema view is then updated to reflect the removal of the selected class.

Setting a Schema View

From the Schema Views Browser, the **Set** command from the Views menu enables you to set the selected schema view to the current active schema view.

Note The **Set** command is enabled only when you select a schema view that is not the current schema view.

To set a schema view as the current schema

1. In the Schema Views Browser window, select the schema view that you want to be the current schema view.
2. Select the **Set** command from the Views menu.

Tip Double-click on a schema view in the Schema Views Browser to quickly set it to the current schema view.

An arrow is then displayed to the left of the selected schema view name in the Schema Views Browser window, indicating that it is the current schema view. That schema view remains the current schema view until another is set.

Defining Dynamic Clusters

The Dynamic Clusters dialog enables you to view the dynamic clusters defined for the class, add a new dynamic cluster, delete an existing dynamic cluster, change an existing dynamic cluster name, and view a list of the design-time and runtime dynamic properties defined in a dynamic cluster.

A dynamic cluster can contain both runtime and design-time dynamic properties. For details, see "Dynamic Clusters and Properties", in Chapter 4.

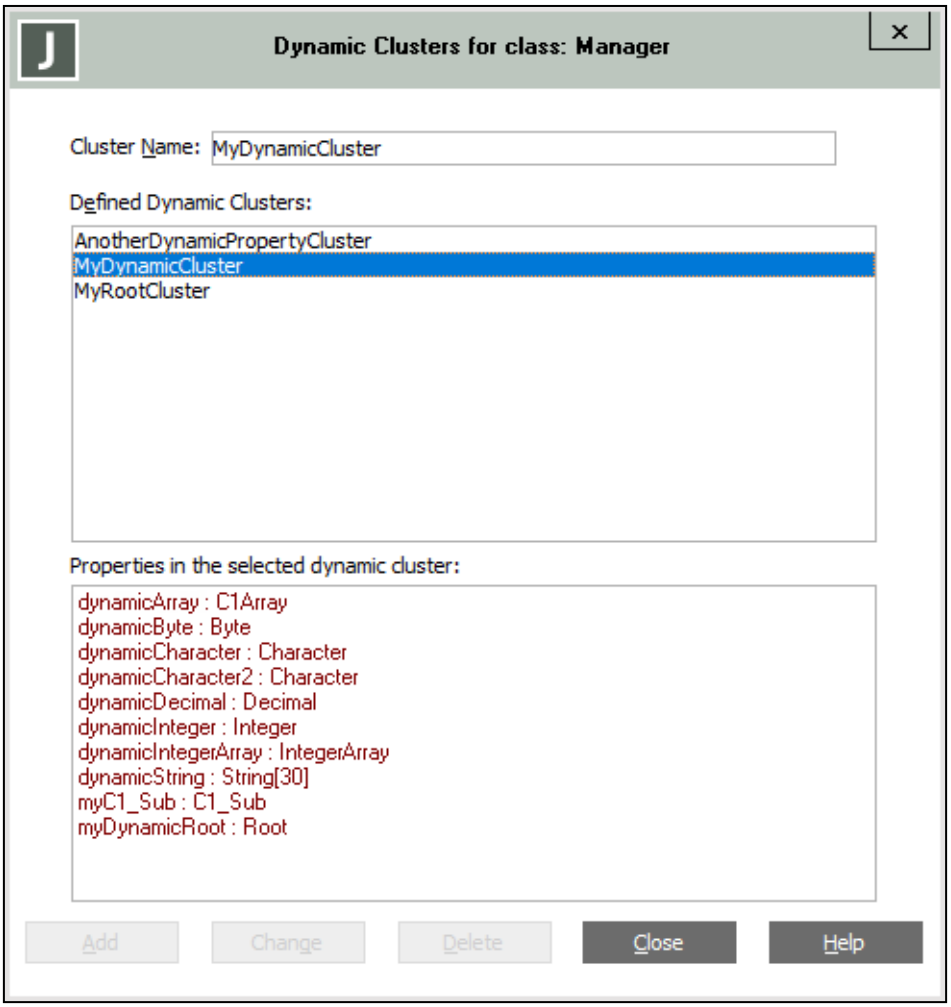
For details about the scope of design-time dynamic properties, including a table providing usage decisions, see "Design-Time Dynamic Property Use" and "Deciding Between Static and Design-Time Dynamic Properties", in Chapter 4.

To access the Dynamic Clusters dialog

1. In the Class Browser window, select the class to which you want to add a dynamic cluster or whose dynamic cluster you want to maintain.

2. Select the **Dynamic Property Clusters** command from the Classes menu. (This command is enabled only when the selected class is allowed to have dynamic clusters.)

   The Dynamic Clusters dialog for the selected class is then displayed, as shown in the following image.



The **Defined Dynamic Clusters** list box displays the dynamic clusters defined for the class and the **Properties in the selected dynamic cluster** list box displays a list of the dynamic properties defined in the cluster selected in the **Defined Dynamic Clusters** list box. By default, runtime dynamic properties are listed in magenta and design-time dynamic properties listed in maroon.

To add a dynamic cluster

1. In the **Cluster Name** text box, specify the name of the dynamic cluster. When defining or maintaining a dynamic cluster:

   - The name cannot be longer than 100 characters, cannot contain spaces, and must begin with a lowercase or an uppercase alphabetic character.
   - The name must be unique within the hierarchy of the class in which they are defined; that is, they cannot be used in a superclass or a subclass of the class, including in any superschemas or subschemas.
   - A class can have a maximum of 255 local dynamic property cluster definitions.

- A class can have a maximum of 65,535 local properties, both static and dynamic.
- You cannot add dynamic property clusters to system schema classes.

2. Click the **Add** button.

You can now define design-time and runtime dynamic properties for the cluster.

To change a dynamic cluster name

1. In the **Defined Dynamic Clusters** list box, select the dynamic cluster that you want to rename.
2. In the **Cluster Name** text box, make the changes to the cluster name that you require.
3. Click the **Change** button.

To delete a dynamic cluster

1. In the **Defined Dynamic Clusters** text box, select the dynamic cluster that you want to delete.

   Note You can neither delete a dynamic cluster from the Jade Platform development environment if it contains runtime properties nor delete it at run time if it contains design-time properties. A dynamic cluster cannot be deleted in the Jade Platform development environment if it has any properties defined.

2. Click the **Delete** button. As a dynamic cluster assigned to a class cannot be deleted if the class has instances, the **Delete** button is disabled if the owning class has instances, because the instances may still have data associated with deleted properties that were owned by the cluster.

Browsing Classes that Are in Use

In a non-production mode database, when a user schema class is opened by a process (application), that process is said to *acquire* the class. Acquiring a class marks the class as being in use by a process.

When a process terminates (or in the case of forms, when the last instance of a form is closed), the process *releases* the class, which indicates that the class is no longer in use by that process. This class acquisition and release is performed for all applications that access user class instances, including components of the development environment itself (for example, browser windows, Painter, the inspector, and the reorganization process).

When a class is in use, no structural changes can be made to that class or to any of its superclasses (unless those changes are made in the latest schema context).

Use the Classes In Use Browser to determine the classes that are in use by all processes, applications, and nodes in the system. (For details about browsing the current processes of a class selected in the Class Browser, see "Displaying Process Usages of a Class", later in this chapter.)

To open a Classes In Use Browser

- Select the **Classes In Use** command from the Browse menu

The Classes In Use Browser enables you to view in-use classes by:

- Class name
- Class number
- Process and its instance number

- Application
- Node

A Classes In Use Browser window is then opened. This browser contains the controls listed in the following subsections.

Schema Combo Box

The **Schema** combo box enables you to specify the user schema for which classes in use are to be displayed. By default, classes in use across all user schemas are displayed. If you select a schema from the list, only classes in use defined in that schema are displayed.

Refresh Button

The **Refresh** button immediately refreshes the browser window. Alternatively, press F5 or select the **Refresh** command from the Edit menu, to refresh the browser window.

Only Show Information for this Node Check Box

The **Only show information for this node** check box enables you to display classes in use, applications, and processes for the current node only. The name of the current node is shown in parentheses after the check box caption. By default, this control is unchecked; that is, classes in use, applications, and processes are displayed for the whole of your Jade system.

Auto Refresh Check Box

The **Auto refresh** check box enables you to specify that the browser window automatically refreshes itself at the specified interval.

To automatically refresh the browser window

1. Check the **Auto refresh** check box. The combo box at the right of this check box is then enabled.
2. In the adjacent combo box, specify the interval at which the browser window is refreshed, by selecting the appropriate number of seconds from the list. (You can select values in the range 5 through 600 seconds.)

By default, the Classes In Use Browser is not refreshed automatically (that is, the check box is unchecked and the combo box is disabled).

Include Sub Classes Check Box

The **Include Sub Classes** check box enables you to specify that all subclasses of the selected class are also checked to see if they are also in use and include them in the display if they are. This check box is displayed only when the Classes In Use Browser is accessed from the **Show Process Usages** command from the Classes menu.

To include subclasses in the browser window

- Check the **Include Sub Classes** check box.

By default, the check box is checked only when the selected class is a subclass of the **Window** class. It is unchecked by default for all other classes.

Show Classes Combo Box

The **Show classes** combo box enables you to restrict the number of classes for which information is displayed. By default, all classes matching the specified selection criteria are displayed and the combo box shows **all matching**, to indicate this.

To restrict the display to a limited number of classes

- Select a value from the combo box. Apart from the default display of all matching classes, you can select from up to 25 classes through to up to 5000 classes.

If you have selected a limited number of classes and there are more than this number of classes in use, the display indicates this in the header line. The header line displays the number of classes in use, processes, and nodes in the system, and is normally displayed in blue. However, when the number of classes in use exceeds the number being displayed, the header line is displayed in red, with a suffix indicating the number of classes that will actually be listed.

To revert to the display of all classes

- Select **all matching** from the list in the **Show classes** combo box.

Browsing Classes by Name

When the Classes In Use Browser is first opened, the hierarchy list box displays by name all classes that are in use, showing all classes sorted by class name that match your specified criteria.

To display the processes, nodes, and applications using a class

- Click on **+** sign in the collapsed node at the left of a class.

The processes, nodes, and applications that are currently using that class are then displayed. In the above example, the **Customer** class has three users, one of which is process **McSkull_1_b5** at node **McSkull_1**, running the **Accounts** application.

To collapse an expanded node

- Click the **–** sign at the left of an expanded class display.

To open the process view with this process selected

- Double-click a leaf entry.

Browsing Classes by Number

Click the **Class Number** button to view all classes sorted by class number that match your specified criteria.

To display the processes, nodes, and applications using a class

- Click on **+** sign in the collapsed node at the left of a class.

The processes, nodes, and applications that are currently using that class are then displayed.

To collapse an expanded node

- Click the **–** sign at the left of an expanded class display.

To open the process view with this process selected

- Double-click a leaf entry.

Browsing Applications

Click the **Application** button to view all processes and classes in use by applications that match your specified criteria.

To display the processes and classes in use for an application

- Click on **+** sign in the collapsed node at the left of an application.

The processes and classes that are currently in use by the selected application are then displayed.

To collapse an expanded node

- Click the **–** sign at the left of an expanded application display.

To open the process view with this process selected

- Double-click a process leaf entry.

To open the class name view with a selected class

- Double-click a class leaf entry.

Browsing Processes

Click the **Process** button to view all classes that match your specified criteria in use by each process in the Jade system.

To display all classes in use for a process

- Click on **+** sign in the collapsed node at the left of a process.

The classes that are currently in use in the selected process are then displayed.

To collapse an expanded node

- Click the **–** sign at the left of an expanded process display.

To open the class name view with a selected class

- Double-click a class leaf entry.

Browsing Nodes

Click the **Node** button to view all processes and classes in use by nodes matching your specified criteria.

To display the processes and classes in use for a node

- Click on **+** sign in the collapsed node at the left of a node entry.

The processes and classes that are currently in use in the selected node are then displayed.

To collapse an expanded node

- Click the **–** sign at the left of an expanded node display.

To open the process view with a process selected

- Double-click a process leaf entry.

To open the class name view with a selected class

- Double-click a class leaf entry.

Displaying Process Usages of a Class

You can display the current process or processes using a specific class in a non-production mode database. (When a class is in use, no structural changes can be made to that class or to any of its subclasses.)

Use the Process Usages for Class Browser to determine the current processes using a class selected in the Class Browser.

To open a Process Usages for Class Browser, perform one of the following actions

- Select the class in the Class Browser and then select the **Show Process Usages** command from the Classes menu.
- Right-click on the class in the Class Browser and then select the **Show Process Usages** command from the popup (context) menu that is displayed.

The Process Usages for Class Browser is then displayed. This browser, which displays a subset of the data displayed in the Classes In Use Browser (that is, process usages for one class only or for the selected class and its subclasses if the **Include Sub Classes** check box is checked in the Classes In Use Browser) enables you to view in-use process usages of the class by:

- Class name (the default)
- Class number
- Application
- Process and its instance number
- Node

The functionality of this browser is like that of the Classes In Use Browser. For details, see "Browsing Classes that Are in Use", earlier in this chapter.

When you have multiple Jade processes running under the same user code, each additional process that uses the same user code is separately identified by a numeric qualifier. For example, the first process found for Wilbur is identified as **Wilbur**, the second is named **Wilbur(2)**, and so on.

Displaying Subschema Copies of a Class

A class defined in a superschema might no longer be used but cannot be removed because it has methods or constants declared in one or more subschemas.

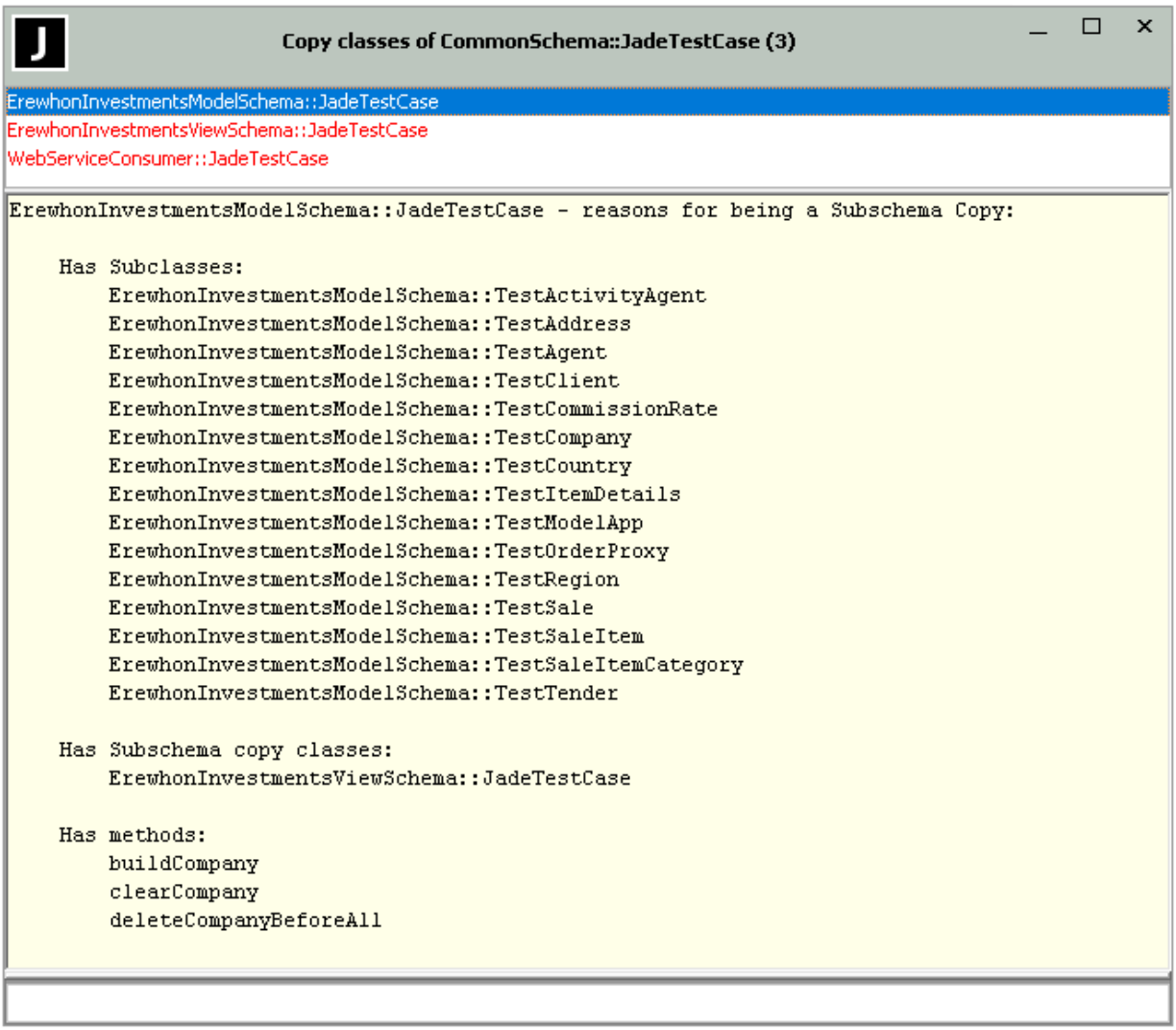
Use the Copy classes of *schema-name::class-name* browser to identify the subschema copies of a class.

To display the subschema copies of a class

- Perform one of the following actions.
  - Select the **Show Subschema Copies** command from the Classes menu in the Class Browser for your schema.
  - Right-click on the class and select the **Show Subschema Copies** command from the popup menu that is then displayed.

Note The **Show Subschema Classes** command is enabled only if the selected class has subschema copies.

The Copy classes of *schema-name::class-name* browser, shown in the following image, is then displayed.



The subschema copies of the current class are displayed in the upper area of the browser. Select a subschema copy to view more details in the lower area of the browser.

To open a Class Browser for a subschema selected in the upper area of the copy classes browser

- Right-click and select the **Open Browser For Class** command from the popup menu that is then displayed.

The Class Browser for the selected subschema and class is then displayed.

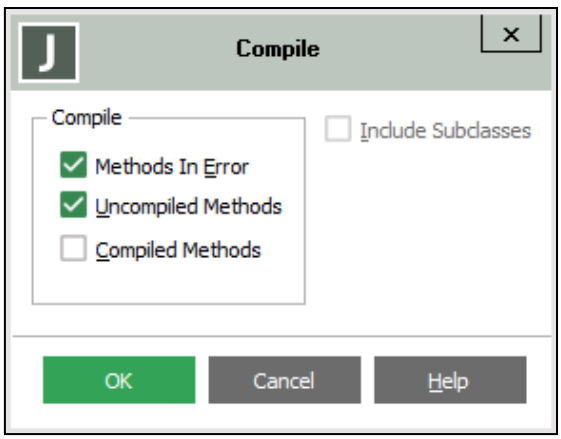
Applies to Version: 2022.0.01 and higher

Compiling All Methods Defined in a Class, Primitive Type, or Interface

You can compile all methods for the class currently selected in the Class List of the Class Browser, the primitive type currently selected in the Primitive Type List of the Primitive Types Browser, or the interface currently selected in the Interface List of the Interface Browser.

To compile methods for the current class, primitive type, or interface

1. Select the **Compile** command from the Classes menu, Types menu, or Interfaces menu. The Compile dialog, shown in the following image, is then displayed.



   By default, only methods that have been compiled are compiled again, but you can specify that you also want methods that are in error and uncompiled methods to be compiled.

2. In the Compile group box, check or uncheck the appropriate check boxes, as follows.
   - Use the **Methods In Error** check box to specify that methods that are in error in the current class, primitive type, or interface are to be compiled. Methods in error are compiled by default (that is, this check box is checked).
   - Use the **Uncompiled Methods** check box to specify that uncompiled methods in the current class, primitive type, or interface are to be compiled. Uncompiled methods are compiled by default (that is, this check box is checked).
   - Use the **Compiled Methods** check box to specify that compiled methods in the current class, primitive type, or interface are to be compiled. Compiled methods are not compiled by default (that is, this check box is unchecked).
3. If the current class contains subclasses, check the **Include Subclasses** check box if you also want methods in subclasses compiled to your selected criteria. (As this does not apply to primitive types or interfaces, this check box is disabled when you access the dialog from the **Compile** command in the Types menu or Interfaces menu.)

   Methods in subclasses of the current class are not compiled by default (that is, this check box is unchecked).
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

A Compiling progress dialog is displayed, showing the names of the methods as they are compiled.

If any method in the current class, primitive type, or interface has been saved but contains syntax errors or it has not been compiled, a message dialog is displayed when all methods in the class, primitive type, or interface have been compiled.

The name of the current class, primitive type, or interface is displayed in the dialog title bar. Click the **Yes** button if you want to view the error list or click the **No** button if you do not want to view it. If any syntax errors were detected during the compile process or a method had not been compiled and you click the **Yes** button, the Method Status List Browser is then displayed.

For details about the Method Status List Browser, see "Using the Method Status List Browser", in Chapter 4.

For details about compiling individual methods, see "Compiling Methods", in Chapter 4. See also "Compiling an Interface Method", in Chapter 14.

Searching for or Replacing an Element in a Schema

The Edit menu provides functions that enable you to locate and optionally replace an element in the editor pane of the current browser or to search for an element in all classes in the current schema. For details, see "Locating an Element in a Browser Editor Pane", "Searching for an Element in all Classes in the Current Schema", and "Searching a Browser for an Entity", later in this section. See also "Finding a Schema, Class, Interface, or Primitive Type", in Chapter 3.

Locating an Element in a Browser Editor Pane

Use the **Find/Replace** command from the Edit menu to locate text in the editor pane of the current browser and optionally replace the located text with a specified value.

The search for your specified text is started from the top of the editor pane or from the current caret position, depending on the settings in the Start From group box. See also "Finding the Next Occurrence of Specified Text", "Searching for Text in a Reverse Direction", "Locating Text on Which the Caret is Positioned", and "Searching for an Element in all Classes in the Current Schema", later in this chapter.

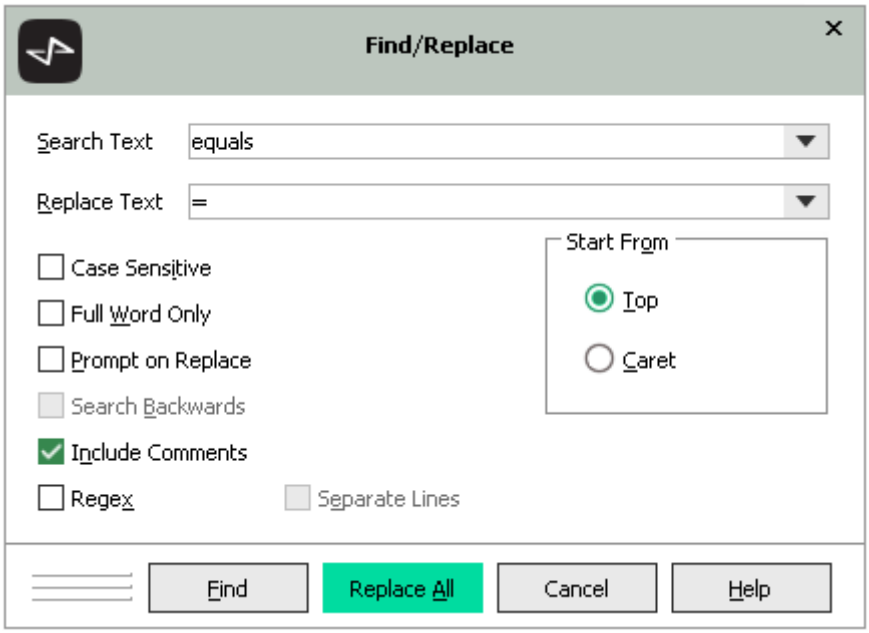
» To limit your search, perform one of the following actions

- Select the part of the editor pane contents in which the search is to be performed.
- Check the **Search Backwards** check box in the Find/Replace dialog.

To locate and optionally replace text in the editor pane

1. Select the **Find/Replace** command from the Edit menu or press Ctrl+F.

   The Find/Replace dialog, shown in the following example, is then displayed.



   The contents of the **Search Text** combo box is selected (overtype mode) when the dialog is opened. This combo box contains a list of the text of the last 50 unique searches that you performed, so that you can repeat previous searches by selecting an entry from the combo box list. This list is maintained in the user profile of your current user logon.

2. In the **Search Text** combo box, specify or select the text that you want to locate in the editor pane.

3. In the **Replace Text** combo box, specify the text that is to replace any located text that is specified, if required. When the dialog is opened, this combo box is always blank, to prevent accidental replacements with text from previous replacements.

   This combo box contains a list of the text of the last 50 unique replacements that you performed, so that you can repeat previous replacements by selecting an entry from the combo box list. This list is maintained in the user profile of your current user logon.

4. If you want the exact match by case (where uppercase or lowercase is significant), check the **Case Sensitive** check box. A search is then performed for text with the same capitalization as the text in the **Search Text** combo box. By default, searching is case-insensitive; that is, this check box is unchecked.

   You can optionally replace text that is located and matched by case with text with the same capitalization as the specified in the **Replace Text** combo box.

5. If you want only text that is an identifier in which the whole word matches to be located and replaced, check the **Full Word Only** check box. If this check box is checked and the text is not an identifier, the locate and replace actions will fail.

   By default, any identifier text that matches the string that is being searched for will be located and replaced, regardless of whether it is embedded or not in another word.

6. If you want to confirm that each occurrence of the specified search text is replaced with the specified replacement text, check the **Prompt on Replace** check box. By default, you are not prompted to confirm text replacement; that is, this check box is unchecked.

   When you select text replacement confirmation and you click the **Replace All** button, the Verify Replace dialog is displayed every time the search text is located. In the Verify Replace dialog, perform one of the following actions.

   - Click the **Yes** button to confirm that the located text is to be replaced.
   - Click the **No** button to leave the highlighted text unchanged and continue searching for the specified string.
   - Click the **Cancel** button to abandon the search and return focus to the editor pane.

7. If you want to search backwards through the contents of the editor pane from the current caret position up to the beginning of the editor pane, check the **Search Backwards** check box. The **Search Backwards** check box is enabled only when you select the **Caret** option button in the Start From group box. By default, searching is performed from the current caret position to the end of the editor pane; that is, this check box is unchecked.

   When this option is checked, you have specified your search and replacement text, and you click the **Replace All** button, the Verify Replace dialog is displayed every time the search text is located.

8. If you want the search to start from a specific position in the editor pane, select the **Caret** option button in the Start From group box. The **Search Backwards** check box is then enabled. (Alternatively, you can press Shift+F3 or select the **Find Again Reverse Direction** command from the Edit menu.)

   By default, the **Top** option button is selected, indicating that the search begins at the top of the editor pane.

9. By default, comments in the source of a method are searched for and optionally replaced, in Jade methods only. Uncheck the **Include Comments** check box if you do not want to search for and optionally replace occurrences of the search text in method comments, including line (*//*) and block (***/* */***) comments.

   The **Include Comments** check box is set to **true** (that is, checked) for the first search performed after logging on. Subsequent displays of the local Find/Replace dialog set the check box value to the prior setting that was used to perform a search or replace action.

10. If you want the search text to be interpreted as a regular expression pattern, check the **Regex** check box; for example, enter **[A-Za-z]+\d{4}** to search for a word followed by a four-digit number. By default, the **Regex** check box is unchecked. If you check the **Regex** check box, the **Separate Lines** check box is enabled.

    The following list shows examples of regular expressions that you can use in a Regex search. To search for:

    - At least a 4- or 5-digit number assignment ignoring whitespace; for example, an error code

      ```
      \s*:=\s*\d{4,5}
      ```

    - Transaction methods

      ```
      (abortTransaction|beginTransaction|commitTransaction)
      ```

    - **unloadForm** in an epilog

      ```
      epilog[\s\S]*unloadForm
      ```

    - A variable declaration named **userName**

      ```
      vars[\s\S]*\buserName\b[\s\S]*begin
      ```

11. If you want to treat the search text as if it is broken up into separate lines, check the **Separate Lines** check box; for example, if you want to search for a specified string at the start or end of a line, check this check box. When this check box is checked, the search breaks the text into individual lines at each **CrLf** end-of-line sequence. By default, the **Separate Lines** check box is unchecked.

12. To find the next occurrence of the specified text, click the **Find** button.

    If Jade finds the text string that matches your specified options, the located text is then highlighted and focus is returned to the editor pane. If Jade cannot find the text string that matches your specified options, a message dialog informs you that the search text was not found and waits for you to click the **OK** button in the message dialog before returning focus to the editor pane.

13. To replace all occurrences of the text specified in the **Search Text** combo box with the text specified in the **Replace Text** combo box, click the **Replace All** button.

    If Jade finds the text string that matches your specified options, all occurrences of the located text are then replaced with the specified replacement text and focus is returned to the editor pane. If you checked the **Prompt on Replace** check box, you are prompted to confirm that each occurrence of the located text is to be replaced.

14. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

Note The values of the **Case Sensitive** and **Full Word Only** check boxes are **false** (that is, unchecked) the first time the dialog is displayed after logging on to the Jade Platform development environment. After you perform a search, the values used in the search are saved. Those values are restored the next time the dialog is used in your current development environment session.

Finding the Next Occurrence of Specified Text

When the text specified in the Find/Replace dialog has been located and focus is returned to the editor pane, you can make further searches in the editor pane for that text.

To find the next occurrence of specified text, perform one of the following actions

- Select the **Find Again** command from the Edit menu
- Press F3

The next occurrence of that text is then located and highlighted or the Message dialog is displayed, advising you that the search text was not found if no further occurrences of that text are located.

For details about specifying the text that you want or locate, see "Locating an Element in a Browser Editor Pane", earlier in this chapter. See also "Searching for an Element in all Classes in the Current Schema", "Searching for Text in a Reverse Direction", and "Locating Text on Which the Caret is Positioned", in the following sections.

Searching for Text in a Reverse Direction

When the text specified in the Find/Replace dialog has been located and focus is returned to the editor pane, you can make further searches in the reverse direction in the editor pane for that text.

To find the next occurrence of specified text in the reverse direction, perform one of the following actions

- Select the **Find Again Reverse Direction** command from the Edit menu
- Press Shift+F3

The next occurrence in the reverse direction of that text is then located and highlighted or the Message dialog is displayed, advising you that the search text was not found if no further occurrences of that text are located.

If the search was specified as a forward search, the reverse search is performed backwards. Conversely, if the search was specified as a backwards search, the reverse search is forwards.

The Shift+F3 shortcut (Find Again Reverse) is listed in the table of **Find** category shortcuts in the **Short Cut Keys** sheet of the Preferences dialog. You can change this shortcut key to a combination of your choice.

Applies to Version: 2018.0.01 and higher

Locating Text on Which the Caret is Positioned

You can locate the next occurrence of the text on which the caret is positioned in the editor pane.

To locate text on which the caret is positioned, perform one of the following actions

- Select the **Find At Caret** command from the Edit menu
- Press Ctrl+F3

The text currently under the caret is then placed in a search string buffer and the next instance of that text in the editor pane is then highlighted.

To perform subsequent searches for located text

- Press F3

The next occurrence of the text in the search buffer is then located and highlighted or the Message dialog is displayed, advising you that the search text was not found if no further occurrences of that text are located.

For details about specifying the text that you want or locate, see "Locating an Element in a Browser Editor Pane", earlier in this chapter. See also "Searching for an Element in all Classes in the Current Schema", in the following section.

Searching for an Element in all Classes in the Current Schema

The Global Search And Replace dialog, accessed from the Edit menu when a browser has focus, enables you to locate and display all occurrences of specified text in the:

- Current class, the current class and subclasses, the current class and superclasses, or in all classes.
- Current schema, the current schema and subschemas, the current schema and superschemas, or in all schemas.

The global find functionality searches all methods that meet the specified search criteria, regardless of whether the method cannot be modified or whether it is currently being edited. Any methods currently modified and unsaved by you are also included, but the find action searches only the saved persistent source; not the unsaved source.

Note The search uses the current method source according to whether or not a delta is set. If you have a delta set, the checked-out source of a method is searched if it exists; otherwise the unchecked-out source is used.

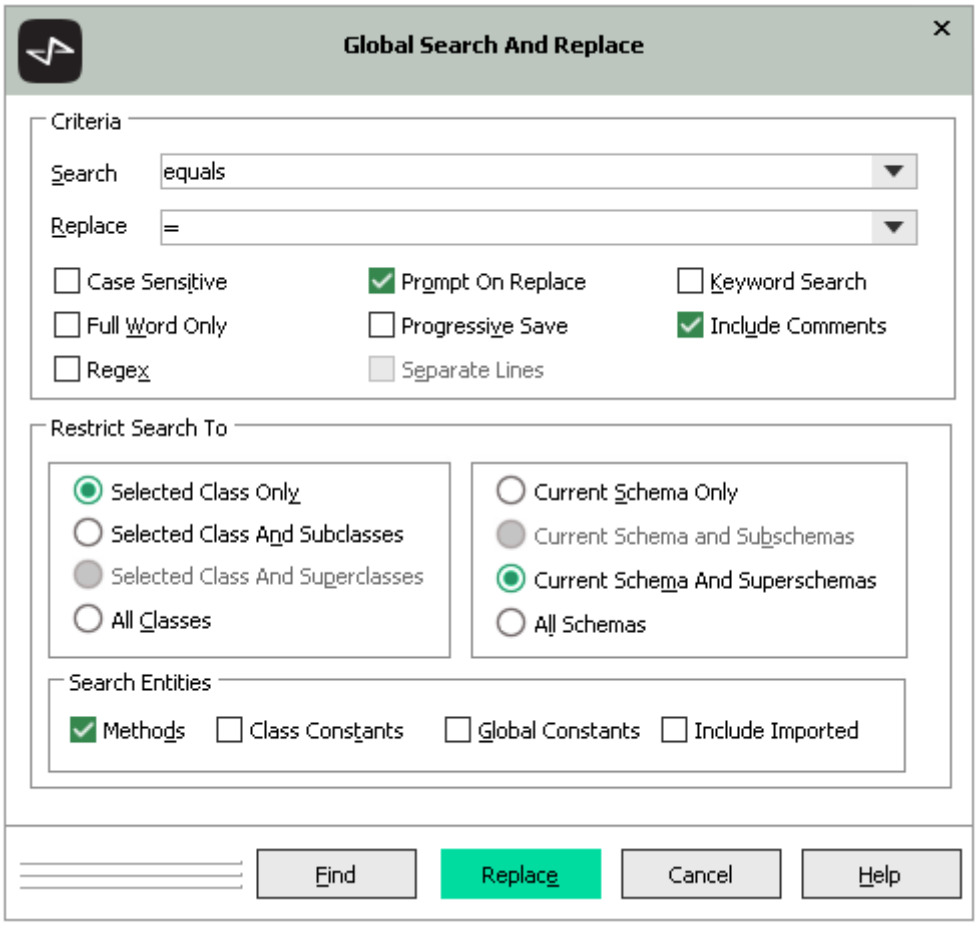
The replace functionality ignores any methods that cannot be modified, including any unsaved methods that you currently have open.

See also "Locating an Element in a Browser Editor Pane", "Finding the Next Occurrence of Specified Text", and "Locating Text on Which the Caret is Positioned", earlier in this chapter.

To locate and display all occurrences of specified text

1. Select the **Global Find/Replace** command from the Edit menu or press Shift+Ctrl+F3.

   The Global Search And Replace dialog, shown in the following image, is then displayed.



   The contents of the **Search** combo box is selected (overtype mode) when the dialog is opened. This combo box contains a list of the text of the last 50 unique searches that you performed, so that you can repeat previous searches by selecting an entry from the combo box list. This list is maintained in the user profile of your current user logon.

2. In the **Search** combo box, specify the text that you want to locate.

3. In the **Replace** combo box, specify the text that is to replace any located text that is specified, if required. When the dialog is opened, this combo box is always blank, to prevent accidental replacements with text from previous replacements.

   This combo box contains a list of the text of the last 50 unique replacements that you performed, so that you can repeat previous replacements by selecting an entry from the combo box list. This list is maintained in the user profile of your current user logon.

4. If you want the exact match by case (where uppercase or lowercase is significant), check the **Case Sensitive** check box. A search is then performed for text with the same capitalization as the text in the **Search** combo box. By default, searching is case-insensitive; that is, this check box is unchecked.

You can optionally replace text that is located and matched by case with text with the same capitalization as that specified in the **Replace** combo box.

5. If you do not want to confirm that each occurrence of the specified search text is replaced with the specified replacement text, uncheck the **Prompt On Replace** check box. You are prompted to confirm text replacement by default; that is, this check box is checked.

   When text replacement confirmation is selected and you click the **Replace** button, the Verify Replace dialog is displayed every time the search text is located, superimposed on an editor pane that displays the located text highlighted in the pane and the schema, class, and method in which the located text occurs displayed in the title bar.

   In the Verify Replace dialog, perform one of the following actions.

   - Click the **Yes** button to confirm that the located text is to be replaced.
   - Click the **No** button to leave the highlighted text unchanged and continue searching for the specified string.
   - Click the **Cancel** button to abandon the search and return focus to the editor pane.

6. If you want to search for a Jade instruction, or keyword, check the **Keyword Search** check box. A search is then performed for the specified Jade instruction. By default, searching is case-insensitive; that is, this check box is unchecked.

   The **Replace** button is disabled when a keyword search is performed.

   Note As Jade instructions are located only in methods that have been compiled, no search results are displayed for any method that matches your selection criteria but is uncompiled.

7. If you want to locate only text in which the whole word matches, check the **Full Word Only** check box. A search is then performed for text that exactly matches the text in the **Search** combo box. By default, any text that matches the string that is being searched for will be located, regardless of whether it is embedded or not in another word; that is, this check box is unchecked.

   You can optionally replace the full word that is located with text with specified in the **Replace** combo box.

8. If you want each method in which the text is located and replaced to be saved as the replacement is made, check the **Progressive Save** check box. By default, all methods in which replacements are made are saved, or committed, after all replacements have been made; that is, this check box is unchecked.

9. By default, comments in the source of a method are searched for and optionally replaced, in Jade methods only. Uncheck the **Include Comments** check box if you do not want to search for and optionally replace occurrences of the search text in method comments, including line (**//**) and block (**/* */**) comments.

   Note The **Include Comments** check box is not displayed if the **Methods** check box is not checked, as it does not apply to the search of class or global constants.

   The **Include Comments** check box is set to **true** (that is, checked) for the first search performed after logging on. Subsequent displays of the Global Search and Replace dialog set the check box value to the prior setting that was used to perform a search or replace action.

10. If you want the search text to be interpreted as a regular expression pattern, check the **Regex** check box; for example, enter **[A-Za-z]+\d{4}** to search for one or more letters followed by four digits. By default, the **Regex** check box is unchecked. If you check the **Regex** check box, the **Separate Lines** check box is enabled.

    The following list shows examples of regular expressions that you can use in a Regex search. To search for:

    - At least a 4- or 5-digit number assignment ignoring whitespace; for example, an error code

      ```
      \s*:=\s*\d{4,5}
      ```

    - Transaction methods

      ```
      (abortTransaction|beginTransaction|commitTransaction)
      ```

    - **unloadForm** in an epilog

      ```
      epilog[\s\S]*unloadForm
      ```

    - A variable declaration named **userName**

      ```
      vars[\s\S]*\buserName\b[\s\S]*begin
      ```

11. If you want to treat the search text as if it is broken up into separate lines, check the **Separate Lines** check box; for example, if you want to search for a specified string at the start or end of a line, check this check box. When this check box is checked, the search breaks the text into individual lines at each **CrLf** end-of-line sequence. By default, the **Separate Lines** check box is unchecked.

12. Use the options in the **Restrict Search To** group box to select the classes and schemas in which the search is actioned.

    When the Class Browser or Primitive Types Browser has focus, the search is performed only on the selected class in the current schema and its superschemas (that is, the **Selected Class Only** option button and the **Current Schema And Superschemas** option button are selected).

    Conversely, when the Schema Browser has focus, options that are not valid are disabled and the **All Classes** and **Current Schema And Superschemas** option buttons are selected by default.

    If you want to restrict the search or replacement to another option, the values that are you can select are listed in the following table, to enable you to select the appropriate option.

Class Restriction Options	Schema Restriction Options
Selected Class Only (Class or Primitive Types Browser default value)	Current Schema Only
Selected Class And Subclasses	Current Schema and Subschemas
Selected Class And Superclasses	Current Schema and Superschemas (default value)
All Classes (Schema Browser default value)	All Schemas

13. To specify the entities for which to search, in the Search Entities group box:

    - If you do not want to search the name and text of all methods that match your search criteria, uncheck the **Methods** check box. This check box is checked by default.

    - If you want to search the names and values of all class constants that match your search criteria, check the **Class Constants** check box. This check box is unchecked by default.

- If you want to search the names and values of all global constants that match your search criteria, check the **Global Constants** check box. This check box is unchecked by default.

  The options in the Restrict Search To group box that control what classes to search are ignored when searching for global constant names and values. If the **Global Constants** check box is the only one checked in the Search Entities group box, the classes options group box is disabled.

- If you want to search the names and values of all imported entities that match your search criteria, check the **Include Imported** check box. This check box is unchecked by default.

Notes The **Class Constants** and **Global Constants** check boxes are disabled if you checked the **Keyword Search** check box in step 6 of this instruction.

The **Class Constants** and **Global Constants** check boxes are unchecked by default, because those options cannot be used with the replace facility. Checking those options disables the **Replace** button.

14. To start searching the selected class and schema options for your specified text, click the **Find** button.

15. If you want to replace all occurrences of the text specified in the **Search** combo box in all classes and schemas that match your selection criteria with the replacement text specified in the **Replace** combo box, click the **Replace** button. (This button is disabled for keyword searches.)

    If Jade finds the text string that matches your specified options and you unchecked the **Prompt On Replace** check box, all occurrences of the located text are then replaced with the specified replacement text and focus is returned to the editor pane.

    The default **Prompt on Replace** check box value indicates that a message dialog is displayed for each occurrence of the searched for text, to enable you to confirm that you want to replace it with the specified text.

16. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

A progress dialog is then displayed, showing the current class that is being searched, the percentage of the search that has been completed, and the current number of occurrences that have been located.

When the search is complete, any entries that are located are displayed alphabetically, and depending on the entities that were included in your search, the list could contain the names of methods, class constants, and global constants. Clicking on a class constant or global constant entry displays the details of the constant definition.

You can cancel a search, by clicking the **Cancel** button in the progress dialog.

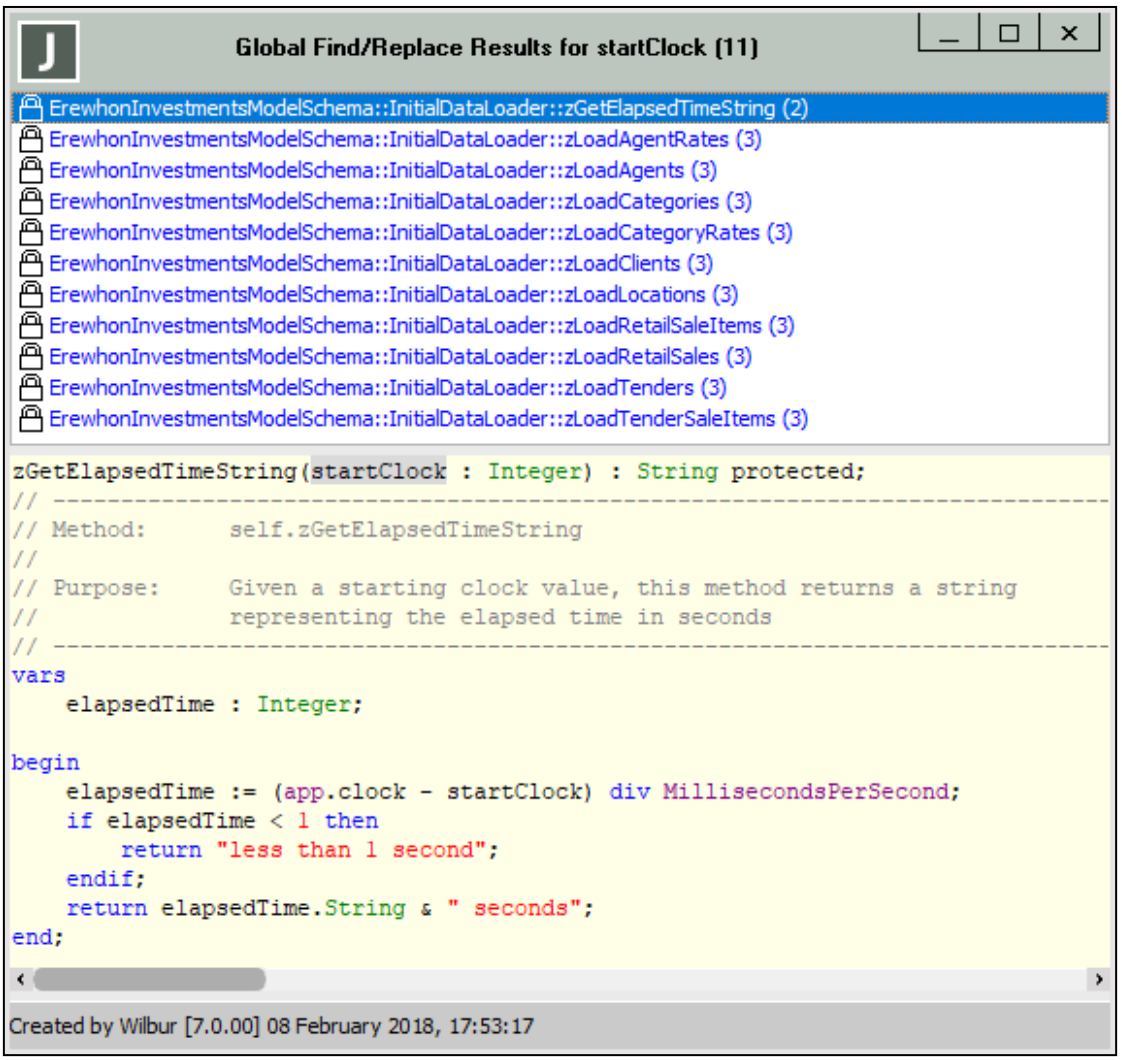
If no occurrences of your specified text are located, the Message dialog is displayed, advising you that the search text was not found.

For details about viewing search results, see "Viewing Located Search Results", in the following section.

Note The values of the **Case Sensitive** and **Full Word Only** check boxes are **false** (that is, unchecked) the first time the dialog is displayed after logging on to the Jade Platform development environment. After you perform a search, the values used in the search are saved. Those values are restored the next time the dialog is used in your current development environment session.

Viewing Located Search Results

When you search by using the Edit menu **Global Find/Replace** command (or the Shift+Ctrl+F3 keys) and Jade has located all occurrences of the text in the classes and schemas that match your search criteria, the Global Find/Replace Results Browser, shown in the following image, is then displayed.



The specified search text is displayed in the title bar of the dialog, with the number of occurrences of that text displayed in parentheses if the text occurs more than once; for example, **AllOrderedThings::findCustomer (2)**.

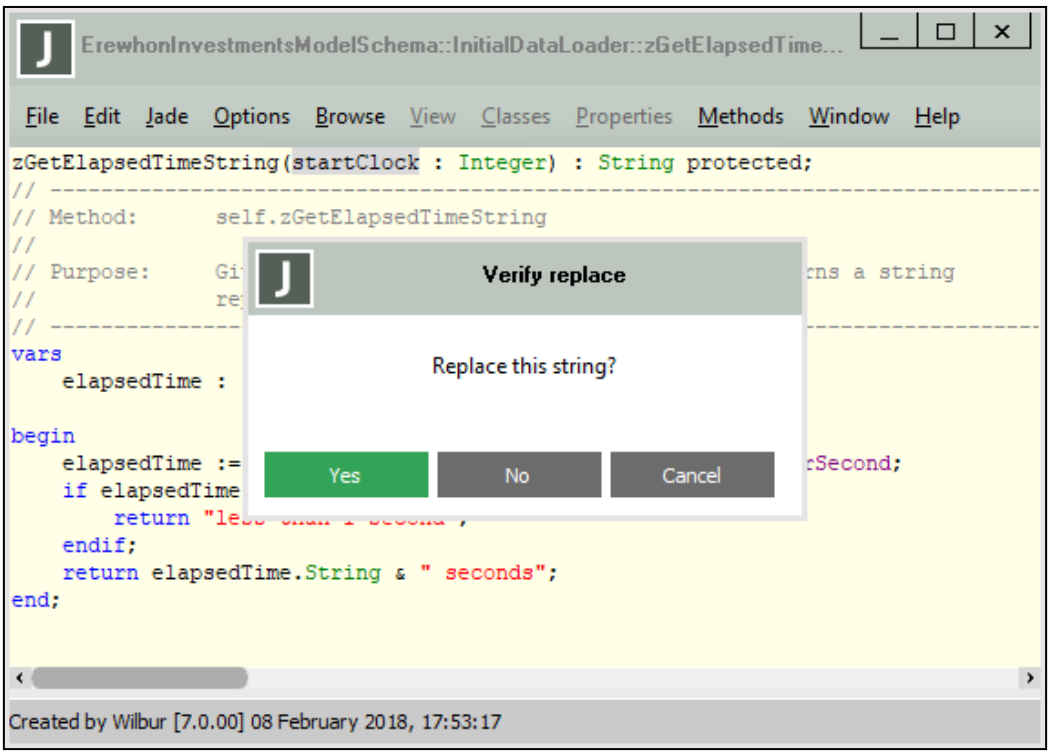
All classes and their methods in which the specified text was located are listed in the top of the dialog. You can scroll up or down the list to view specific methods, if appropriate. When you select a class and method in the list, the method is displayed in the editor pane, with the text that was searched for highlighted.

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods", in Chapter 4.

You can use the editor pane to modify and compile a method, if required. You can also extract a single method selected in the methods list at the top of the dialog. For details, see "Extracting a Method Selected in a Methods List", in Chapter 4.

When you perform a search and replacement by clicking the **Replace** button in the Global Search and **Replace** dialog and an occurrence of your specified text is located, the editor pane is then displayed for the method containing the first instance of your specified text and you did not change the default value of the **Prompt On Replace** check box (that is, you want to be prompted for each replacement).

The following image shows an example of the Global Find/Replace Results Browser with the search text highlighted and the Verify Replace dialog, displayed for each occurrence of the text string, superimposed on the editor pane.



In the Verify Replace dialog, perform one of the following actions.

- Click the **Yes** button to confirm that the located text is to be replaced.
- Click the **No** button to leave the highlighted text unchanged and continue searching for the specified string.
- Click the **Cancel** button to abandon the search and return focus to the editor pane.

When you have responded to each prompt for replacement confirmation to meet your requirements, the next occurrence of that text is then located and highlighted until you have responded to the prompt for each occurrence.

The Global Find/Replace Results Browser is then displayed, to enable you to scroll through the methods to view or modify and compile any method you require.

Tip When your specified text has been located and focus is returned to the editor pane, you can make further searches in the editor pane for that text by selecting the **Find Again** command from the Edit menu, or simply pressing F3. The next occurrence of that text is then located and highlighted or the Message dialog is displayed advising you that the search text was not found if no further occurrences of that text are located.

The **Find Again Reverse Direction** command from the Edit menu performs another search using your last set of search options but in the opposite direction. If the search was specified as a forward search, the reverse search is performed backwards. Conversely, if the search was specified as a backwards search, the reverse search is forwards.

When searching using Shift+Ctrl+F3 and multiple search result windows are open, each search results window restores its own search criteria if it is not current so that F3 works as expected if you have multiple search result windows open.

Searching the Results of a Previous Search

When you perform search and replace actions over the current list of methods displayed on a form (including the list of methods displayed for a class; the list of methods displayed for references of a class, property, or method; the list of implementors of a method; and the list of methods displayed after a search), you can select the **Search/Replace Methods in Method List** command from the Methods menu to use the results of the previous method search as the starting point for a subsequent search.

When you select this command, the Search/Replace the Methods in the Methods List dialog is then displayed. (This is the standard global search and replace form without the controls in which to select the schema and classes.)

When you specify your search or replace criteria and then click the **Search** or **Replace** button, only those methods on the form when the search or replace action is performed are searched. For example, you can perform a text search over all methods for a schema using a global search and then perform another search over only those methods found in the first search (and then another search over the results of the second search, and so on).

Applies to Version: 2016.0.02 (Service Pack 1)

Searching a Browser for an Entity

You can search a browser for a specific property or method, or the hierarchy Class Browser for a specific class constant, by using one of the following menu commands.

- Properties menu **Search For Property Name** command (Ctrl+6)
- Methods menu **Search For Method Name** command (Ctrl+7)
- Constants menu **Search For Constant Name** command (Ctrl+6) from the hierarchy Class Browser only

Note To search for a constant name, the **Const** sheet of Properties List in the upper middle pane of the Class Browser must be selected. To search the Class Browser for a property by name, a Properties List sheet other than **Const** must be selected.

The displayed list of properties or methods is filtered based on what you type into the search text box. You can search all properties or methods displayed in a list box in the:

- Class Browser
- Class References Browser
- Messages Browser
- Methods Browser

- Interface Browser
- Primitive Types Browser
- Sender Browser lists such as references and implementors (the Methods menu **Go To Line** command, which defaults to line number mode, enables you to go to a specific line number or code position)
- Unused Entities Browser

When the entity search request is made, the following occurs.

1. A text box is displayed at the top of the list box to be searched.
2. A table is displayed below the text box, and it initially contains all of the elements in the list box to be searched (except duplicate names and class names).
3. Focus is positioned in the text box.

You can perform the following actions.

- Enter text in the text box. After each character is entered, only those entries in the list box that have the specified string somewhere in the entity text (which is case-insensitive) are displayed in the list.
- Press the up or down arrow to move through the list.
- Click an entry in the list to hide the text box and list, and then select the clicked list entry in the original list box.
- Press Enter to hide the text box and list, and then select the currently highlighted entry in the list in the original list box.
- Press Esc to hide the text box and list so that focus is positioned in the original list box without any changes.
- Click another window to hide the text box and list without impacting the original list box.

Applies to Version: 2020.0.01 and higher

Specifying Text for a Schema Element

When you define or maintain one of the following schema elements, an **Enter Text** button on the appropriate definition dialogs accesses a text editor window to enable you to specify or maintain descriptive text for the current element as part of the definition or maintenance of that element.

- Attribute (for details, see "Adding an Attribute Property", in Chapter 4)
- Condition (for details, see "Adding Conditions to Classes or Primitive Types", in Chapter 4)
- Constant (for details, see "Adding a Class Constant", in Chapter 4)
- External function (for details, see "Defining External Functions", in Chapter 8)
- Interface (for details, see "Defining an Interface", in Chapter 14)
- Jade method or an external method (for details, see "Adding Jade Methods to Classes or Primitive Types", in Chapter 4, or "Adding External Methods to Classes", in Chapter 8)
- Reference (for details, see "Adding a Reference Property", in Chapter 4)
- User-defined global constant in a user schema (for details, see "Adding a Global Constant", in Chapter 4)

For details about specifying or changing text for a class, see "Specifying Text for a Class" and "Using the Free-Standing Editor Window to Define Text", elsewhere in this chapter. For details about specifying or changing text for an interface, see "Specifying Text for an Interface" in Chapter 14 and "Using the Free-Standing Editor Window to Define Text", later in this chapter.

Notes The supplementary descriptive text that you define for an element is for display only. For example, when you specify text for properties or methods in a class and you subsequently print that class, your text is printed after each of the properties or methods in which it was defined or you can view it by selecting the appropriate **Text** menu command for the element whose descriptive text you want to view.

You can also specify descriptive text for an element when you are not defining or changing that element, by selecting the **Text** command from the appropriate menu (for example, the Methods, Conditions, or Properties menus, depending on the element that is currently selected).

To specify descriptive text for an element

1. Click the **Enter Text** button on the element definition dialog (for example, the Define Attribute dialog). The Enter Text window is then displayed.
2. In the editor pane, enter the descriptive text for the element that you are defining or changing. The editor determines the behavior of the text editor window. Text is *not* automatically wrapped to fit the current window size. When you want text to start in the next line, press Ctrl+Enter. (If you want text to wrap in text and source windows, see "Maintaining Editor Options", in Chapter 2.)
3. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your entry.

The Editor Text window is then closed and focus returns to the definition dialog from which you accessed the Editor Text window.

Descriptive text for schema elements can be viewed in one or more of the following ways, depending on the type of element for which text is defined.

- In bubble help when the cursor is positioned over the element for a few seconds (for example, a method)
- In the integrated editor pane of the browser when an element is selected (for example, a condition or a property)
- Printed beneath the element name when the element is printed (by selecting the **Print Selected** command from the File menu) and the **Text** check box and the appropriate element type check box in the Print dialog is checked

For example, descriptive text of an attribute or reference property is displayed in bubble help when the cursor is positioned over the property in the Properties List of the Class Browser and it is also printed beneath the class name when the class is printed (by selecting the **Print Selected** command from the File menu) and the **Text** check box in the Print dialog is checked.

Using the Free-Standing Editor Window to Define Text

When you select the **Text** command from a menu (for example, the Classes, Interfaces, Methods, or Properties menu), a free-standing editor window is then displayed. The free-standing text editor window functions in the same way as the editor pane on the **Text** sheet of the Define Class dialog or on the Enter Text window, which is accessed by clicking the **Enter Text** button on the definition dialog of other schema elements.

For details about specifying descriptive text during the definition phase of a schema element, see "Specifying Text for a Class" or "Specifying Text for a Schema Element", earlier in this chapter. See also "Specifying Text for an Interface", in Chapter 14.

» **To close the text editor window, perform one of the following actions**

- Click the close icon at the top right corner of the window.
- Press Ctrl+F4.

If you only viewed existing class or interface text and made no changes, the editor window is then closed.

If you made any changes in the editor window, a message box is then displayed, asking you to confirm that you want to save your text.

» **To save your changes and close the editor window, perform one of the following actions**

- Click the **Yes** button in the message box.

  Alternatively, click the **Cancel** button if you want to continue specifying or maintaining text for your class, or the **No** button if you want to close the editor window without saving your new or changed text.
- Select the **Save** command from the File menu.

After a momentary delay, the text editor window is then closed.

Removing a Schema Element

You can physically delete an element from a user-defined schema by selecting the **Remove** command from the appropriate menu for the selected element in a browser window or clicking the **Remove** button in the String Browser. For example, if you want to delete a class selected in the Class List of the Class Browser, select the **Remove** command from the Classes menu.

This section covers the physical deletion of the following user-defined schema elements.

- Applications
- Categories for global constants
- Classes
- Constants
- External functions
- Formats
- Libraries
- Map files
- Methods, including conditions
- Properties
- Servers
- Translatable strings

Notes You can remove only user-defined elements. You cannot remove a user-defined element that is defined in another schema, an application that is used in a package, a class that has property references, or a class that has subclasses. You must first remove any subclasses of a class before you can delete the higher-level superclass.

The **Remove** command is disabled if the selected element has subobjects, is a system object, or it is defined in another schema. Use the Copy classes of *schema-name::class-name* browser to find subschema copies of a class. For details, see "Displaying Subschema Copies of a Class", earlier in this chapter.

For details about removing:

- A user-defined schema, see "Removing a User-Defined Schema", earlier in this chapter
- A class from a schema view, see "Removing a Class from a Schema View", earlier in this chapter
- An interface and its associated constants, methods, and any text, see "Removing an Interface", in Chapter 14
- A relational view, see "Removing a Relational View", in Chapter 9
- An RPS mapping, see "Removing an RPS Mapping", in Chapter 15
- A C# or web service exposure, see "Removing a C# Exposure Definition", in Chapter 17 of this document, or "Defining a Web Service Provider Application", in Chapter 11 of the *Developer's Reference*, respectively
- An external database schema, see "Deleting an External Database Schema", in Chapter 3 of the *External Interface Developer's Reference*
- An import or export package, see "Removing a Package", in Chapter 8 of the *Developer's Reference*
- An ActiveX type or .NET library, see "Removing an ActiveX Type Library" or "Removing a .NET Library", in Chapter 16
- An HTML document file and its associated HTML document subclass, see "Removing an HTML Document", in Chapter 12

To remove a schema element

1. In the appropriate browser, select the element that you want to physically delete from the Jade database; for example, an application selected in the Application Browser or a reference property selected in the Properties List of the Class Browser.
2. Select the **Remove** command from the menu that relates to your selected element (for example, the Application menu if you have selected an application in the Application Browser or the Properties menu if you have selected a reference property in the Properties List of the Class Browser.) Alternatively, if the current browser is the String Browser, click the **Remove** button.

   A Confirm Delete message box is then displayed.
3. To confirm that you want to remove the specified element and all of its constants, properties, and methods, click the **OK** button. Alternatively, click the **Cancel** button to abandon the deletion.

   If you selected the deletion of a class that has property references and the class deletion is valid (that is, it is a user-defined element in the current schema), an error message box is then displayed. Click the **OK** button to return to the Class Browser and take the appropriate actions before you attempt the removal of that class again.

   Caution Deleting a class with instances that have a superclass property reference set does not display the error message box. The deletion of the class is completed and the superclass reference properties are marked as invalid.

When the deletion of the element is successful, the browser is then updated to reflect the removal of the element. (There may be a momentary delay while this updating occurs.) When you delete a translatable string, you are warned if other translatable strings use the translatable string that you want to delete. For details about translatable strings, see "Translating Strings", in Chapter 11.

Notes As Jade must have at least one application in a schema, you can remove an application only if there are two or more applications defined in the schema and the application is not used in a package.

If you selected a user-defined global constant category for removal and that category contains global constants, the message box that is displayed advises you that the category has constants that will be deleted with the category. Clicking the **Yes** button in the message box initiates the deletion of both the category and the global constants that it contained.

You can also use the **Cut** button from the Browser toolbar to logically delete a selected portion of a method in the current editor pane. For details, see "Cut Toolbar Button", in Chapter 2.

Printing a Selected Schema Element

The Print Options dialog, accessed by clicking the **Print Selected** icon in the Browser toolbar or by selecting the **Print Selected** command from the File menu, has options that enable you to print selected elements in the Jade Platform development environment.

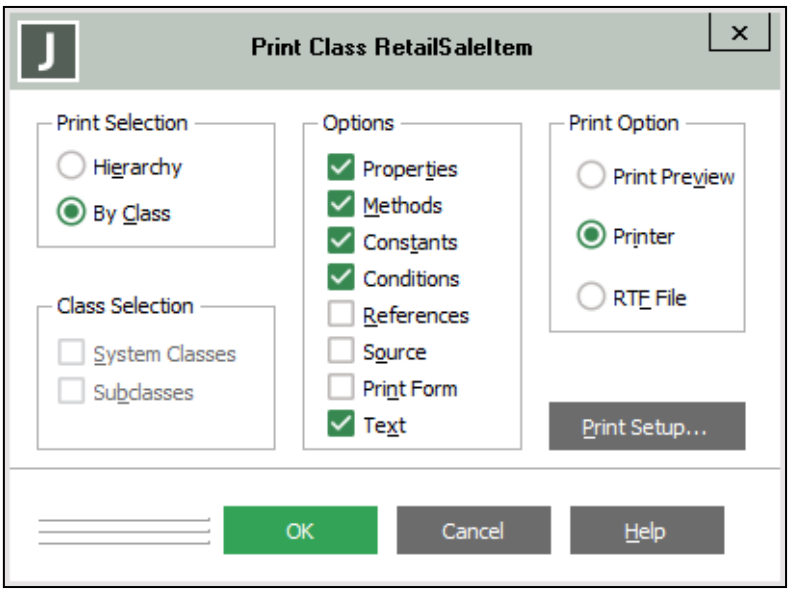
The current element determines the selective print options. For example, the Print Options dialog accessed from the Class Browser provides different options from that accessed from the Primitive Types Browser or Interface Browser.

You can access the Print Options dialog only from the Class Browser, Primitive Types Browser, Interface Browser, Global Constants Browser, and the Summary of Patches window.

Note The **Print Selected** command enables you to define how specific parts of your Jade Platform development environment for the selected object is to be printed. It does *not* print output from runtime Jade applications, whose print requirements are defined using the **Printer** class or the **CMDPrint** (Printer Dialog) class.

See also "Using the Relationship View Window" and "Printing Class or Primitive Type Constants", in Chapter 4 of this document, "Printing an Interface" in Chapter 14 of this document, and "Printing a Patch History Summary", in Chapter 3 of the *Development Environment Administration Guide*.

An example of the dialog that is displayed when you select a class in the Class List of the Class Browser is shown in the following image.



You can select operations documentation (technical objects such as subclasses, methods, or references) for development purposes, and user documentation (design-related objects such as forms) for such things as end-user sign-off.

Click the appropriate check boxes and option buttons for the selections that you require for your documentation. A check mark is displayed in check boxes of options that are selected for printing. Option buttons are displayed as filled, or solid, when selected.

Invalid selections for the selected element are dimmed; for example, if you are selecting your print options from the Class Browser and you select the **Hierarchy** Print Selection option button, the **Sub Classes** Class Selection check box and the **References** and **Source** check boxes in the Options group box are disabled.

To select the print options for the element selected in a browser window

1. When your selected element is a class, the Print Selection group box enables you to print detailed information for the class in class or hierarchy order. (This group box is displayed only when you access the Print Option dialog when you have selected a class in the Class Browser.)

   The default **By Class** option button prints the inheritance of each selected class and its subclasses. For example, if you select printing of an **Animal** class by class, your output might be as follows:

   - Animals
     - Properties
     - Constants
     - Jade Methods
   - Bird
     - Properties
     - Constants

- Jade Methods
- Gull
  - Properties
  - Constants
  - Jade Methods
- Parrot
  - Properties
  - Constants
  - Jade Methods

Alternatively, select the **Hierarchy** option button if you want to print a summary of information for the selected class in hierarchical order. For example, if you select hierarchical printing of an **Animal** class, your output might be as follows:

- Animal
  - Bird
  - Kiwi
  - Parrot
  - Mammal
  - Cat
  - Dog

2. When the selected element is a class, the Class Selection group box enables you to select additional class details. (This group box is displayed only when you access the Print Option dialog when you have selected a class in the Class Browser and the **By Class** option button is selected in the Print Selection group box of the Print Options dialog.)

   Check the:

   - **System Classes** check box if you want your print output to include system classes when printing the selected class in hierarchical order. (This check box is enabled only when the **Hierarchy** option button is selected in the Print Selection group box.)
   - **Subclasses** check box if you want your print output to also include subclasses of the selected class when printing the detailed class information. (This check box is enabled only when the **By Class** option button is selected in the Print Selection group box and the selected class has subclasses.)

3. In the Options group box, select the information that you want or print for the selected element by checking check boxes that are not selected by default or unchecking any check box whose option is selected by default if you do not want to include that information in your print output.

   Your print output can contain the following information, to meet your requirements.

   - The **Properties** check box, displayed when a class or property is the selected element, prints the properties and constants of the selected class (and subclasses, if applicable) or details of the selected property. Properties information is printed by default; that is, this check box is checked.

- The **Methods** check box, displayed when a class, primitive type, interface, or method is the selected element, prints the Jade (user-defined) and external (system) methods for the selected class (and subclasses, if applicable) or primitive type, or details of the selected method. Method details are printed by default; that is, this check box is checked.
- The **Constants** check box, displayed when a class, primitive type, interface, or constant is the selected element, prints the class constants provided by the selected class (and subclasses, if applicable) or primitive type, or details about the constant selected in the Properties List of the Class or Primitive Types Browser. Constant information is printed by default; that is, this check box is checked.
- The **Conditions** check box, displayed when a class or primitive type is the selected element, prints the conditions defined in the selected class (and subclasses, if applicable) or the selected condition. Condition information is printed by default; that is, this check box is checked.
- The **References** check box, displayed when a class is the selected element, prints the print all references (relationships) to the selected class (and subclasses, if applicable). References are not printed by default; that is, this check box is unchecked.

  The **References** check box is enabled only when the **By Class** option button is selected in the Print Selection group box.
- The **Source** check box, displayed when a class, primitive type, or method is the selected element, prints the method source (code) of the selected class (and subclasses, if applicable) or primitive type, or of the selected method. Method sources are not printed by default; that is, this check box is unchecked.
- The **Print Form** check box, displayed when a class is the selected element, prints all form layouts defined in the Jade Painter when the selected class is the **Form** or **Object** class and you checked the **Subclasses** check box in the Class Selection group box or the form layout if the selected class is a subclass of the **Form** class. Forms are not printed by default; that is, this check box is unchecked.
- The **Text** check box prints all associated text specified for the selected element. Text information is printed by default; that is, this check box is checked. For more details, see "Specifying Text for a Schema Element", earlier in this chapter.

4. If you do not want your selections to be output to the printer, select one of the following.
   - **Print Preview** option button, to preview the output on your workstation monitor. For details about previewing print output, see "Previewing Print Output" under "Printer Class", in Chapter 1 of the *Encyclopaedia of Classes*.
   - **RTF File** option button, to output your selections to a Rich Text Format (.**rtf**) file in your working (**bin**) directory. You can view your .**rtf** file output by using a text editor; for example, Word for Windows.
5. Click the **Print Setup** button if you want to change the setup of your printing. For details, see "Setting Up Your Printer", in the following subsection.
6. Click the **OK** button to confirm your selections. Alternatively, click the **Cancel** button to abandon your selections.

Setting Up Your Printer

The Print Setup dialog enables you to modify the printing of the schema element selected in a browser.

Note The print setup dialogs that are displayed and the options that are available are operating system and printer-dependent. As these dialogs cannot be controlled from within Jade, this section provides only an example only of a printer setup. These dialogs and the options that they provide may therefore differ in your environment.

To select your printer set-up options

1. Select the **Print Setup** command from the File menu or the **Print Setup** button on the Print Options dialog. (For details about the Print Options dialog, see "Printing a Selected Schema Element", in the previous section.)
2. Select a specific printer in the **Name** drop-down list box if you do not want your print output directed to your default printer.
3. In the Paper group box, select the paper size that you require in the **Size** drop-down list box if you do not want your selected schema element printed on the default paper size of the printer selected in the **Name** list box.
4. In the Paper group box, select the location of paper in the printer in the **Source** drop-down list box if you do not want the paper sourced from the default source of the printer selected in the **Name** list box.
5. In the Orientation group box, select the appropriate **Portrait** or **Landscape** option button if you want your selected element printed in the orientation other than the default for the selected printer.
6. Click the **Properties** button when you have made you required selections. The Document Properties dialog is then displayed, to enable you to specify print requirements for your print output.

   Use the Document Properties dialog to specify the format of your print output, which can include the following.

   - Paper size
   - Two-sided printing
   - Number of copies
   - **Options** button (accesses the Advanced Document Properties dialog)
   - **Halftone** button (accesses the Halftone Color Adjustments dialog)

   The Print Setup, Document Properties, Advanced Document Properties, and Halftone Color Adjustments dialogs are Windows dialogs. For more details, refer to your *Microsoft Windows User's Guide*.

   If the Document Properties dialog provides **Options** or **Halftone** buttons, use these to select additional print properties or connect to another printer, respectively.
7. Click the **OK** button to confirm your selection. Alternatively, click the **Cancel** button to abandon your selections.

Performing a Reorganization

Use the **Reorg** command from the Schema menu to perform reorganization actions when the current schema requires reorganization. The **Reorg** command is enabled only when the schema that is selected in the Schema Browser requires reorganization; for example, after you have added, deleted, or modified a property.

Note You can also reorganize your Jade schema from outside the Jade Platform development environment, by using the standalone Schema Load utility or the batch **JadeSchemaLoader** application command line. For details, see the *Schema Load User's Guide*, or "Reorganizing the Database from the Command Line", later in this chapter.

This section contains the following topics.

- Using the Reorg Command
- Exiting from Jade when a Reorganization is Required
- Reorganizing the Database from the Command Line

For reorganization reference information, see Chapter 14 of the *Developer's Reference*.

Using the Reorg Command

When a class in a schema requires reorganization, the **Reorg** command in the Schema menu from the Schema Browser is enabled.

If no reorganization is required for that schema, this command is disabled. The **Reorg** command submenu enables you to perform the following actions.

- Reorganizing Your Schema
- Reorganizing All Schemas that Require Reorganization
- Initiating the Reorganization Transition
- Aborting an Incomplete Reorganization
- Restarting an Interrupted Reorganization

For details, see the following subsections.

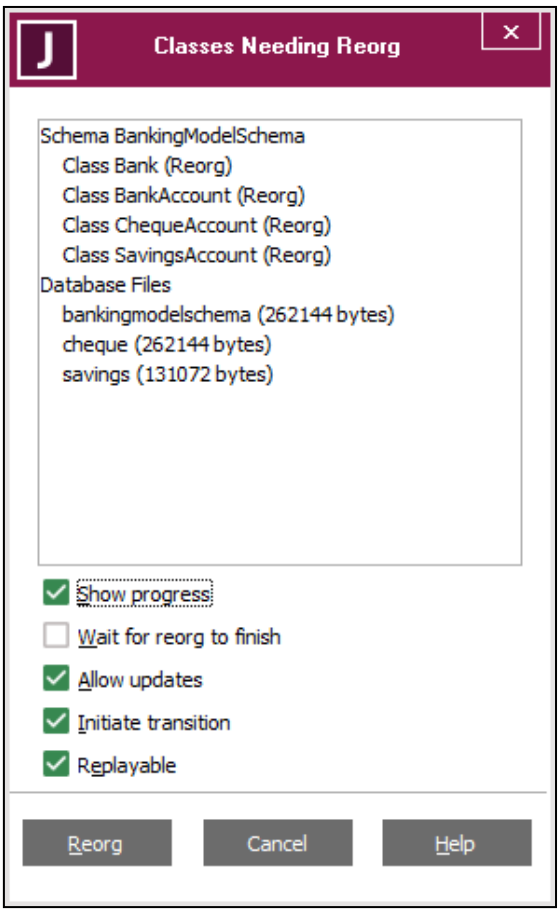
Reorganizing Your Schema

To reorganize classes and database files

1. Perform one of the following actions to reorganize your current schema when it is marked for reorganization (indicated by a red light).
   - Click the **Schema Needs Reorg** toolbar button. When the schema is not marked for reorganization, the toolbar button changes to a green light and bubble help for the button displays **Reorg Is Not Required**.
   - Select the **Reorg** command from the Schema menu and then select the **Reorg Schema** command from the submenu that is displayed.

Note If you perform either of these actions when a reorganization is in progress, the progress dialog of the currently executing reorganization is displayed (that is, a new reorganization is not started, as only one reorganization can be in progress at any one time).

The Classes Needing Reorg dialog, shown in the following image, is then displayed, listing all classes in all schemas that require reorganization. The database files and their sizes are also displayed at the end of the list, to enable you to estimate the potential disk requirements and time that the reorganization requires.



2. If you do not want the reorganization progress dialog displayed during the reorganization, uncheck the **Show progress** check box. When this is checked (the default value), you can monitor the progress of the reorganization and you can cancel the reorganization at any time. You can close the progress dialog by clicking the close icon at the top right corner in the window title bar. This does not cancel the reorganization.

3. If you want to wait until the reorganization is complete, check the **Wait for reorg to finish** check box. By default, this check box is not checked; that is, other work can continue in the Jade Platform development environment during the reorganization. In multiuser mode, you can exit from the Jade Platform development environment and the reorganization continues.

   If a reorganization is required before a form and data definition (.**ddb** or .**ddx**) file can be loaded, the Classes Needing Reorg dialog is displayed with the **Wait for reorg to finish** check box checked and disabled.

4. If you do not want to allow other users to continue development and update the database while the reorganization is in progress, uncheck the **Allow updates** check box. (The **Initiate transition** check box is then checked if it was unchecked and it is disabled.)

The **Allow updates** check box is checked by default, to allow updates to proceed up until the reorganization transition is initiated; for example, other users can still modify (that is, edit and compile) methods in the development environment. This check box is disabled and unselected when the **FastBuildBTreeCollections** parameter in the [JadeReorg] section of the Jade initialization file is set to **true**. If updates are disabled while the reorganization is in progress, the reorganization must initiate the transition. Any error occurring during the reorganization results in the reorganization being aborted; that is, the reorganization cannot be restarted. For details, see "Allowing Updates" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*.

5. If you do not want the reorganization to initiate the transition, uncheck the **Initiate transition** check box. For details, see "Initiating Transition" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*. The **Initiate transition** check box is checked and disabled if you unchecked the **Allow updates** check box in the previous step.

   Note When performing a reorganization in production mode, the transition must be initiated in single user mode. For details, see "Running Jade Production Mode Databases", in Chapter 1 of the *Runtime Application Guide*.

6. If you do not want to perform a replayable reorganization, uncheck the **Replayable** check box. If archival recovery is enabled (that is, the **EnableArchivalRecovery** parameter in the [PersistentDb] section of the Jade initialization file is set to **true**), the **Replayable** check box is enabled and checked by default. For details, see "Replayable Reorganizations" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*.

7. Click the **Reorg** button. Alternatively, click the **Cancel** button to abandon the reorganization.

The reorganization of all classes and database files in all schemas requiring reorganization is then initiated. After a delay (depending on the number of class instances requiring reorganization), a message box advises you of the reorganization completion status.

Notes If concurrent updates are disallowed, reorganization cannot be initiated if there are any outstanding transactions. If other users are using the Jade database, you must wait until all transactions are complete before you initiate the reorganization. No transactions can be started when a reorganization that disallows concurrent updates is in progress.

If a reorganization is in progress and you select the **Reorg** command from the Schema menu, the progress dialog is displayed.

If a schema load or the removal of versioning from a schema is attempted when a reorganization is in progress (regardless of whether the reorganization progress dialog is displayed), the load or version removal fails.

For details about:

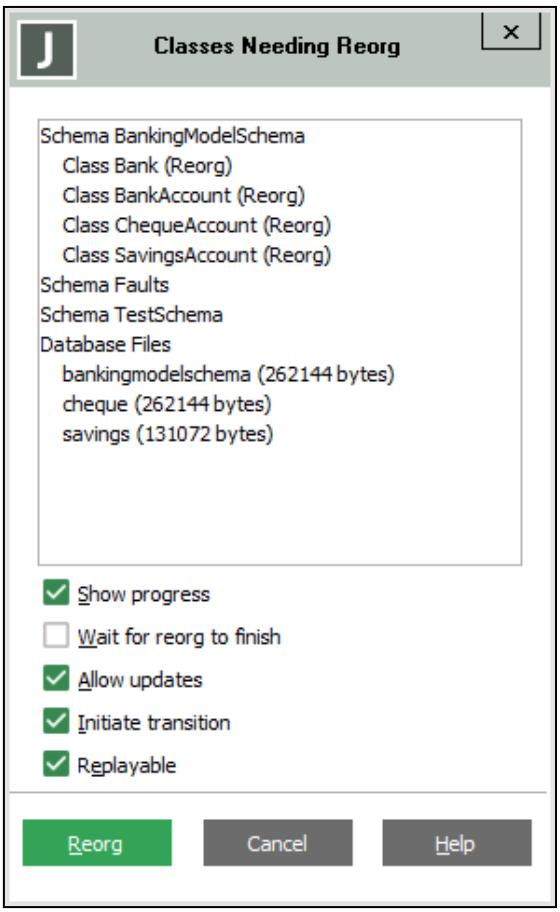
- Reorganizing all schemas that require reorganization, see "Reorganizing All Schemas that Require Reorganization", in the following section.
- Reorganization in a single user, production mode database, see Running Jade Production Mode Databases", in Chapter 1 of the *Runtime Application Guide*.

Reorganizing All Schemas that Require Reorganization

» To reorganize all classes and database files in all user schemas that require reorganization

1. Select the **Reorg** command from the Schema menu and then select the **Reorg All Schemas** command from the submenu that is displayed. The **Reorg** command and the **Reorg All Schemas** command on the reorganization submenu are enabled only when a schema is marked for reorganization (indicated by a red light in **Schema Needs Reorg** toolbar button).

The Classes Needing Reorg dialog, shown in the following image, is then displayed, listing all schemas that are marked for reorganization.



The database files and their sizes are displayed at the end of the list, to enable you to estimate the potential disk requirements and time that the reorganization requires.

2. Perform the actions that you require by checking or unchecking the appropriate check boxes.

For details about the other actions that you can perform, see "Reorganizing Your Schema", in the previous section.

Initiating the Reorganization Transition

When you reorganize a schema and uncheck the **Initiate transition** check box in the Classes Needing Reorg dialog (for details, see the previous section), the Reorg message box is displayed when the database reorganization is ready to initiate the transition.

To initiate the reorganization transition

1. Select the **Reorg** command from the Schema menu.
2. Select the **Initiate Transition** command from the submenu that is then displayed. The Classes Needing Reorg dialog is then displayed, with the **Initiate transition** check box checked and disabled.
3. Click the **Reorg** button. Alternatively, click the **Cancel** button to abandon the reorganization.

The reorganization transition is then initiated. For details, see "Initiating Transition" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*.

When the reorganization transition is complete, a message box advises you of the reorganization completion status.

Aborting an Incomplete Reorganization

When a schema reorganization has not completed, the **Abort Reorg** command is enabled in the Reorg submenu.

To abort an incomplete reorganization

1. Select the **Reorg** command from the Schema menu.
2. Select the **Abort Reorg** command from the submenu that is then displayed. (This command is disabled if a reorganization is not in progress.)

You can abort the following reorganizations.

- A currently active reorganization
- An interrupted replayable reorganization (for example, a replayable reorganization failed because an inverse could not be populated or a dictionary could not be rebuilt)
- An incomplete non-replayable reorganization (for example, when a system failure occurs during a non-replayable reorganization)

Aborting a reorganization undoes the effects of the incomplete reorganization. When a reorganization is performed, backup copies of reorganized files are retained and they are restarted if the reorganization fails for any reason.

If the reorganization does not complete because of a database server crash, you must restart the reorganization manually by using the **Restart Reorg** command. (For details, see "Restarting an Interrupted Reorganization", in the following section.)

Restarting an Interrupted Reorganization

The **Restart Reorg** command is enabled in the Reorg submenu when a reorganization was unable to complete (for example, because of a database server crash or out of disk error).

You can restart the incomplete reorganization.

To restart an incomplete reorganization

1. Select the **Reorg** command from the Schema menu.
2. Select the **Restart Reorg** command from the submenu that is then displayed.

The interrupted reorganization is then restarted.

Exiting from Jade when a Reorganization is Required

When you exit from Jade, a message box is displayed if a schema contains classes or database files that require reorganization.

To initiate the reorganization of schemas before Jade is closed down

- Click the **Yes** button.

The reorganization is then initiated. When the reorganization has completed successfully, Jade is then closed down.

To exit from Jade without initiating the reorganization

- Click the **No** button.

The reorganization does not take place, and Jade is closed down. The **Reorg** command for the schema that requires reorganization is then enabled next time you start a Jade session.

Reorganizing the Database from the Command Line

You can use the **jadclient** non-GUI client application executable to automate the reorganization of one or more user-defined schemas, passing command line arguments after the **startAppParameters** parameter to specify your reorganization requirements.

For details about the **jadclient** non-GUI client application, see "Running a Non-GUI Client Application using jadclient", in Chapter 1 of the *Runtime Application Guide*.

Note If the restarting of a reorganization fails, you must abort the reorganization before you can initiate an new reorganization.

To reorganize schemas as a non-GUI application, specify the following arguments in the **jadclient** program.

```
jadclient path=database-path
          [ini=jade-initialization-file]
          [server=multiUser|singleUser]
          schema=RootSchema
          app=JadeReorgApp
          startAppParameters
          [action=reorganization-action
          [reorganization-arguments]]
```

The non-GUI client program arguments are described in the following subsections. When specifying reorganization arguments:

- The arguments are case-insensitive.
- The **action** argument and any associated arguments that apply to the **initiateReorg**, **initiateReorgAllSchemas**, or **restartReorg** argument must follow the **startAppParameters** argument.
- Any optional reorganization arguments that you specify are ignored if the value is not applicable to the specified action (for example, the **auditEnableSecondaryApps** action for the **JadeReorgApp** application is disallowed if a reorganization is in progress).

Note The **jadclient** program treats processing arguments enclosed in double (**""**) or single (**''**) quotation marks after the **startAppParameters** argument as single-string entries in the **HugeStringArray**. The handling of strings in this **HugeStringArray** is application-specific. For example, **dir=program files** is treated as a two-string entry and **dir="program files"** is treated as a one-string entry. How these entries are handled is determined by your application.

The following is an example of the command prompt that runs a non-GUI client application to initiate a replayable reorganization of all schemas in the database. The reorganization is suspended before initiating the transition.

```
jadclient.exe path=c:\jade\system ini=c:\jade\system\jade.ini server=multiUser
app=JadeReorgApp schema=RootSchema startAppParameters
action=initiateReorgAllSchemas waitForReorg=true initiateTransition=false
replayableReorg=true reorgAllowUpdates=false noReorgRecovery=true
fastBuildBTreeCollections=false
```

The following example resumes the reorganization process by initiating the transition following a prior reorganization action with **initiateTransition** set to **false**.

```
jadclient path=c:\jade\system server=multiUser schema=RootSchema app=JadeReorgApp
startAppParameters action=initiateTransition
```

The following is an example of the command prompt that runs a non-GUI client application to initiate a reorganization of all schemas requiring reorganization and that are waiting for the reorganization to complete.

```
jadclient path=c:\jade\system server=multiUser schema=RootSchema app=JadeReorgApp
startAppParameters action=initiateReorgAllSchemas
```

The following is an example of the command prompt that runs a non-GUI client application on the primary database to restart server applications on secondary databases. You could do this when such server applications have been shut down on the secondary database because a change has been made in a primary database to a class with no persistent instances in the Jade Platform development environment or by using the Schema Load utility (that is, the **jadload** or **jadloadb** program) with the **Only Structural Versioning** load style.

```
jadclient path=c:\jade\system ini=s:\jade\system\jadesds.ini server=multiUser
schema=RootSchema app=JadeReorgApp startAppParameters
action=auditEnableSecondaryApps
```

The following is an example of the command prompt that runs a non-GUI client application to initiate the offline phase of the reorganization on the primary system.

```
jadclient path=c:\jade\system ini=c:\jade\jade.ini server=singleUser
schema=RootSchema app=JadeReorgApp startAppParameters
action=initiateManagedTransition
```

The **JadeReorgApp** application returns an error code if the command line is invalid. This value corresponds to exception 1407 (*Invalid argument passed to method*). You can obtain the invalid argument in a script by using the status returned from the invocation of the **jadclient** non-GUI client application.

action

The arguments that you can specify in the **action** parameter after the **startAppParameters** parameter are listed in the following table.

Action	Instructs the jadclient program to …
abortReorg	Undo the effects of an incomplete reorganization.
auditEnableSecondaryApps	Restart server applications on a secondary database (run from the primary database).
initiateReorg	Reorganize the schemas identified by the **schemas** parameter.
initiateReorgAllSchemas	Reorganize all schemas requiring reorganization.
initiateTransition	Initiate the transition.
initiateManagedTransition	Initiate the offline phase on the primary system and schedule the initiation of the transition on a secondary database.
restartReorg	Restart an interrupted reorganization. If a previous reorganization was unable to complete (for example, because of a system crash, out of disk error, and so on), use this argument to restart the failed reorganization at the point at which it failed.

If you specify the **initiateReorg**, **initiateReorgAllSchemas**, or **restartReorg** value in the **action** parameter, you can define the optional parameters described in the following sections. (These parameters are ignored if the **action** parameter has any other value.)

fastBuildBTreeCollections

You can specify **true** in the optional **fastBuildBTreeCollections** parameter if you want to allow the building or rebuilding of **ObjectSet** and **MemberKeyDictionary** collections is moved from the Relationship Maintenance phase to the Object Conversion phase. (The default value for this parameter is **false**.)

The following is an example of a Jade non-GUI client command line.

```
jadclient.exe path=C:\Jade\system ini=c:\Jade\system\jade.ini server=multiUser
app=JadeReorgApp schema=RootSchema startAppParameters
action=initiateReorgAllSchemas waitForReorg=true initiateTransition=false
replayableReorg=true reorgAllowUpdates=false noReorgRecovery=true
fastBuildBTreeCollections=true
```

Fast building of collections significantly reduces the elapsed time of large database reorganizations involving collection maintenance by using a faster extract/sort/build algorithm and by allowing the collection maintenance to be performed in parallel by multiple reorganization workers. However, additional disk space up to three times the total size of the collections being built or rebuilt is required for extract and sort files.

Note Fast building of collections is available only for non-updating reorganizations. If you enable fast building of collections, updates are disallowed when you initiate the reorganization.

For details, see "Fast Building of Collections", in Chapter 14 of the *Developer's Reference*.

initiateTransition

You can specify **false** in the optional **initiateTransition** parameter if you do not want the reorganization process to initiate the reorganization transition. For details, see "Initiating Transition" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*. This parameter is ignored if the **action** parameter has any value other than **initiateReorg**, **initiateReorgAllSchemas**, or **restartReorg**.

The default value for this parameter is **true**.

noReorgRecovery

You can specify **true** in the optional **noReorgRecovery** parameter if you want to disallow the creation of temporary backups (**.bak** files) of the original database files (**.dat** files) when a reorganization takes place.

For details, see "Replayable Reorganizations" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*. (The default value for this parameter is **false**, which means that the temporary backup files are created.)

When you set the value to **true**, temporary backup files are *not* created. With this setting, if a reorganization failed, you would need to restore the system from a backup taken before the reorganization.

The following is an example of this parameter specified in the command line when initiating the reorganization.

```
jadclient.exe path=C:\Jade\system ini=C:\Jade\system\jade.ini server=multiUser
app=JadeReorgApp schema=RootSchema startAppParameters
action=initiateReorgAllSchemas waitForReorg=true initiateTransition=false
replayableReorg=true reorgAllowUpdates=false noReorgRecovery=true
fastBuildBTreeCollections=false
```

The **noReorgRecovery** parameter applies only to reorganizations that mutate objects or move objects instances between map files. It has no effect for file compaction and for re-indexing operations.

Caution Your system will not be recoverable if the reorganization fails and you do not have a pre-deployment backup.

Roll-forward recovery fails if this parameter is set to **true** and a reorganization that was aborted is replayed. Replay on an SDS secondary database fails if this parameter is set to **true** and a reorganization that was aborted is replayed.

reorgAllowUpdates

You can specify **true** in the optional **reorgAllowUpdates** parameter if you want to allow other users to continue development and update the database up until the reorganization transition is initiated; for example, other users can still modify (that is, edit and compile) methods in the development environment.

For details, see "Replayable Reorganizations" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*.

The default value for this parameter is **false**; that is, the updates cannot proceed before the reorganization transition. This parameter is ignored if the **action** parameter has any value other than **initiateReorg**, **initiateReorgAllSchemas**, or **restartReorg**.

replayableReorg

If the **EnableArchivalRecovery** parameter in the [PersistentDb] section of the Jade initialization file is set to **true**, in the optional **replayableReorg** parameter you can specify:

- **true**, if you want to perform a replayable reorganization of your Jade database.
- **false**, if you do not want to perform a replayable reorganization of your Jade database.
- **default**, if you want to perform a replayable reorganization only if the schemas are loaded into a primary database.

This parameter is ignored if the **action** parameter has any value other than **initiateReorg**, **initiateReorgAllSchemas**, or **restartReorg**. For details, see "Replayable Reorganizations" under "Reorganization Options", in Chapter 14 of the *Developer's Reference*.

schemas

Use the optional **schemas** parameter to specify a comma-separated list of the schemas that you want to reorganize.

This parameter is ignored if the **action** parameter has any value other than **initiateReorg**.

Chapter 4 Defining Methods, Properties, Constants, and Conditions

This chapter covers the following topics.

- Defining and Compiling Jade Methods and Conditions
  - Adding Jade Methods to Classes, Primitive Types, or Interfaces
  - Compiling Methods
  - Renaming Methods
  - Using Constructors and Destructors
    - create Method Constructors
  - Finding a Method Source Position
  - Going to a Specified Method Line
  - Highlighting Methods in Lists of Methods
  - Displaying Method References, Implementations, and Messages
  - Locating Unused Schema Entities
  - Displaying Unreferenced Methods
  - Displaying Unused Local Variables and Parameters
  - Locating Possible Transient Object Leaks
  - Using a Method in a Workspace or the JadeScript Class
  - Running a Unit Test
  - Viewing Code Coverage Results to Analyze Methods in Your Applications
  - Refactoring a Jade Method
  - Extracting a Method Selected in a Methods List
- Defining Properties
  - Adding an Attribute Property
  - Adding a Reference Property
  - Dynamic Clusters and Properties
  - Adding a Method Mapping for a Property
  - Displaying all References to the Selected Property or Constant
  - Displaying all References that Can Update the Selected Property

- Displaying all References that Can Read the Selected Property
- Displaying all Local References to the Selected Property
- Displaying all Relationships to a Class

- Defining Class, Primitive Type, and Interface Constants
  - Adding a Class, Primitive Type, or Interface Constant
  - Using Class, Primitive Type, or Interface Constants in Your Methods
- Defining Global Constants
  - Accessing the Global Constants Browser
  - Adding a Global Constant Category
  - Adding a Global Constant

Defining and Compiling Jade Methods and Conditions

You can define a Jade user-defined method or condition at any time. When you have defined a Jade method or condition, it can be invoked for any instance of the class or primitive type in which it is defined for any instance of a subclass of that class.

Object methods are always defined for a class. Object methods are generally called from other methods, with the exception of a group of methods referred to as *event* methods. An event method can also be called by the runtime Jade system in response to a window event.

For details about:

- The editor pane, see "Tips for Using the Editor Pane" under "Using the Editor Pane", in Chapter 3
- **JadeScript** and Workspace methods, see "Using a Method in a Workspace or the JadeScript Class", later in this chapter
- Using the AutoComplete functionality, see "Using Jade AutoComplete Functionality", in Chapter 2
- Grouping methods from any class in any schema into a named workspace, see Chapter 13, "Using Method Views to Bookmark Workflows"
- Interface methods, see "Adding Interface Methods" and "Compiling an Interface Method", in Chapter 14
- Defining external methods, see "Adding External Methods to Classes", in Chapter 8
- Web service methods, see "Creating Web Service Methods", in Chapter 11 of the *Developer's Reference*
- Splitting the editor pane horizontally, see "Splitting the Editor Pane View", in Chapter 3
- Changing or renaming an entity (for example, a property, local constant, variable, or method parameter) selected within the body of a method in the editor pane, see "Renaming or Changing an Entity", later in this chapter

Conditions are declarative restricted methods that return a Boolean result. Constraints are conditions used to maintain automatic inverse references when the specified conditions are satisfied. For details, see "Adding Conditions to Classes or Primitive Types", later in this chapter.

Notes A condition that is used as a constraint cannot have parameters nor can it use array indexing ([<*index*>]) and dictionary indexing ([<*key, key, ...*>]). You cannot reimplement a condition inherited from a superschema or superclass. In addition, a condition cannot use array indexing ([*index*]) or dictionary indexing ([*key*, *key*, ...]).

Only automatic references can have a constraint. When the manual side of the inverse reference is set, the condition used as the constraint is evaluated and the automatic inverse is set only if the value of the condition is **true**. If the automatic reference is a **Collection** type, the condition is applied to the members of the collection.

You cannot add a method to the current version of a class if that method was added to the latest version of any superclass or subclass, including those on superschemas and subschemas.

The Jade Platform development environment enables you to move classes and copy or move methods by dragging them to the required class or primitive type. To do this:

1. Select the class or method and then press the:
   - Ctrl or Shift key to move a class
   - Ctrl key to copy a method
   - Shift key to move a method

   Note You can drag a key from an external key dictionary only if there are no items in that dictionary.
2. Drag the class to the class of which it is to be a subclass or the method to the class or primitive type to which you want to copy or move it.
3. Release the button. If the copy or move action is valid, a message box then prompts you to confirm that you want to move the class to the required class or method to the required class or primitive type.
4. Click the **Yes** button to complete the move action.

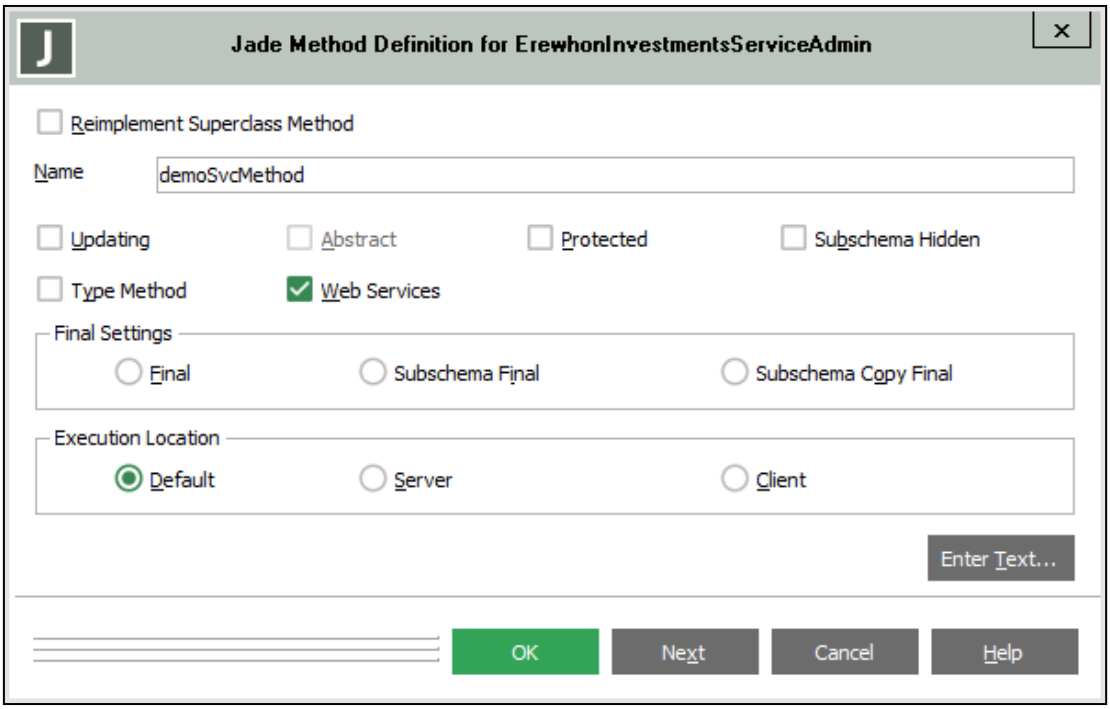
Adding Jade Methods to Classes or Primitive Types

Note Before you define a method, you must select the class or primitive type to which the method is to be added.

» To add a method to a class or primitive type

1. In the Class List of the Class Browser or the Primitive Types list of the Primitive Types Browser, click on the class for which the method is to be added.
2. Select the **New Jade Method** command from the Methods menu.

The Jade Method Definition dialog, shown in the following image, is then displayed.

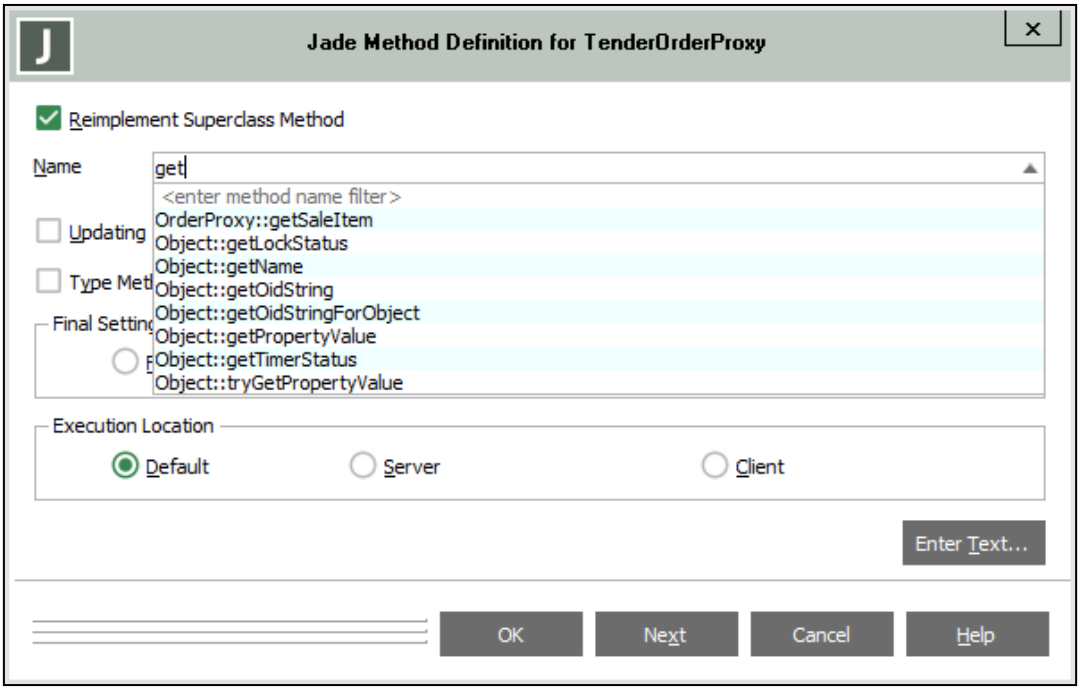


3. Check the **Reimplement Superclass Method** check box if you want to reimplement an existing method in a superclass. This check box is displayed only when you access the dialog from the Class Browser.

   You can reimplement a method only if it is not final in the superclass. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1. A drop-down list containing all superclass methods that are not final is then displayed. Initially, all possible superclass methods are displayed. As the methods are listed by schema and prefixed with the class name, it can make selection of a method difficult.

   The combo box displaying the list of superclass method names is a **Style_DropDown** (**0**) type so that you can enter text. The first entry of <enter method name filter> displayed in the combo box drop-down list is a prompt. It is disabled and cannot be selected.

When you enter text in the **Name** combo box, only those entries that have the entered text (which is case-insensitive) somewhere in the entries in the drop-down list are displayed. (Those that do not contain the specified text are hidden.) As a result, the more text you enter, the shorter the list of possible methods, as shown in the following image.



Select the method in the parent class that you want to reimplement in the current class. A message box then advises you that you are about to reimplement a superclass method, and prompts you to click the **Yes** button if you want to continue. The parent class and the method that you selected for reimplementation are then displayed in the **Name** text box.

4. Specify the name of your new method in the **Name** text box if you do not want to reimplement an existing method in the parent class. (For details about reimplementing a method, see step 10, later in this process.)

Tip From version 2020.0.01 and higher, you can paste text into the **Name** text box so that the paste action displays the initial name in the text box as the name of the new method. The paste action strips all but the first identifier (alpha, numeric, and underscore characters) and inserts that text into the **Name** text box. For example, pasting **get_Customer_Details(formatType : Integer): String;** in the text box results in the following displayed as the method name.

```
get_Customer_Details
```

5. Check the **Updating** check box if the method can modify properties in the object to which it is sent.

   If you do not specify that the method is updating, the method does not modify the receiver object (that is, the object that is executing the method). Any instructions that attempt to update properties of the receiver or any calls to updating methods in the receiver are marked as errors at compile time.

6. Check the **Abstract** check box if the selected class is abstract and the method is to be implemented in all concrete subclasses. An **abstract** method defines a signature with which all subclass implementations must comply.

7. Check the **Protected** check box if you want the **protected** method option added to the signature of your method so that it can be referenced only by methods in the same class or its subclasses or in the same primitive type. (Protected methods are displayed in the browser Methods List with a padlock symbol to the left of the method name.)

   By default, your methods are not protected; that is, they are displayed in the Methods List of the Class Browser or Primitive Types Browser with the public access icon to the left.

8. Check the **Subschema Hidden** check box if you want to specify that the method is available only in the local schema; that is, it is not available for use in any subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

9. Check the **Type Method** check box if you want the **typeMethod** method option added to the signature of your method so that the method is a type method.

   Type methods provide a way of calling a method declared on a type (class, primitive, or interface) without having to have an instance of the type. For more details, see "Type Methods", under "Jade Language Notation" in Chapter 1 of the *Developer's Reference*.

10. If you are defining a method for a subclass of the **JadeWebService** class or one of its subclasses, uncheck the **Web Services** check box if you do *not* want the method defined as a web services method. This check box is enabled only when a web services class is selected in the Class List and it is selected by default. When this check box is checked, the **Web Services Options** command in the Methods menu is enabled.

    For details about web services methods, see "Creating Web Service Methods", in Chapter 11 of the *Developer's Reference*.

11. In the Final Settings group box, select one of the following option buttons for the appropriate final setting, if required.

    - **Final**, if you want to specify that the method cannot be reimplemented in a subclass. The **final** method option makes methods available to other schemas but prevents those schemas from modifying, reimplementing, or circumventing the defined method behavior. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

    - **Subschema Final**, if you want to specify that the method can be extended or reimplemented in its local schema but not in a subschema. The **subschemaFinal** method option makes methods available to other schemas but prevents those schemas from modifying, reimplementing, or circumventing the defined method behavior. For details and an example, see "subschemaFinal Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

    - **Subschema Copy Final** (displayed only when you access the dialog from the Class Browser), if you want to specify that the method cannot be reimplemented in a subschema copy class. For details and an example, see "subschemaCopyFinal Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

12. In the Execution Location group box, select the **Server or Client** option button if you want the **serverExecution** or **clientExecution** method option added to the signature of your method so that the method and all methods subsequently called by this method are executed on the server node or client node, respectively. By default, no execution option is added to the method signature and the node in which the method executes is determined by the location of the calling method. For more details, see "Method Options", in Chapter 1 of the *Developer's Reference*.

13. Click the **Enter Text** button if you want to specify or maintain descriptive text for the Jade method as part of the definition or maintenance of the method. For details, see "Specifying Text for a Schema Element", in Chapter 3.

Tip You can also specify descriptive text for the Jade method at any time, by selecting the **Text** command from the Methods menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

14. Click the **OK** button or the **Next** button.

    If the schema is versioned and a method with the specified name added in one schema context (for example, the current version) exists in the other context (for example, the latest version), a message box advises you that the method exists in the other context and that the methods in both contexts will be linked. If linking, you have the option of copying the existing source from the corresponding method in the other context or of starting with an empty method template.

    Click the:

    - **Cancel** button, to abort creation of the method in the context to which you added it.
    - **Yes** button, to create the method, copying the method source from the other context.
    - **No** button, to create the method without copying the existing method source from the other context.

A method template is then displayed in the editor pane of the Class or Primitive Types Browser.

Note If a method template has been defined for all new methods in your Jade development database or for your own Jade methods, the defined method template is displayed in the editor pane. For details, see "Maintaining Text Templates", in Chapter 2.

Modify the template by adding Jade code to perform the required operations, and then compile it. (For more details, see "Compiling Methods", later in this section, and "Method Options", in Chapter 1 of the *Developer's Reference*.)

Tips If you are unsure of the syntax of a Jade instruction, specify the instruction in the appropriate place in your method and then press Ctrl+S. The syntax for that instruction is then displayed in the method, to enable you to specify the appropriate values. For example, if you specify **foreach;** and then press Ctrl+S, the following is displayed:

```
foreach identifier in (collection | expression to expression
    [step expression]) [reversed] [where expression] do
    [ : label]
    optionalStatementList;
endforeach [label];
```

Alternatively, to obtain online help for a Jade-supplied class, method, property, primitive type, instruction, or method option on which the caret is positioned in the editor pane, press F1. The online help topic for that element is then displayed. (See also "Using Bubble Help in the Editor Pane", later in this section and "Displaying Bubble Help in Browser Lists" under "Using the Class, Primitive Types, or Interface Browser", in Chapter 3.)

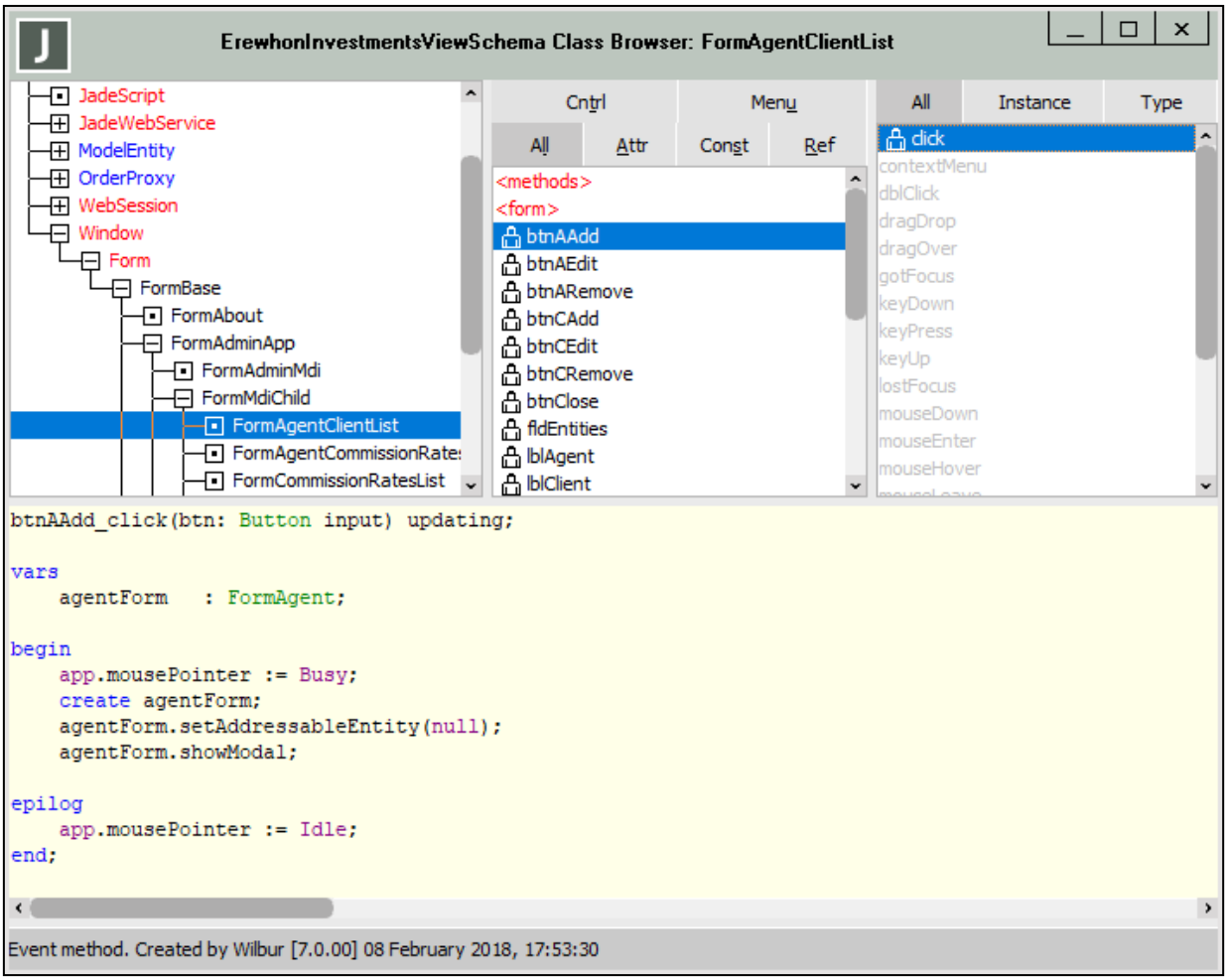
For details about creating a mapping method for a property selected in the Properties List of a hierarchy browser, see "Adding a Method Mapping for a Property", later in this chapter.

To define a Jade event method

1. In the Properties list of the Class Browser window, click on the form or control for which the event method is to be defined. A list of valid events for the selected control is then displayed.
2. Click on the required event in the Event list.

A template for that method is then displayed in the editor pane of the Class Browser.

The following image shows an example of an event method.



Modify the template by adding Jade code to perform the required operations, and then compile it. (For more details, see "Compiling Methods", later in this subsection and to "Jade Language Reference", in Chapter 1 of the *Developer's Reference*.)

Notes Event methods are valid only for forms or controls. For controls, the concatenation of the property name and method name cannot exceed 100 characters. When this maximum length is exceeded, the event method is not displayed and a warning is displayed on the status line.

If a method template has been defined for all new methods in your Jade development database or for your own Jade methods, the defined method template is displayed in the editor pane. For details, see "Maintaining Text Templates", in Chapter 2.

Adding Conditions to Classes or Primitive Types

A conditional expression is a restricted form of a method. The major use of a condition is as a constraint, which is used to maintain automatic inverse references when a specified condition is satisfied. When defining conditions, note the following.

- You can specify constraints only on automatic inverse references and not on manual or manual/automatic inverse references.

  When the manual side of the inverse reference is set, the condition used as the constraint is evaluated and the automatic inverse is set only if the condition is **true**. If the automatic reference is a **Collection** type, the condition is applied to the members of the collection.

- A condition used as a constraint cannot have parameters.

- You cannot reimplement conditions defined in superschemas or superclasses.

- You can add a condition only to Instance methods.

- Only **if** and **return** instructions can be used in condition methods.

For examples of conditions and constraints, see "Condition Examples", in the following subsection.

Notes Before you define a new condition, you must select the class or primitive type to which the method is to be added.

Conditions cannot involve reference properties.

To add a condition to a class or primitive type

1. In the Class List of the Class Browser or the Primitive Types List of the Primitive Types Browser, click on the class or primitive type for which the condition is to be added.
2. Select the **New Condition** command from the Methods menu. (This command is enabled only when the **All** or **Instance** sheet has focus.)

   The Add Condition dialog is then displayed.
3. Specify the name of your new condition in the **Name** text box. The name value must start with a lowercase character and the name must be unique to the class or primitive type to which it is being added.
4. Check the **Protected** check box if you want the **protected** condition option added to the signature of your condition so that it can be referenced only by methods or conditions in the same class or its subclasses.

   By default, conditions are not protected.

   Note A padlock symbol is not displayed to the left of the condition name in the browser Methods List. To determine if an existing condition is protected, select the condition in the Methods List and see whether the **protected** option is displayed in the condition signature in the editor pane.

   ---
5. Check the **Subschema Hidden** check box if you want to specify that the condition is available only in the local schema; that is, it is not available for use in any subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.
6. Click the **Enter Text** button if you want to specify or maintain descriptive text for the condition as part of the definition or maintenance of the condition. For details, see "Specifying Text for a Schema Element", in Chapter 3.

Tip You can also specify descriptive text for the condition at any time, by selecting the **Text** command from the Methods menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

7. Click the **OK** button or the **Next** button.

A template for the condition is then displayed in the editor pane of the browser window.

The error symbol is displayed to the left of the condition name in the Methods List until you have specified any parameters that you require and the *boolean-expression* return value for the condition, and then compiled it.

If a method template has been defined for all new methods and conditions in your Jade development database or for your own Jade methods and conditions, the defined method template is displayed in the editor pane. For details, see "Maintaining Text Templates", in Chapter 2.

Note When adding a condition method using a template, the text template strips out the **vars** and **epilog** sections if they had been included in your template.

A condition has the following syntax.

```
condition-name([parameters]): Boolean condition;
[constants
    constant-declarations]
begin
    condition-instructions;
end;
```

Only **if** and **return** instructions can be used in condition methods. A returned *boolean-expression* can contain only references to the properties of the class and calls to other conditional expressions.

Notes This syntax (with the exception of the optional *parameters* and *constant-declarations*) is automatically displayed with the **return** instruction in the editor pane when you click the **OK** button in the Add Condition dialog unless a method template is defined for your Jade Platform development environment, in which case the editor pane display is determined by the template.

A condition cannot use array indexing ([*index*]) or dictionary indexing ([*key*, *key*, ...]).

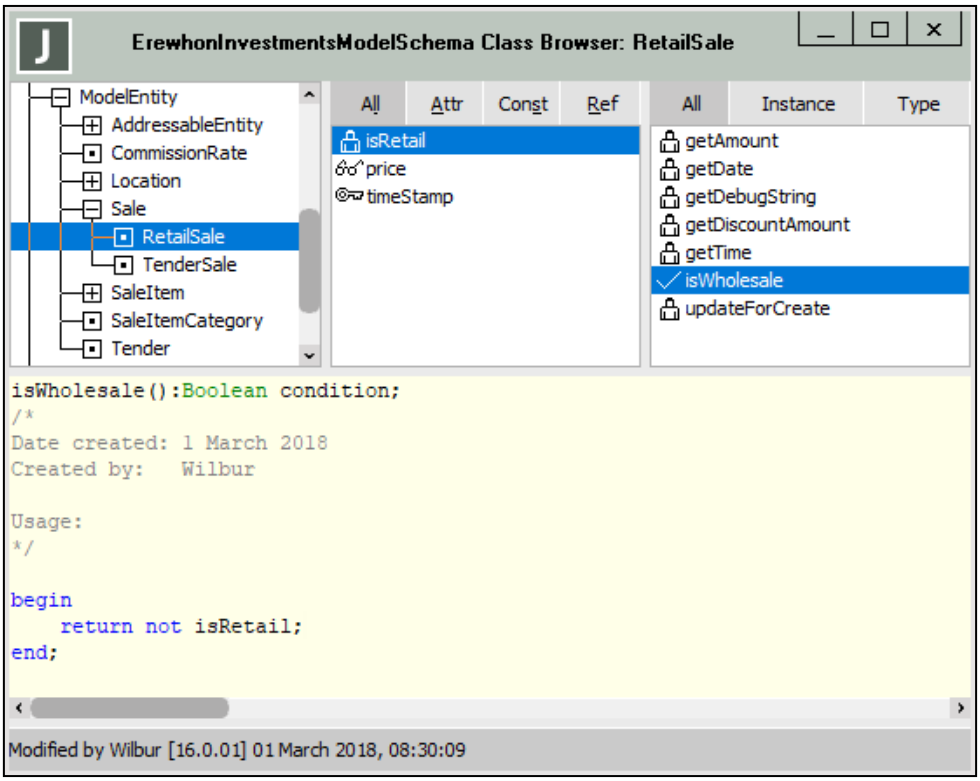
You can define optional parameters only for conditions that are not constraints, and the parameters must be constant values. For more details, see "Parameters" under "Jade Language Syntax", in Chapter 1 of the *Developer's Reference*.

Modify the template by specifying the boolean expression that you require following the **return** instruction to perform the required operations or replace the return instruction with the appropriate **if** instruction.

When you have specified the boolean expression to return or the **if** condition and any constant parameters that you require for a condition that is not a constraint, compile the condition. (For details, see "Compiling Methods", later in this section.)

When you have compiled the condition, the condition symbol is then displayed to the left of the condition name in the Methods List.

The following image shows an example of a condition.



Condition Examples

Use a constraint to specify that automatic references can be set only if the referenced object meets a specified condition. (You can specify constraints only on automatic references and you cannot specify parameters.)

In the examples in this subsection, the reference properties listed in the following table are defined on the **Company** class, with an inverse **myCompany** reference.

Reference	Defined Inverse
allSalaried	**EmployeeDictionary**, where **isSalaried** of **Employee** explicitInverse;
allWaged	**EmployeeDictionary**, where **isWaged** of **Employee** explicitInverse;
allContractors	**EmployeeDictionary**, where **isContractor** of **Employee** explicitInverse;
allTempStaff	**EmployeeDictionary**, where **isTempStaff** of **Employee** explicitInverse;
allEmployees	**EmployeeDictionary**, explicitInverse;

The value of the **Inverse Not Required** check box on the extended Define Reference dialog determines whether exceptions are raised when no inverse is set, as none of the conditional inverses is set. Both the compatibility and constraints are applied.

An exception is raised when the **Inverse Not Required** check box is unchecked (the default) and one of the following applies.

- No inverse is set because the type of the object is not compatible with any of the inverse references
- No inverse is set because the constraint fails

If no inverse is set and the **Inverse Not Required** check box is checked, no exception is raised if the compatibility or constraint fails. Similarly, no exception is raised if the last inverse is removed as the result of change to a property and the **Inverse Not Required** check box is checked.

The following condition examples assume that an **Employee** class has an integer value **payFrequency** property, a string value **status** property, and a **myCompany** property of type **Company**. The **compareUpper** condition is a condition defined on the **String** primitive type.

```
isWaged(): Boolean condition;
begin
    return payFrequency = 7;
end;

isSalaried(): Boolean condition;
begin
    return not isWaged;
end;

isTempStaff(): Boolean condition;
begin
    return status.compareUpper("T");
end;

isContractor(): Boolean condition;
begin
    return not isTempStaff;
end;
```

Conditions on Primitive Types

The following examples are of an **isMultipleOf** condition method defined on the **Integer** primitive type and an **isEqual** condition method defined on the **String** primitive type.

```
isMultipleOf(d : Integer): Boolean condition;
begin
    return self div d = 0;
end;

isEqual(name: String): Boolean condition;
begin
    return self = name;
end;
```

Using these **isEqual** and **isMultipleOf** condition methods, the following example is a **primCond** condition defined on a **Cls1** class with an **int** attribute of primitive type **Integer**.

```
primCond(): Boolean condition;
begin
    return str.isEqual("END") or int.isMultipleOf(9);
end;
```

This **primCond** condition can be used as a constraint because it has no parameters, which applies even though it calls conditions that have **constant** value parameters.

Using the Method Status List Browser

Use the Method Status List Browser to display methods and class constants that are in error or require compilation, and amend or compile the error, if required.

The following statuses can be displayed.

- Uncompiled methods and class constants (displayed by default)
- Methods and class constants in error (displayed by default)
- Compiled methods and class constants

For details about changing the default status list options, see "Maintaining Status List Options", in Chapter 2. For details about compiling interface methods, see "Compiling an Interface Method", in Chapter 14.

You can view the status of a method or methods from the Schema, Class, Primitive Types, or Interface Browser, by selecting the schema, class, primitive type, or interface whose methods that match your status criteria you want to display. (For example, if you access the Method Status List Browser from a schema selected in the Schema Browser, all methods and class constants that match your status criteria are displayed.)

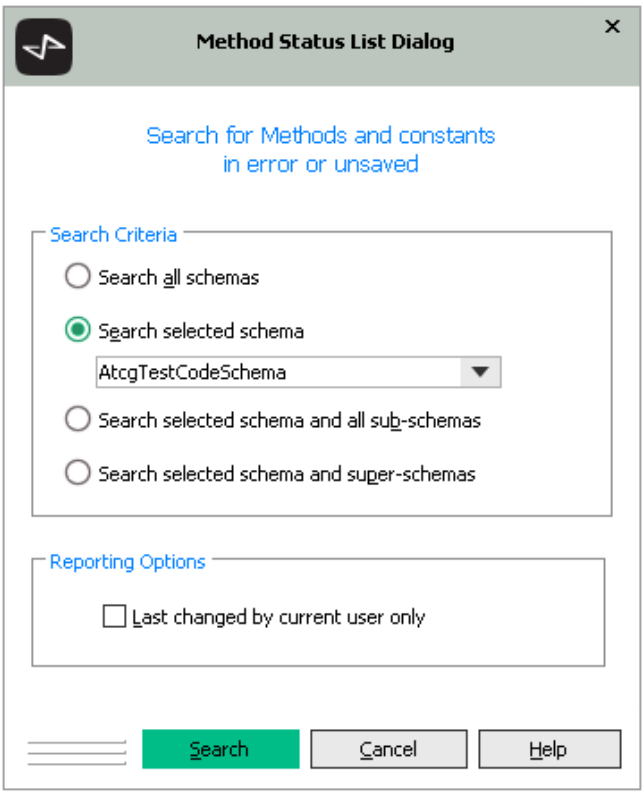
In addition, from version 2018.0.01, you can select:

- Another schema.
- All schemas.
- The selected schema and all subschemas.
- The selected schema and all superschemas.

To view methods and class constants requiring compilation

- Select the **Status List** command from the Browse menu.

The Method Status List Dialog is then displayed, with the **Search selected schema** option selected by default.



1. In the Search Criteria group box, select the schema in which to search.
2. Check the **Last changed by current user only** check box if you want to search only for constants and methods that were last changed by you. (By default, this check box is unchecked.)
3. Click the **Search** button.

A progress dialog is displayed while the classes are scanned for methods and class constants that require compilation.

If no methods or class constants match your status criteria, a message box is then displayed, advising you that there is no output to view. If methods and class constants are located that match your search criteria, the Method Status List window is displayed after a few seconds, listing all methods and class constants that require compilation or are in error.

To view the code for a listed method or class constant

- Click on the required method or class constant.

The code for that method or class constant is then displayed in the editor pane. You can make changes to the code, if required, or compile a selected method or class constant from the Method Status List window, by positioning the caret anywhere in method logic in the editor pane and then pressing F8 or selecting the **Compile** command from the Methods menu. The displayed method or class constant is then compiled.

If an error is detected, the first error that is detected is highlighted and the corresponding message is displayed in the status line, to enable you to amend the logic. When the method or class constant is compiled and no errors are detected, the message *Compilation complete - no errors* is displayed in the status line, and the status list is refreshed, as appropriate.

Notes To display a separate editor pane, double-click on the required method or class constant.

You can copy the methods in the pane at the top of the Method Status List Browser to the clipboard (for example, if you want to send an email message containing the list of methods that did not compile during a schema load), by selecting all of the methods and then selecting the **Copy** command from the Edit menu.

Alternatively, you can view code for a method or class constant that contains compilation errors by selecting a method in the Methods List whose compilation error icon to the left of the method or class constant indicates that it contains one or more compilation errors.

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods".

To display the source of a method selected in the methods list in a separate window

- Double-click on a method or double-click a method entry while holding down the Shift key.

That method is then displayed in a separate window. This enables you to fix a compile error and keep the method displayed. (When you fix a compile error in the editor pane of the Method Status List Browser and press F8, the method is removed from the list so is no longer displayed.)

To view compiler error, warning, and information messages for a listed method

- Click on the required method and then press F8.

The Compiler Output Viewer is then displayed, if you have configured it in your preferences. For details, see "Using the Compiler Output Viewer", later in this chapter.

Examples of Jade Methods

The following is an example of a method for a **Form** subclass.

```
getRowsVisibleInTable(): Integer;
vars
    rows : Integer;
    totalHeight : Integer;
begin
    totalHeight := theTable.rowHeight[1];      // the fixed row
    rows := 1;
    while totalHeight < theTable.height do
        rows := rows + 1;
        if theTable.rows < rows then
            theTable.rows := rows;
        endif;
        totalHeight := totalHeight + theTable.rowHeight[rows];
    endwhile;
    rows := rows - 1;
    return(rows);
end;
```

The following is an example of a method that calculates the average cost of shares.

```
averageCost(set: Boolean; value: Decimal io) mapping;
/*
Date:   10 August 2001
User:   wilbur
Method: averageCost
```

```
*/
vars
    totalCost   : Decimal[14,4];
    totalShares : Decimal[14,4];
    tran        : Transaction;
begin
    if not set then
        foreach tran in self.myTransactions do
            sharedLock(tran);
            totalCost := totalCost + (tran.shares * tran.price);
            totalShares := totalShares + tran.shares;
            unlock(tran);
        endforeach;
        value := totalCost / totalShares;
    endif;
end;
```

Using Bubble Help in the Editor Pane

Use bubble help in the editor pane to quickly:

- Locate and insert the required constant, property, or method for a class or the constant or method for a primitive type or interface
- Display the signature of a method and position the display on the editor pane while you specify the required parameters

  When the mouse is hovered over the method name at the start of the source, the AutoComplete bubble help for the method includes the schema and class to which the method belongs.
- View descriptive text specified for the schema element

Note The bubble help information may not be displayed correctly if there is a syntax error earlier in the method source. (See also "Displaying Bubble Help in Browser Lists" under "Using the Class, Primitive Types, or Interface Browser", in Chapter 3.)

To display the properties provided by a class

1. Press Ctrl+1 (that is, the control key and the numeric **1** key) when the caret is positioned at the end of a class. The list of properties provided by that class and its superclasses is then displayed in a combo box below the class.
2. Select the property that you want to insert following the class.
3. Press the Enter key or the Tab key to insert the selected property after the caret position. (The **.** dot operator notation is also inserted if it is required.)

If you view the displayed properties and then decide not to insert a property after the caret position, press Esc to return focus to the editor pane.

When bubble help is shown for a property and the type of that property is a **Collection**, the displayed details include the membership of the collection and the keys, if it is a **MemberKeyDictionary**; for example:

```
Name: allMyBigData (19)
Class: OdbcRoot ()
Type: DictBigData
        ------- DictBigData Collection Class Details -------
        Membership: BigData
```

```
        Member Keys:
              transactionDate (Date) ascending, case sensitive, sort order:
Binary
Access: public
SubId: 18
Ordinal: 19
non-virtual

Inverses:
        myOdbcRoot of OdbcData
Update Mode: Automatic
Relationship Type: parent
```

To display the methods provided by a class, primitive type, or interface

1. Press Ctrl+2 when the caret is positioned at the end of a class, primitive type, or interface. The list of methods provided by that class, superclasses, primitive type, or interface is then displayed in a combo box below the class, primitive type, or interface.
2. Select the method that you want to insert following the class, primitive type, or interface.
3. Press the Enter key or the Tab key to insert the selected method after the caret position. (The **.** dot operator notation is also inserted if it is required.)

If you view the displayed methods and then decide not to insert a method after the caret position, press Esc to return focus to the editor pane.

To display the signature of a method

1. Press Ctrl+1 (that is, the control key and the numeric 1 key) or Ctrl+2 when the caret is positioned at the end of a method name or immediately inside the opening parenthesis of a method call.

   The signature of that method is then displayed in a window below the caret position. If the caret is positioned elsewhere (for example, after a comma between parameters), press Ctrl+5 to display the signature of the last method before the caret position.

   When the mouse is hovered over the method name at the start of the source, the AutoComplete bubble help for the method includes the schema and class to which the method belongs.
2. Specify the appropriate parameters. (The signature remains displayed while you specify your required values.)
3. When you have specified the method or you no longer want to view the signature, press Esc.

The signature is no longer displayed and focus returns to your editor pane.

To display the constants provided by a class, primitive type, or interface

1. Press Ctrl+3 when the caret is positioned at the end of a class, primitive type, or interface. The list of constants provided by that class, primitive type, or interface is then displayed in a combo box below the class, primitive type, or interface.
2. Select the constant that you want to insert following the class, primitive type, or interface.
3. Press the Enter key or the Tab key to insert the selected constant after the caret position. (The **.** dot operator notation is also inserted if it is required.)

If you view the displayed constants and then decide not to insert one after the caret position, press Esc to return focus to the editor pane.

To display values for a collection class or a String or Binary primitive type

1. Position the caret immediately after an opening bracket (**[**) operator immediately after a valid object type.

   An error message is displayed in the status line if the object type cannot be resolved (for example, bracket operators are not valid for that object).

2. Press Ctrl+1, Ctrl+2, or Ctrl+3.

   Comma-separated keys of that class are inserted after the caret position if the class is a **Dictionary** class or subclass, the text **indx]** is inserted if the class is an **Array** class or subclass, or the text **startPos : forLength]** is inserted if the class name is a **String** or **Binary** primitive type.

3. Make any changes that you require to the inserted values.

To display the local variables or parameters of the current method

1. Press Ctrl+4 when the caret is positioned anywhere in an editor (method source) window.

   A combo box filled with the local variables and parameters of the current method is then displayed. (The combo box is not displayed if there are no local variables or parameters for the current method.)

2. Select the local variable or parameter that you want to insert into the current method.

3. Press the Enter key or the Tab key to insert the selected local variable or parameter after the caret position.

Saving Methods as Another Name

Use the **Copy** command from the Methods menu to save an existing method as a new name.

The Copy Method As dialog (displayed when you select an existing method in the Methods List and then select the **Copy** command from the Methods menu) displays the name of the existing method that is being copied in the **New Name** text box, and the entire method name is selected. Specify the new name that you require for the current method.

If some or all of the method is selected in the editor pane, the **Copy Highlighted Text Only** check box is enabled, which you can check if you want to copy only the selected text to the new method.

Method validation rules apply.

A new method (containing all code defined in the original method or the selected code if you checked the **Copy Highlighted Text Only** check box when text was selected) is then created with your specified name.

Defining Translatable Strings from the Editor Pane

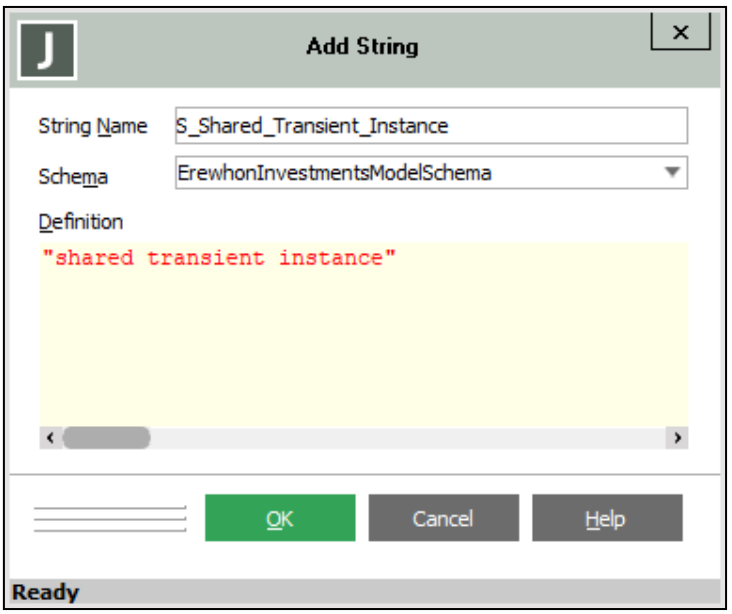
You can define strings for translation from the editor pane.

To define a translatable string

1. Select the text that is to be translated.

2. Press F6.

The Add String dialog, shown in the following image using the selected comment from the previous method example, is then displayed.



The **String Name** text box displays a string name manufactured by Jade from the selected text by:

- Removing illegal characters
- Truncating the name to 100 characters
- Replacing spaces with underscore characters
- Changing the first character and each character following an underscore character to uppercase

3. If you want to define the translatable string in a superschema of the current schema, select the appropriate superschema from the list in the **Schema** combo box.
4. You can change the string name, if required. The text selected in the editor pane is displayed in the **Definition** text box and can be modified to meet your requirements.

When you click the **OK** button, the translated string is displayed in the method with a preceding dollar sign (**$**), as shown in the following example.

```
$The_Fixed_Row
```

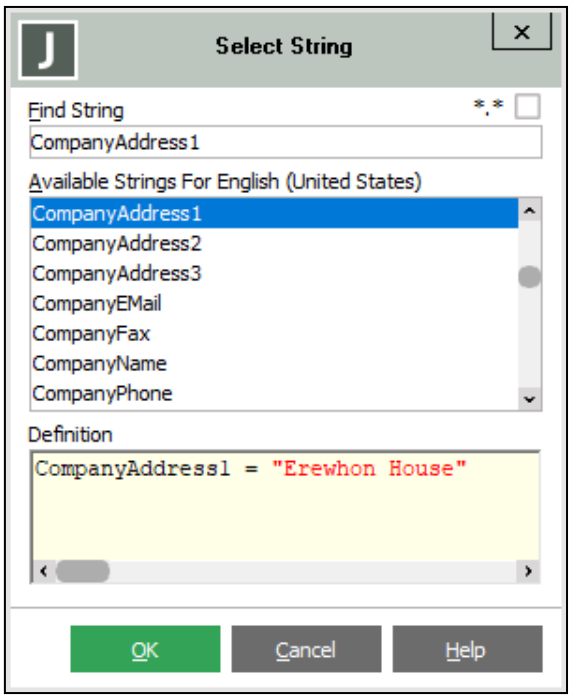
Inserting a Translatable String into an Existing Jade Method

You can insert an existing translatable string into a Jade method displayed in the editor pane.

» To insert an existing translatable string into the current method

1. Position the caret at the position in the method at which the string is to be inserted.
2. Press Shift+F6.

3. The Select String dialog, shown in the following image, is then displayed. The dialog lists all translatable strings that are available to the current schema.



   This form is also displayed when you select a property in the Translatable Property Browser in the Jade Painter. (For details, see "Using the Translatable Property Browser", in Chapter 5.)

4. In the **Available Strings** list box, select the string that is to be inserted into the Jade method or specify the name of the string in the **Find String** text box. The value specified in the **Find String** text box acts as the starting key to locate an available string in the current schema.

5. If you want to perform a wildcard search, check the ***.*** check box at the upper right of the dialog and then press Enter. All translatable strings in the current schema that contain the text specified in the **Find String** text box in its name or its value are then displayed in the **Available Strings** list box. The search is case-insensitive.

6. When you have selected the translatable string that you want to insert, click the **OK** button.

The name of the string (rather than the string definition itself) is then inserted at the current caret position into the Jade method displayed in the editor pane. For example, **$The_Fixed_Row** is inserted into the Jade method if you selected the **The_Fixed_Row** string.

Note When you use a translatable string in a Jade method, the string must be prefixed with a dollar sign (**$**); for example:

```
igfFrame.myOutln.IGOutln.addXLbl(time.userFormat($PlainTime));
```

Jade Method Syntax Coloring

Different parts of your methods are displayed in different colors (for example, keywords, or Jade instructions, are displayed in blue and string literals in red by default). The colors that are displayed in methods in the editor pane are specified for your site when Jade is installed.

You can specify your own colors for the display of methods in the editor pane on your workstation. For details about changing the default colors for your methods, see "Setting User Preferences", in Chapter 2.

Compiling Methods

When you have defined or modified a method, you must compile it before you can execute it. (You cannot compile system methods.)

To compile the current method, perform one of the following actions

- Press F8.
- Select the **Compile Method** command from the Methods menu. Alternatively, right-click on the method name in the Methods List of the Class Browser or Primitive Types Browser and then select the **Compile Method** command from the popup menu that is displayed.

The method is then compiled. The first syntax error that is detected is highlighted and the corresponding error message is displayed in the status line, to enable you to amend the code, as required, before compiling and executing your method again.

Notes Only the first error in the logic is reported. When that error has been corrected and the method recompiled, any following error is reported individually, or the compilation completed.

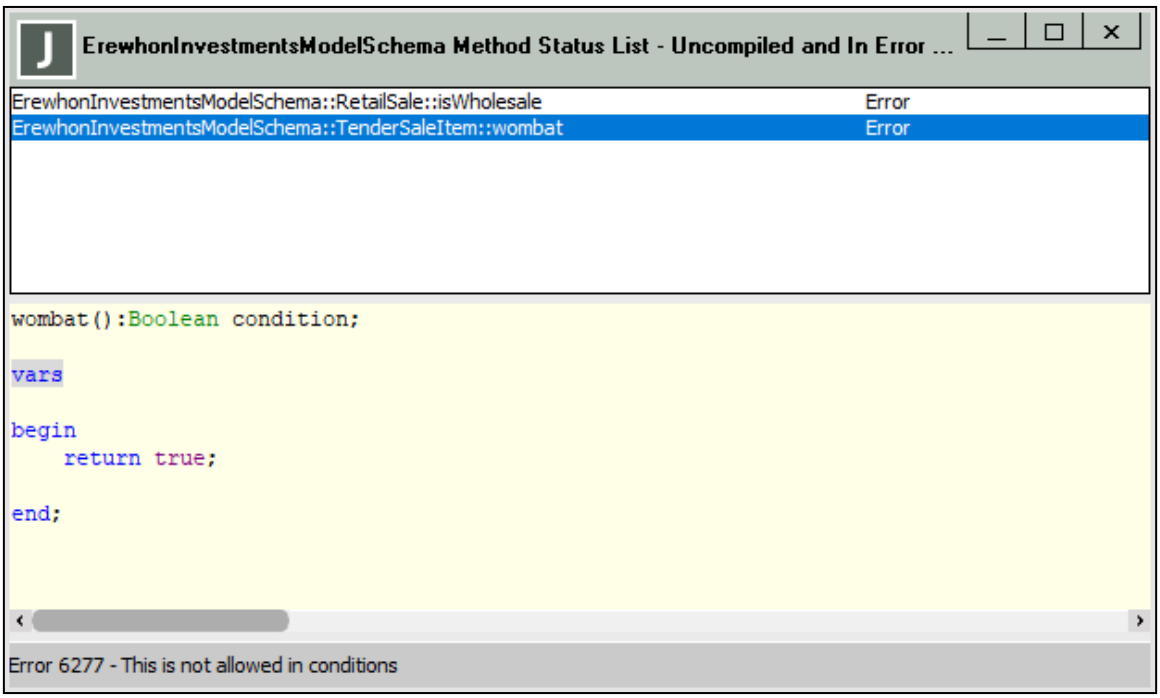
If configured in your preferences, the Compiler Output Viewer is automatically displayed when you compile a class or method that generates error, warning, or information compiler messages. For details, see "Using the Compiler Output Viewer", later in this chapter.

When the method is compiled and no errors are detected, the message *Compilation complete - no errors* is displayed in the status line.

To view methods that require compilation, select one of the following Browse menu commands

- The **Status List** command, to view all methods for all development environment users.
- The **Status List for Current User** command, to view all uncompiled methods that were last changed by you.

A progress dialog is then displayed while the classes are scanned for methods that require compilation. If any method requires compilation or contains errors, the Methods Status List dialog, shown in the following image, is then displayed.



Alternatively, you can view code for a method that contains compilation errors by selecting a method in the Methods List whose compilation error icon to the left of the method indicates that it contains one or more compilation errors.

For details about compiling all methods in a class, primitive type, or interface, see "Compiling All Methods Defined in a Class, Primitive Type, or Interface", in Chapter 3. For details about recompiling all methods in your Jade database that have not yet been compiled or are in error, see "Recompiling All Methods in the Database", in the following subsection. See also "Compiling an Interface Method", in Chapter 14.

Recompiling All Methods in the Database

You can use the **jadclient** non-GUI application with the following arguments to recompile all methods in your Jade database. (For details about automating a non-GUI application, see "Running a Non-GUI Application", in Chapter 1 of the *Runtime Application Guide*.)

```
jadclient path=database-path
          [ini=jade-initialization-file]
          schema=RootSchema
          app=JadeRecompileAllMethods
          startAppParameters
          [command-line-arguments]
```

The command line argument that you can define after the **startAppParameters** parameter in ***command=value*;** format is as follows.

- includeAlreadyCompiled=true|false

The following example recompiles uncompiled methods and methods that are in error.

```
jadclient path=c:\jade\system ini=c:\jade\system\testjade.ini schema=RootSchema
app=JadeRecompileAllMethods
```

To force a recompilation of all previously successfully compiled methods, specify the **includeAlreadyCompiled** command line argument with a value of **true**. (The **includeAlreadyCompiled** argument defaults to **false**.)

The following example recompiles all methods that had previously successfully compiled.

```
jadclient path=c:\jade\system ini=c:\jade\system\testjade.ini schema=RootSchema
app=JadeRecompileAllMethods startAppParameters includeAlreadyCompiled=true
```

The **JadeRecompileAllMethods** application returns a non-zero exit code 1183 (*Uncompiled or in error methods remain*) if methods could not be compiled or if any methods did not compile without error. The output is logged to the **JadeRecompileAllMethods.log** file in the default **log** directory. The log file contains details of the methods that were not successfully compiled.

The following information is output to **stdout**.

- Number of methods compiled
- Number of methods that are in error
- Number of methods that could not be compiled because there was no source

If methods are in error, you must access the Jade Platform development environment Methods Status List Browser to view these errors and make the appropriate changes.

Note The **JadeRecompileAllMethods** application does not compile methods that are checked out.

Using the Compiler Output Viewer

The Compiler Output Viewer enables you to view error, warning, and information messages that occur when you compile schemas, classes, and methods in the Jade Platform development environment.

For details about setting up the Compiler Output Viewer, see "Maintaining Compiler Output Options", in Chapter 2.

To open the Compiler Output Viewer

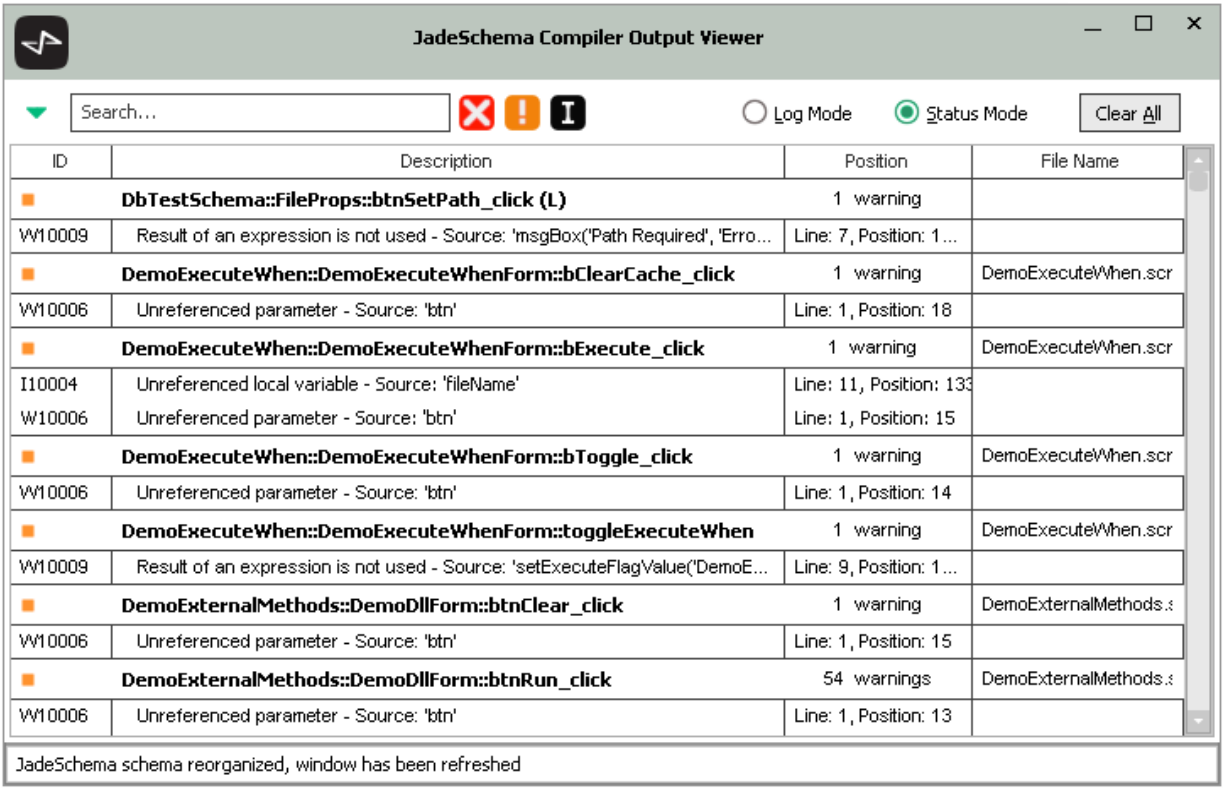
- Select the **Compiler Output** command from the File menu.

The Compiler Output Viewer is then displayed.

Note If configured in your preferences, the Compiler Output Viewer is automatically displayed when you:

- Compile a class or method that generates compiler messages.
- Load a schema that contains classes or methods that generate compiler messages. (This could cause your schema or schemas to load slowly if they generate a large number of messages.)

The following image shows an example of Compiler Output Viewer with errors, warnings, and information messages displayed after loading multiple schemas.



The following columns are displayed.

- **ID**, which displays the compiler message identifier.
  - Error message identifiers are prefixed with **E** and display a red square at the left.
  - Warning message identifiers are prefixed with **W** and can display an orange square at the left.
  - Information message identifiers are prefixed with **I**.
- **Description**, which displays the method that generated the compiler message, the number of warnings or errors generated, and the message details.

  The method is displayed as ***schema-name::class-name::method-name*** and displays **(L)** at the right if there are any versioned entities.

  Note For details about the messages, see "Compiler Output Messages", later in this chapter.
- **Position**, which displays the line position in the source code that generated the compiler message.
- **File Name**, which displays the name of the file that was loaded.

By default, compiler messages are listed in the order in which they are generated.

Tip To change the sort order of compiler messages in the table, click the header for the column that you want to sort into ascending or descending order. For example, click the **Description** header to sort compiler messages into ascending or descending order.

Use the controls at the upper left of the Compiler Output Viewer to filter your view of the compiler messages.

To filter specified compiler messages

1. Click the **Filter Options** button, represented by a green triangle. The Suppressed Warnings options are then displayed.

   Note Messages displayed in red text are suppressed by your administrator and you cannot change them.

2. Check or uncheck the check box of each message that you want to hide or display on the Compiler Output Viewer and then click the **Update** button. Alternatively, to suppress all compiler messages, click the **Suppress All** button.
3. To restore all available compiler messages, click the **Reset** button.
4. To hide the Suppressed Warnings options, click the **Filter Options** button or click elsewhere within the Compiler Output Viewer.

To search for specified text in the Compiler Output Viewer

- In the **Search** text box, enter the text that you want to find in the table. As you type, the table contents are filtered to display only those entries that contain text matching your input.

To restore the Compiler Output Viewer table entries, delete your search text.

When you open the Compiler Output Viewer, errors, warnings, and information messages are displayed, by default.

To display or hide warning, error, or information messages

- Click the toggle button of the message type that you want to hide or display.

Toggle buttons are described in the following table.

Button	Toggles the display of...
	Error messages
	Warning messages
	Information messages

A toggle button that is greyed-out indicates that the message type is not displayed in the table. Toggle buttons also represent the color-coding of the compiler messages that are displayed; for example, error message details are displayed with a red square at their left.

Use the option buttons at the upper right to change how information is displayed in the table. Select the:

- **Log Mode** option button, to list the compiler messages in the order in which they were generated.
- **Status Mode** option button, to list the compiler messages by ***schema-name::class-name::method-name*** row header. When you open the Compiler Output Viewer this option button is selected, by default.

If you select a table entry that spans multiple rows, the selected row is highlighted with a blue background and any associated rows are highlighted with a light-purple background (in light mode) or navy (in dark mode).

To view the code of a method

- Double-click on the required method.
- Right-click on the required method and then select the **Navigate To *schema-name::class-name::method-name*** command from the popup menu that is displayed.

The appropriate browser is then displayed for the method; for example, the Class Browser, Primitive Types Browser, or Interface Browser. The code that generated the compiler message is highlighted in the Editor of the browser that is displayed. For control or menu event methods, the associated control or menu property is selected in the **Cntrl Events** or **Menu Events** sheets, respectively, in the browser.

You can make changes to the code, if required, or compile a selected method, by positioning the caret anywhere in method logic in the editor pane and then pressing F8 or selecting the **Compile** command from the Methods menu. The displayed method is then compiled.

To copy one or all of the Compiler Output Viewer table entries to the clipboard

- To copy one table entry only, right-click on that entry and then select the **Copy Row** command from the popup menu that is displayed.
- To copy all table entries, right-click in the table area and then select the **Copy All** command from the popup menu that is displayed.

The selected content is then copied to the clipboard, to enable you to paste it into another application (for example, **Notepad**).

To clear the table on the Compiler Output Viewer

- Click the **Clear All** button.

Applies to Version: 2022.0.01 and higher

Compiler Output Messages

The following table lists the messages that can be displayed on the Compiler Output Viewer.

ID	Description
10001	Same name as existing class or type
10002	Same name as existing class or type constant
10003	Same name as existing global constant
10004	Unreferenced local variable
10005	Unreferenced local constant
10006	Unreferenced parameter
10007	Unreachable code
10008	Not all code paths return a value
10009	Result of an expression is not used
10010	Redundant type guard

ID	Description
10011	Same name as existing class property
10012	Referencing a deprecated feature
10013	Referencing an unpublished feature
10014	Referencing an unpublished type
10015	Invalid Use of InstanceType
10501	Deprecated feature

Renaming Methods

You can change the name of an existing user-defined method in the current schema. (You cannot rename system methods.)

» To rename a user-defined method

1. In the Methods List of the Class Browser, Primitive Types Browser, or Interface Browser, select the method that you want to rename.
2. Select the **Rename** command from the Methods menu. The Rename Method dialog is then displayed.
3. In the **New Name** text box, specify the new name for your selected method.

   The name must start with a lowercase character and must be unique to the class, primitive type, or interface to which it belongs.

   This dialog enables you to change only the name of the method. You cannot modify parameters, the return type, or options.
4. Click the **OK** button.

   Alternatively, click the **Cancel** button to abandon the operation.

When a method is renamed, Jade performs the following actions.

- Replaces all references with the new name in all methods that referenced the old name.

  Note The method name replacement may not be successful in methods that are in error.

- Recompiles all methods with references to the old method name.
- Renaming a control event method automatically renames all event implementations.

For details about renaming a method selected within the body of a method in the editor pane, see "Renaming or Changing an Entity", later in this chapter.

Using Constructors and Destructors

A *constructor* is a method that is executed every time a new instance of a class is created.

To identify a method as a constructor, name the method **create**. Use a constructor method to set defaults when an object is created, as shown in the following example.

```
create() updating;
begin
    self.bevelInner    := 0;             // none
    self.bevelOuter    := 0;             // none
    self.boundaryWidth := 0;
    self.boundaryBrush := 0;             // black
    self.caption       := null;
    self.borderStyle   := 0;
    self.backColor     := 15269887;      // buff
end;
```

A *destructor* is a method that is executed every time an instance of a specific class is deleted.

To identify a method as a destructor, name the method **delete**. Use a destructor method for cleanup tasks when an object is deleted, as shown in the following example.

```
delete() updating;
vars
    tran : Transaction;
begin
    foreach tran in self.myTransactions do
        sharedLock(tran);
        delete tran;
    endforeach;
end;
```

Notes Constructors and destructors for all superclasses are automatically invoked, to ensure that you do not compromise the behavior of superclasses.

Destructor methods must not have parameters. For details about using a constructor with parameters from version 2018.0.01 and higher, see "create Method Constructors", in the following subsection.

You cannot specify constructors and destructors for an application class or a global class (that is, a subclass of the **Application** class or **Global** class, respectively).

Constructors are called starting with the highest superclass implementation and continuing down the hierarchy to the current class. Destructors are executed in the reverse order, starting with the current class and continuing up through the hierarchy to the highest superclass. To ensure that a form has no orphaned children when deleting a window, the lowest child in the hierarchy is deleted first, followed by the next lowest child, and so forth up the hierarchy until the parent is deleted. You therefore do not need to handle the deletion of a form's children in your code, as the child or children no longer exist when the parent is deleted.

create Method Constructors

A constructor can be implemented with one or more parameters. Jade does not support method overloading, so a class can have one explicit constructor only. However, the number and type of the parameters on the **create** method on any class in the hierarchy can differ.

The syntax of a constructor with parameters is as follows.

```
create(optional-parameters) [::super(required-parameters)] updating;
```

The optional **::super** value indicates that the superclass constructor is invoked with the specified parameters. For this syntax to be used, the **create** method in the superclass must have parameters. If the superclass constructor has parameters, the compiler verifies that the call to the constructor is present, that the number of parameters is correct, and that the types are compatible with the definition of the superclass **create** method. If the superclass constructor does not have parameters, an explicit call is not allowed.

The result of a method call that returns a value of the appropriate type can be provided as the value to a parameter to a superclass constructor. The receiver of the method is a partially initialized instance of the class being created. The constructor parameters for the immediate superclass and current class are not called until after the method has executed. If the method is declared on a superclass and reimplemented on the class of the instance being created, the reimplemented method implementation is called.

Notes Changing the signature of a **create** method results in any methods creating an instance of the class being marked for recompile.

Parameters must be usage **constant** or usage **input**. Usage **io** and usage **output** parameters are not supported.

If a class with a constructor with parameters is exported from a package, the **create** method must also be exported.

Jade does not allow parameters to be declared on the **create** method of classes that derive from any RootSchema class other than **Object**. This restriction is required because when Jade instantiates instances of these classes, the parameters values are not always known. For example, **Form** and **Control** instances are created when a form is displayed, and exclusive collection instances are instantiated when the collection is populated.

Applies to Version: 2018.0.01 and higher

See also "Using Constructors and Destructors", earlier in this chapter.

Constructor Method Example

The hierarchy in the examples of the **create** method with parameters syntax is as follows.

```
A - create() // explicit
  \
    B - create(string: String) // explicit
      \
        C - create() // explicit
```

The following, based on the above hierarchy, are examples of the new method syntax.

```
A - create() updating;
B - create(string: String) updating;
C - create() ::super("my name") updating;
```

When an instance of **B** is created, **A::create()** is called then **B::create()** is called with the parameters provided in the **create** instruction.

When an instance of **C** is created, **A::create()** is called and then **C::create()** is called. **B::create()** is called explicitly by **C::create()**, before the body of the **C::create()** method is executed.

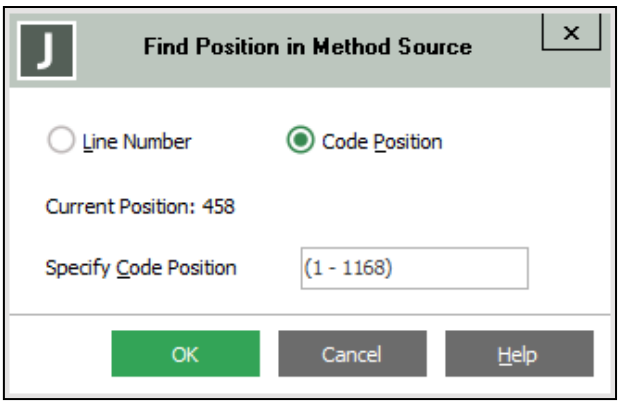
Finding a Method Source Position

You can locate a specific position in a method source; for example, when you have used the **Remove Sources** command from the **Admin** menu to remove method source code from applications in a user-defined schema released to a third-party site and you want to determine the position at which an exception occurred at that site.

To find a method source position

1. In the Methods List of the Class Browser or Primitive Types Browser, select the method whose position in the source code you want to locate.
2. Select the **Find Code Position** command from the Methods menu. The Find Position in Method Source dialog is then displayed, with the **Code Position** option button selected by default.

   The current caret position is displayed, and the range of code positions is displayed in parentheses in the **Specify Code Position** text box as hint text, as shown in the following image.



3. In the **Specify Code Position** text box, specify the code position you want to locate (for example, the position indicated in the exception handler in the deployed application from which source code has been removed).

   This position is an integer value within in the range of code positions for the current method. A message box is displayed if you specify an integer outside this range.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon the operation.

The caret is then positioned at the specified position in the method source.

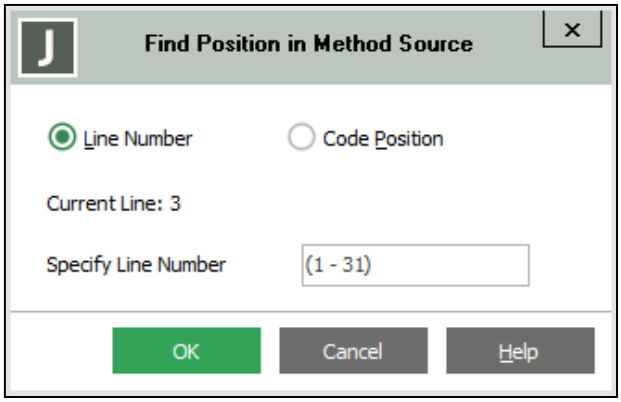
Going to a Specified Method Line

You can locate a specific line of a method in the editor pane.

To find a specific method line

1. In the Methods List of the Class Browser, Primitive Types Browser, or Sender Browser lists such as references and implementors, select the method whose line in the source code you want to locate.
2. Select the **Go To Line** command from the Methods menu, or press Ctrl+Alt+G. The Find Position in Method Source dialog is then displayed, with the **Line Number** option button selected by default.

   The current line at which the caret is positioned is displayed, and the range of line numbers is displayed in parentheses in the **Specify Line Number** text box as hint text, as shown in the following image.



3. In the **Specify Line Number** text box, specify the number of the line you want to locate. This position is an integer value within the range of lines for the current method.

   A message box is displayed if you specify an integer outside this range.

4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon the operation.

The caret is then positioned at the start of the specified line.

Applies to Version: 2020.0.01 and higher

Highlighting Methods in Lists of Methods

The Methods menu of the following forms provide the **Highlight Method Item Background** command, whose submenu commands enable you to use the five color commands to set the background color of the selected method list item or items; for example, as a reminder that more work on the selected method or methods is required. The color-related menu commands show the background color that will be used.

- Methods View Browser
- Method Status List Browser
- Unreferenced Methods Browser
- References Browsers
- Implementor Browsers
- Messages Browser
- Global Find/Replace Results Browser

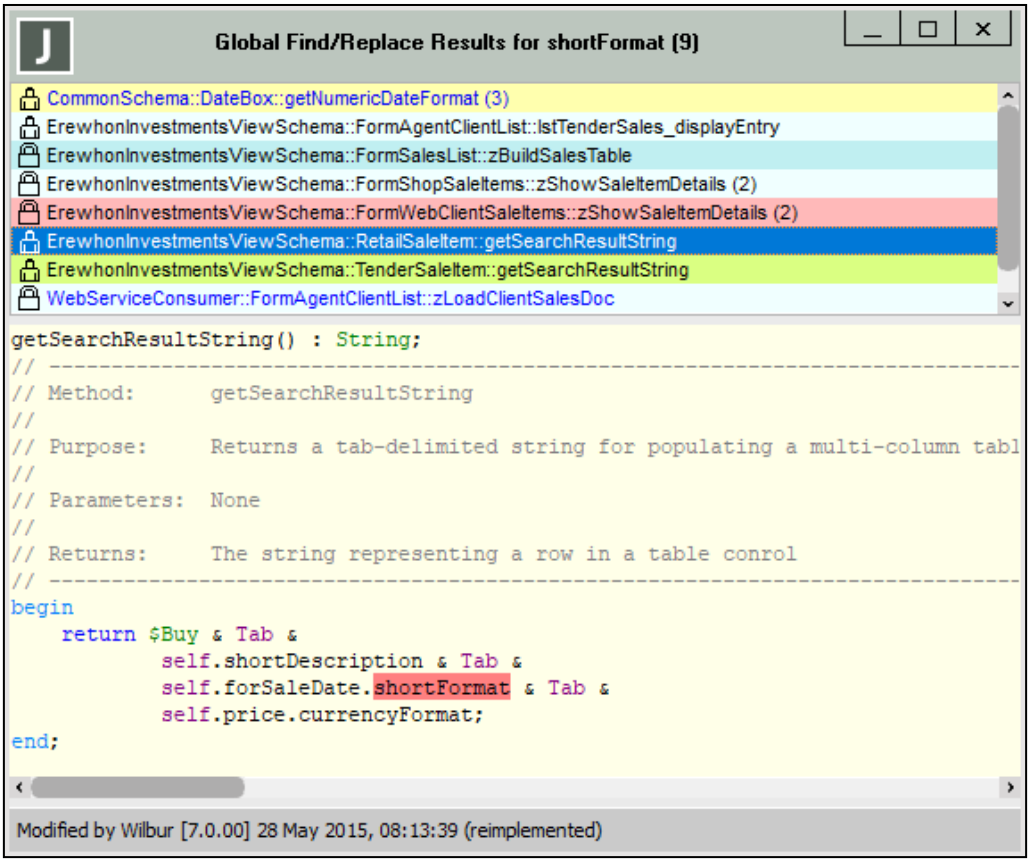
Clicking the **Highlight Method Item Background** command displays the following submenu commands.

- Clear Highlighting
- Blue
- Gray
- Green

- Red
- Yellow

Applies to Version: 2018.0.01 and higher

Clicking a color sets that color as the background color of the current selected method list item or items, as shown in the following image.



Select the **Clear Highlighting** command, to clear the highlighting color from the selected item or items. The command is disabled if no highlight has been assigned to the currently selected method list item or items.

Note This highlighting applies only while the current form is open; it is lost when the form is closed.

Displaying Method References, Implementations, and Messages

The Methods menu of the Class Browser, Primitive Types Browser, or Methods Browser enables you to display references to and implementations of the current method, and messages sent by the current method.

You can view:

- All references in the schema to the selected method
- All implementations in the schema of the selected method
- Interface methods to which the class method is mapped

- All local references in the current class to the selected method
- All local implementations in the current class of the selected method
- References to local implementors of a method
- Implementor references to the selected method
- Messages sent by the current method

In the References, Implementors, or Messages window, select the class or method whose references, implementation, or messages you want to view. The details of the selected method are then displayed in the editor pane. You can maintain the selected method.

For details about:

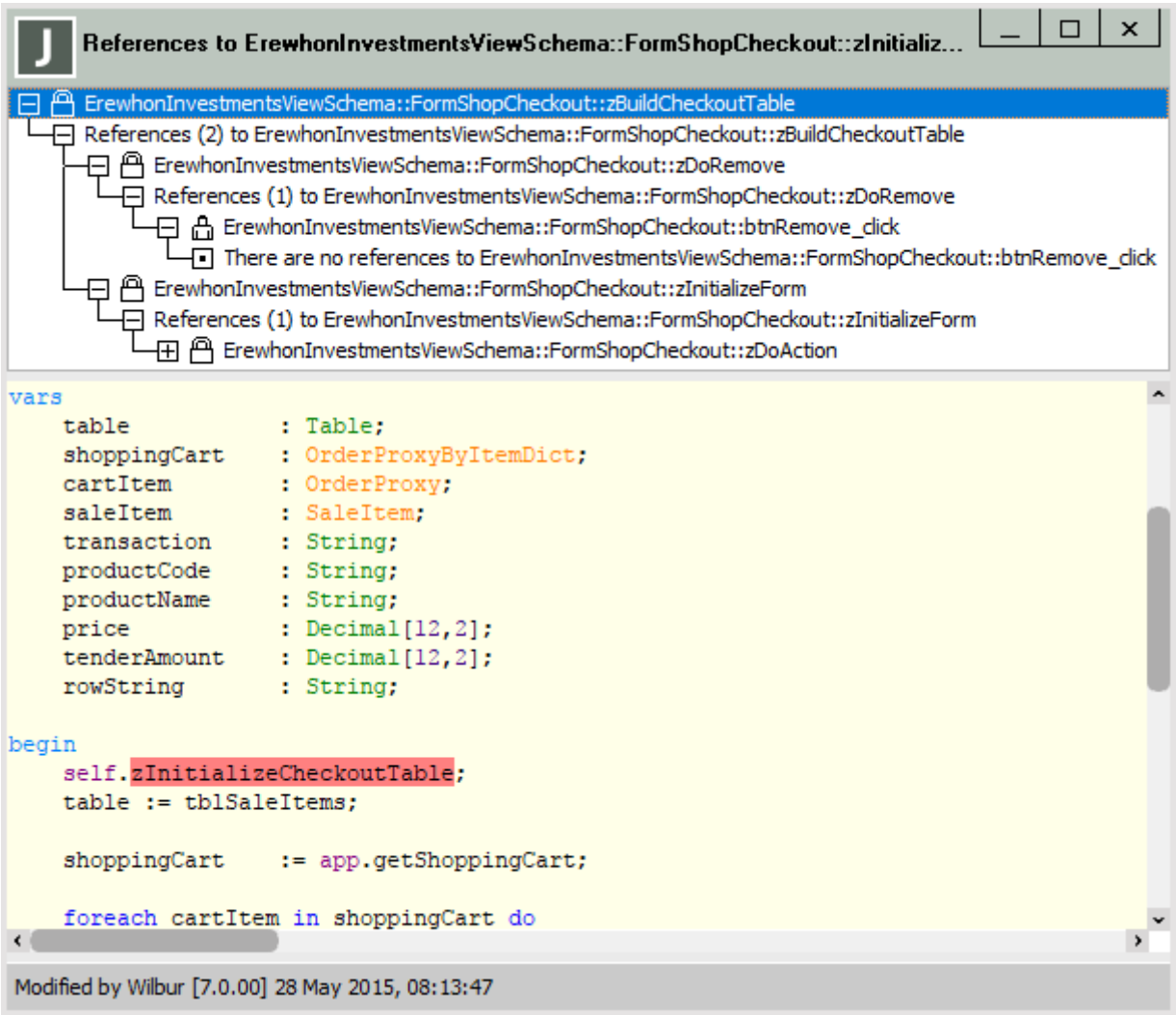
- Displaying references to interfaces, see "Displaying all References to the Selected Interface", in Chapter 14.
- Extracting a single method selected in the methods list at the top of a references window, see "Extracting a Method Selected in a Methods List", later in this chapter.
- Highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods".

Displaying all Method References in the Current Schema

To display all references in the schema to the selected method

- Select the **References** command from the Methods menu.

The Reference Browser is then displayed, listing all classes and their methods that reference the selected method, and enables you to view specific references to that method. Each method reference entry in turn has its references added to the list, and so on, recursively, as shown in the example in the following image.



Each method entry in the references list has an associated **+** (expand) or **-** (collapse) icon. Clicking on the **+** icon for a displayed method in the list box adds references to the method as child entries in the list box. Clicking on the **-** icon again hides the list of references to that method.

Each added method reference can have its references added to the display in the same way, up to a limit of 32 deep.

When the **+** icon is clicked, a child item is added to the list box: This entry is either:

- *There are no references to <method-name>*, when the method has no references.
- *References (<count>) to <method-name>*, when there are references. The reference entries to the method are added as a list box item children of this heading line.

The **Application** class **initialize** and **finalize** methods, property references (condition methods), and RPS mapping references have a leaf icon, and cannot be expanded.

Note The Methods menu **References** command for the displayed method displays the references to the selected method in a separate References Browser.

Displaying all Method Implementations in the Current Schema

To display all implementations in the schema of the selected method, perform one of the following actions

- Select the **Implementors** command from the Methods menu.
- Position the caret on or before the name of the method in the header in the editor pane and then press F11.

The Implementors window for the selected method is then displayed.

This window lists all classes in the current schema that implement the selected method and enables you to view specific implementations of that method.

Note The implementations are of methods with the name selected in the Methods List. Methods with the same name in different classes may have different logic.

Displaying Interface Methods to which the Selected Method is Mapped

To display all interface methods to which the selected method is mapped

1. In the Class Browser, select the method whose method mappings to one or more interface methods you want to view.
2. Select the **Interface Mappings** command from the Methods menu.

   This command is enabled only when the selected method is mapped to an interface method (that is, it is a stub method that was automatically generated in the current class when an interface method was defined).

The Method is mapped with the following interface methods browser window for the selected class method is then displayed. The title of the browser window states in parentheses the number of interface methods to which the class method maps. This window lists each interface method implemented by the selected class method, to enable you to view specific implementations of that method.

You can amend and compile a selected method, if required; for example, to automatically update the method signature when the signature of the interface method has changed. For details, see "Compiling an Interface Method" and "Implementing an Interface", in Chapter 14.

Displaying all Local Method References

To display all local references to the selected method

- Select the **Local References** command from the Methods menu.

The Local References window for the selected method is then displayed.

This window lists all subclasses and their methods in the selected class that reference the selected method and enables you to view specific references to that method.

Displaying all Local Method Implementations

To display all local implementations of the selected method

- Select the **Local Implementors** command from the Methods menu

The Local Implementors window for the selected method is then displayed.

This window lists all methods in the selected class and its subclasses that implement the selected method, and enables you to view specific implementations of that method. The list includes the schema name, regardless of the schema in which the selected method is defined.

Note The implementations are of methods with the name selected in the Methods List. Methods with the same name in different classes may have different logic.

Displaying References to Local Implementors of a Method

To display references to local implementors of the selected method

- Select the **Local Implementors References** command from the Methods menu

The Local Implementors References window for the selected method is then displayed.

This window lists all references to local implementors of a method, enabling you to assess the impact of a change, for example. These references are the equivalent of finding the local implementors of a method and then searching for local references to the located implementors.

Displaying Implementor References to a Method

To display implementor references to the selected method

- Select the **Implementor References** command from the Methods menu

The Implementors References window for the selected method is then displayed.

This window lists all implementor references of a method, enabling you to assess the impact of a change, for example. These references are the equivalent of finding the implementors of a method and then searching for references to the located implementors.

Applies to Version: 2018.0.02 (Service Pack 1) and higher

Displaying Messages Sent by the Current Method

To display all messages sent by the current method

- Select the **Messages** command from the Methods menu

The Messages window for the selected method is then displayed. This window lists all messages sent by the selected method and enables you to view references or implementors of these messages.

Using the Methods Menu of the Messages Window

The Methods menu, accessed from the Messages window, contains the commands listed in the following table.

Command	Action
References	Displays the references to the selected message
Implementors	Displays the implementors of the selected message
Compile	Compiles the current method

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods".

Displaying Message References

To display the references of the current message

1. Select the required message from the Messages List in the top left of the Messages window.
2. Select the **References** command from the Methods menu.

All references of the selected message are then displayed in the References List at the top right of the Messages window. You can also select a reference of the selected message from the Reference List in the top right of the Messages window.

The logic for the method containing the selected reference is then displayed in the editor pane. You can maintain this method.

Displaying Message Implementors

To display the implementors of the current message

1. Select the required message from the Messages List in the top left of the Messages window.
2. Select the **Implementors** command from the Methods menu.

All implementors of the selected message are then displayed in the Implementors List at the top right of the Messages window.

You can also select an implementor of the selected message from the Implementor List in the top right of the Messages window.

The logic for the method containing the selected implementor is then displayed in the editor pane. You can maintain this method.

Searching for Unused Schema Entities

The Schema menu in the Schema Browser enables you to locate unused classes, properties, class constants, and methods in one or more schemas. An entity is considered unused according to the following criteria.

- Classes
  - Has no subclasses
  - Is not a subclass of **Application**, **Global**, or **WebSession**
  - Is not used in a web service exposure

- Is not the membership of a **Collection** class
- Is not exported
- Has no method references, excluding its own methods
- Has no properties used as keys
- Is not used in an RPS mapping
- Has no property references in other classes
- Has no inverses

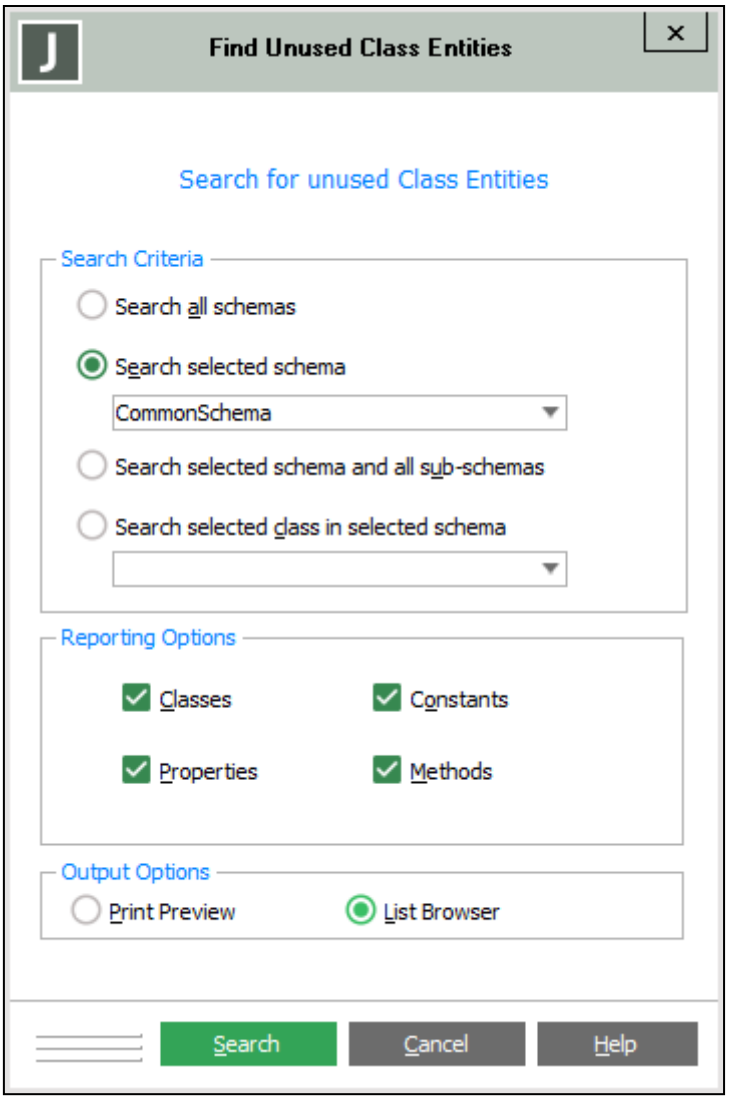
- Constants
  - Has no method references
  - Has no constant references
  - Is not exported
- Properties
  - Has no method references
  - Is not used as a key
  - Is not exported
  - Is not used in a web service
  - Is not a form control property
  - Is not used in a condition
  - Is not an HTML property
- Methods
  - Is not a condition
  - Does not implement an interface
  - Is not used in an RPS mapping
  - Is not an event method
  - Is not used in a web service
  - Is not exported
  - Is not a **JadeWebService** method
  - Is not a generated web services method
  - Is not checked out
  - Is not an interface method with implementors

- Is not a re-implemented RootSchema method
- Has no method references other than itself
- Is a re-implemented method but the super-methods have no method references
- Is not an application **initialize** or **finalize** method

To locate unused entities

1. Select the **Find Unused Class Entities** command from the Schema menu in the Schema Browser.

   The Find Unused Class Entities dialog, shown in the following image, is then displayed.



2. If you do not want to search for unused entities in the schema selected in the Schema Browser (the default value), perform one of the following actions in the Search Criteria group box.
   - Select the **Search all schemas** option button, to locate unused entities in all schemas in your Jade database.
   - In the combo box below the **Search selected schema** option button, select the schema in which you want to locate unused entities.
   - Select the **Search selected schema and all sub-schemas** option button and in the combo box above the option button, select the schema in which you want to locate unused entities in that schema and its subschemas.
   - Select the **Search selected class in selected schema** option button and in the combo box below the option button, select the class of the schema selected in the first combo box or that has a local copy class that will be scanned for unused entities.

     Tip For details about filtering the entries displayed in the list box associated with the combo box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.
3. In the Reporting Options group box:
   - Uncheck the **Classes** check box if you do not want to report on or display unused classes. This check box is checked by default; that is, unused classes are output to the report or displayed in a table in the Unused Class Entities form.
   - Uncheck the **Properties** check box if you do not want to report on or display unused properties. This check box is checked by default; that is, unused properties are output to the report or displayed in a table in the Unused Class Entities form.
   - Uncheck the **Constants** check box if you do not want to report on or display unused class constants. This check box is checked by default; that is, unused class constants are output to the report or displayed in a table in the Unused Class Entities form.
   - Uncheck the **Methods** check box if you do not want to report on or display unused methods. This check box is checked by default; that is, unused methods output to the report or displayed in a table in the Unused Class Entities form.
4. If you want to output the unused entities to a print-ready preview instead of displaying them in a table in the Unused Class Entities form (the default output action), select the **Print Preview** option button in the Output Options group box.
5. Click the **Search** button to initiate the search of unused entities that match your search criteria. Alternatively, click the **Cancel** button to abandon your selections.

Note If the selected schema or schemas are versioned, only the classes in the current version of the schema or schemas are searched for unused entities.

If you select the **Print Preview** output option, print-ready output is produced, containing the following columns.

- Schema
- Class
- Entity Type (the type of the entity that is unused)
- Class Entity Name (class constant, method, or property name; else blank when the unused entity is the class)

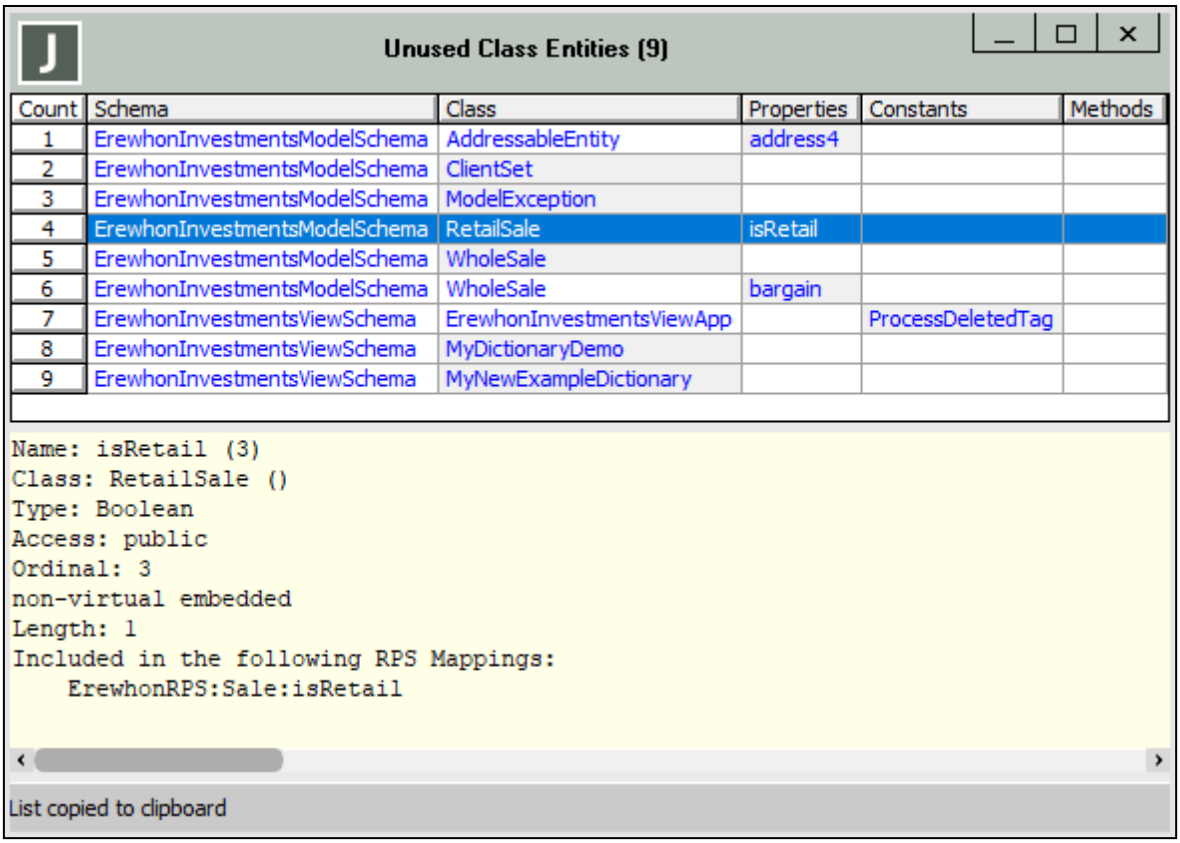
This list is ordered by schema name followed by class name and then unused constants, unused methods, and unused properties within each class.

For details about displaying unused entities output to the Unused Class Entities form, see "Displaying Unused Schema Entities", in the following section.

Applies to Version: 2016.0.01 and higher

Displaying Unused Schema Entities

When the default **List Browser** output option is selected in the Find Unused Class Entities dialog, the output is displayed on the Unused Class Entities form in a table, as shown in the following image. (For details about searching for unused entities, see "Locating Unused Schema Entities", in the previous section.)



The first column (**Count**) indicates the line number. The background color of a cell containing the name of the unused entity is drawn in light gray.

The table is ordered by schema name and class name and then unused constants, unused methods and properties within each class.

When you right-click on an unused entity in a cell in the table, the appropriate command for the Classes, Constants, Methods, or Properties menu for that unused entity is enabled. The commands that are available are:

- Classes
  - **References to *class-name***, which displays any references to the class
  - **Open Browser**, which opens a new Class Browser that displays and selects the class

- **Remove From List**, which removes the selected row from the table

- Properties
  - **References to *property-name***, which displays any references to the property
  - **Open Browser**, which opens a new Class Browser that displays and selects the class and the property
  - **Remove From List**, which removes the selected row from the table
- Constants
  - **References to *constant-name***, which displays any references to the constant
  - **Open Browser**, which opens a new Class Browser that displays and selects the class and the constant
  - **Remove**, which removes the selected constant from the class
  - **Remove From List**, which removes the selected row from the table
- Methods, which displays a standard Methods menu except for:
  - Some commands are disabled (for example, the **New Jade Method** command, and so on)
  - The **Open Browser** command, which opens a new Class Browser that displays and selects the class and the method
  - **Remove From List**, which removes the selected row from the table

Note The class and property **Remove** commands are not available from the popup menu Unused Class Entities form, as you must remove these from the Class Browser because they potentially cause versioning.

Applies to Version: 2016.0.01 and higher

Displaying Unreferenced Methods

The Browse menu enables you to select and display the methods that are not referenced by any other method in the current schema or in other schemas. You can specify the display of unreferenced methods by performing one of the following actions.

- Type of methods (that is, external methods, Jade methods, or all methods)
- Methods provided by the selected class and optionally its subclasses, superclasses, or for all classes
- Methods provided by the selected schema and optionally its subschema, superschemas, or for all schemas

A method is considered unreferenced according to the following criteria.

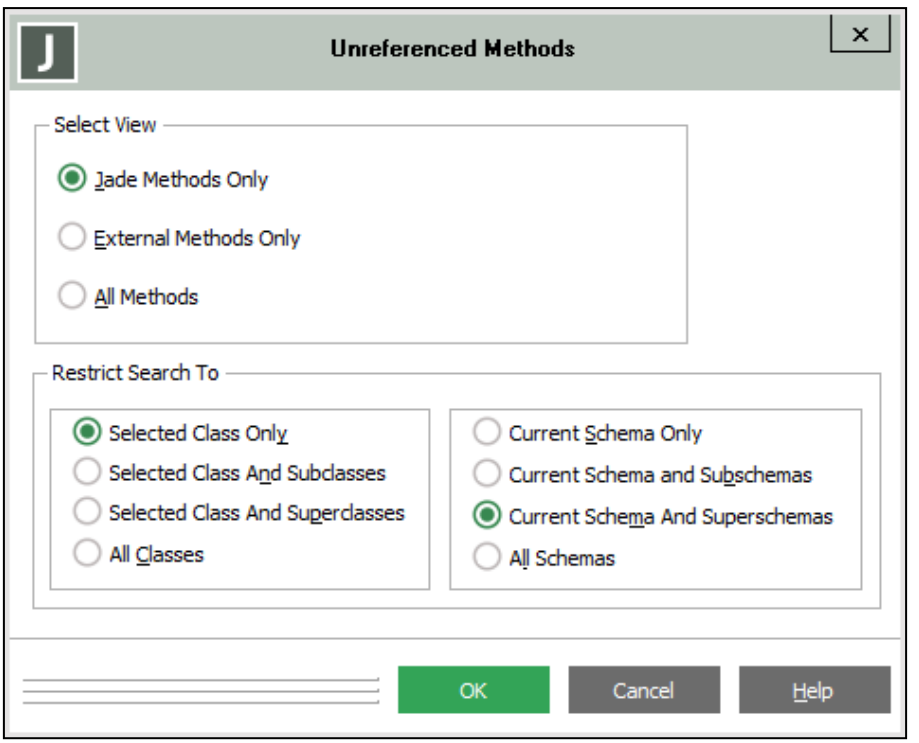
- Not a condition
- Does not implement an interface
- Not used in an RPS mapping
- Not an event method
- Not used in a web service

- Not exported
- Not a **JadeWebServices** class or subclass method.
- Not a generated web services method
- Not checked out
- Not an interface method with implementors
- Not a reimplemented RootSchema method
- Has no method references other than itself
- Is a reimplemented method but the superclass methods have no method references
- Not an application **initialize** or **finalize** method
- Not a **JadeScript** method
- Not a **create**, **delete**, **userNotification**, **sysNotification**, or **timerEvent** method

To specify the unreferenced methods that you want to view

1. From the browser, select the class or superclass whose unreferenced methods you want to view if you do not want to view the unreferenced methods of all classes.
2. Select the **Unreferenced Methods** command from the Browse menu.

   The Unreferenced Methods dialog, shown in the following image, is then displayed, to enable you to specify the unreferenced methods that you want to view.



3. In the Select View group box, select the type of unreferenced methods that you want to view. By default, only unreferenced Jade methods are displayed.

4. Use the options in the Restrict Search To group box to select the classes and schemas to which you want to restrict the search for unreferenced methods if you do not want to display only the unreferenced methods of the current class in the current schema and its superschemas.

   Note The search is context-dependent. For example, if the selected schema has no subschemas, the **Current Schema And Subschemas** option button is disabled.

   When the Class Browser or Primitive Types Browser has focus, the search is performed only on the selected class in the current schema and its superschemas by default (that is, the **Selected Class Only** option button and the **Current Schema And Superschemas** option button are selected).

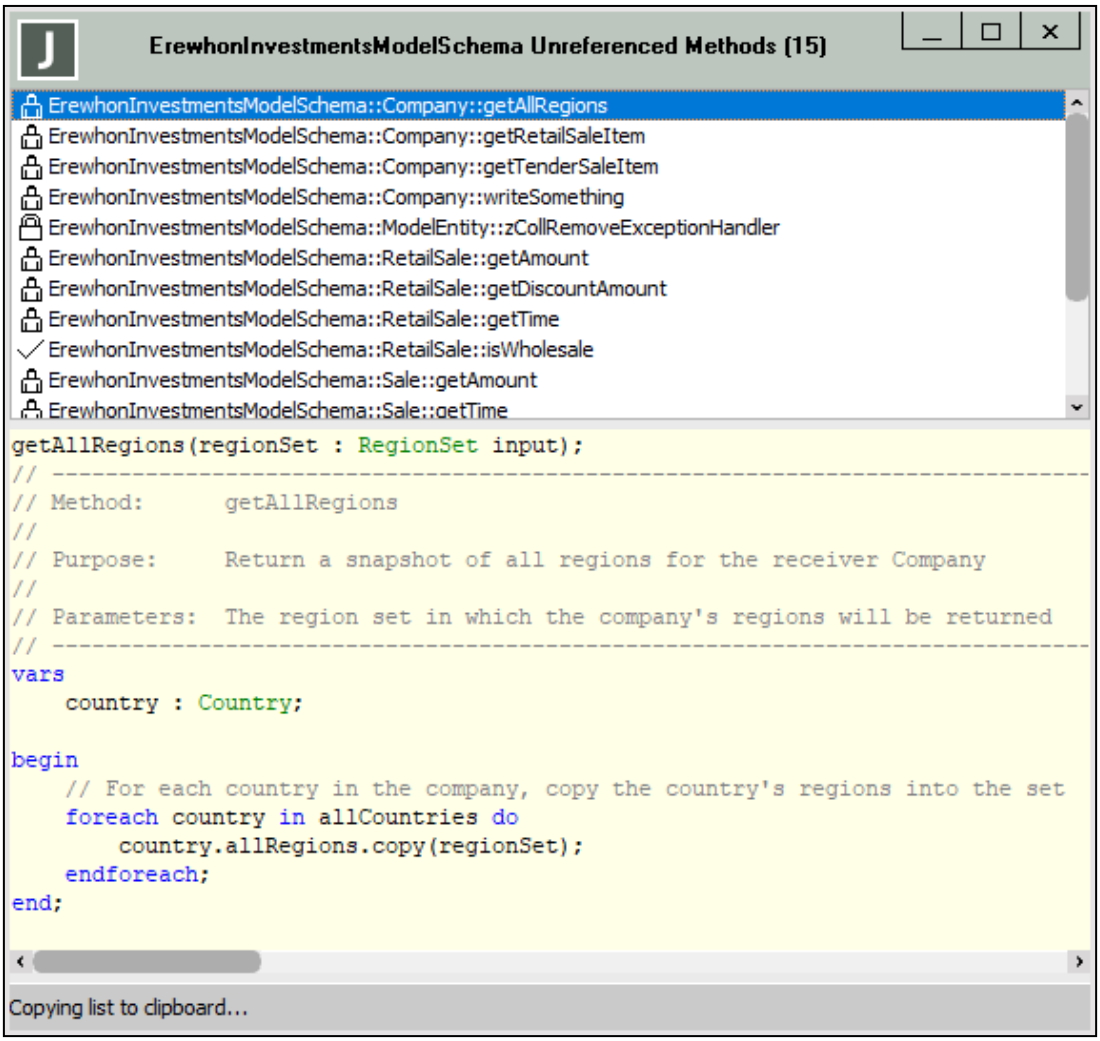
   Conversely, when the Schema Browser has focus, options that are not valid are disabled and the **All Classes** and **Current Schema And Superschemas** option buttons are selected by default.

   If you want to restrict the search for unreferenced methods to another option, the values that are you can select are listed in the following table, to enable you to select the appropriate option.

Class Restriction Options	Schema Restriction Options
Selected Class Only (Class or Primitive Types Browser default value)	Current Schema Only
Selected Class And Subclasses	Current Schema and Subschemas
Selected Class And Superclasses	Current Schema and Superschemas (default value)
All Classes (Schema Browser default value)	All Schemas

5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

When the search is complete, the Unreferenced Methods Browser, shown in the following image, is then displayed.



Use the Unreferenced Methods Browser to view the unreferenced methods that match your specified search options. The method list displays all selected unreferenced methods in alphabetical order by class. User-defined methods are displayed in black and methods defined in superschemas are displayed in blue.

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods".

You can extract a single method selected in the methods list at the top of the Unreferenced Methods Browser. For details, see "Extracting a Method Selected in a Methods List", later in this chapter.

To view the logic for a method

- Select the required method in the method list.

The logic for the selected method is then displayed in the editor pane. You can modify and compile user-defined methods but you cannot modify system methods.

Displaying Unused Local Variables and Parameters

The Methods menu of the Class Browser or Primitive Types Browser enables you to audit the local variables in the current method, display those that are unused, and remove one unused local variable or all unused local variables in the method.

» To audit and maintain local variables in the method displayed in the editor pane

1. Select the **Unused Local Variables** command from the Methods menu. (This command is disabled if the method is uncompiled or in error.)

   If there are no unused variables, a message box displays *There are no unused local variables*.

   If there is one or more unused local variable in the method, the first unused local variable is highlighted in the editor pane and the Find Unused Variables dialog displays the name of the first unused local variable.

2. To display the next unused variable, click the **Find Next** button. If there are no more in the current method, the dialog is closed and a message box displays *There are no more unused local variables*.

3. To remove the displayed unused variable from the method source, click the **Remove** button.

   The logic then continues as though you had clicked the **Find Next** button.

4. To remove all unused variables from the method source, click the **Remove All** button.

   The dialog is then closed and a message box lists the names of all variables that have been removed from the method.

5. To close the Find Unused Variables dialog, click the **Cancel** button. (No previous remove actions are undone.)

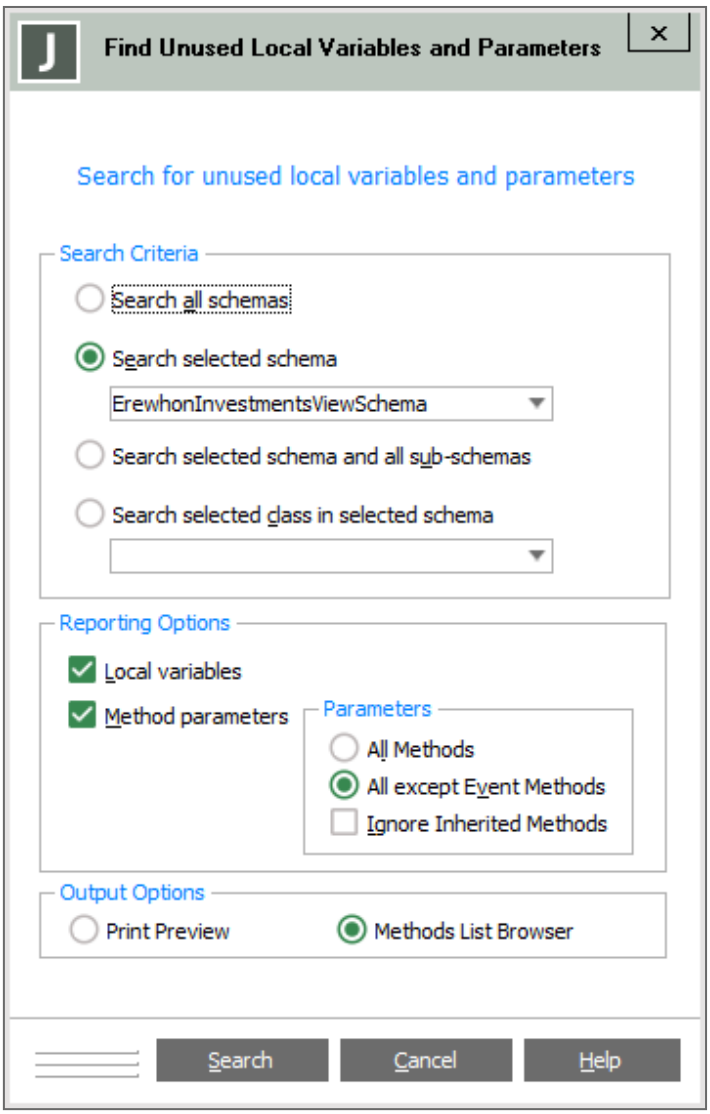
Note Any changes made to the method can be undone and the method will be unsaved. (Pressing F8 saves and recompiles a changed method.)

The Schema menu in the Schema Browser enables you to audit the local variables that are unused in all methods in a schema or class.

To audit local variables and parameters in all methods or a class of the schema selected in the Schema Browser

1. Select the **Find Unused Local Variables/Parameters** command from the Schema menu.

   The Find Unused Local Variables and Parameters dialog, shown in the following image, is then displayed.



2. If you do not want to search for unused local variables and parameters in all methods in all schemas (the default value), perform one of the following actions in the Search Criteria group box.

   - Select the **Search selected schema** option button and in the combo box below the option button, select the schema in which you want to locate unused local variables and parameters in all methods.
   - Select the **Search selected schema and all sub-schemas** option button and in the combo box above the option button, select the schema in which you want to locate unused local variables and parameters in all methods in that schema and its subschemas.

- Select the **Search selected class in selected schema** option button and in the combo box below the option button, select the class of the schema selected in the first combo box or that has a local copy class whose methods will be scanned for unused local variables and parameters.

  Tip For details about filtering the entries displayed in the list box associated with the combo box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.

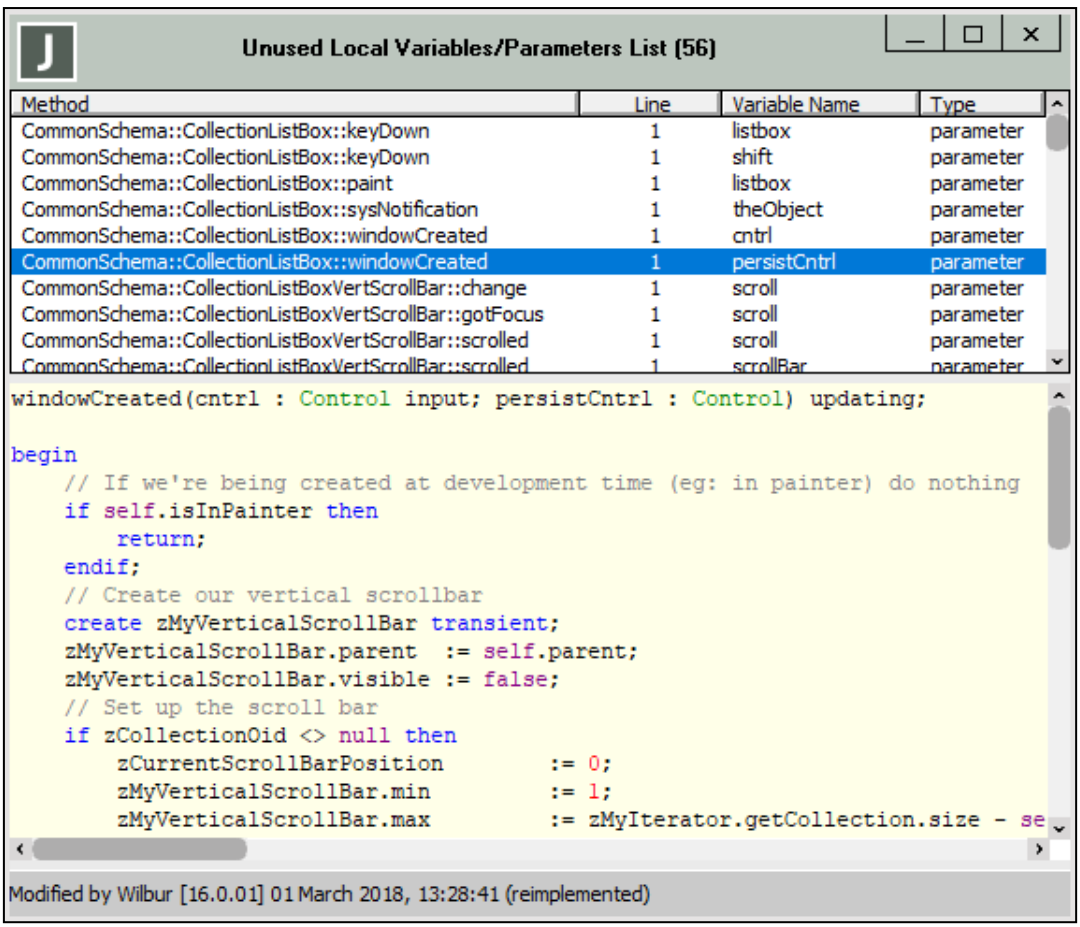
3. In the Reporting Options group box, uncheck the:

   - **Local Variables** check box if you do not want to display unused local variables. This check box is checked by default; that is, unused local variables are output to the report.
   - **Method Parameters** check box if you do not want to display unused method parameters. This check box is checked by default; that is, unused method parameters are output to the report.

     If you want to display unused method parameters in *all* methods in the schema instead of all methods *except* for event methods (the default), select the **All Methods** option button. The Parameters group box is visible only when the **Method Parameters** check box is checked.

4. In the Output Options group box, select the **Print Preview** option button if you want to preview the unused local variables and parameters report output.

   By default, the **Methods List Browser** option button is selected, indicating that the unused local variables and parameters are displayed in a Methods Browser, shown in the following image.



To display the method source with the unused item visible in the editor pane below the list on the form, click on a method in the list. You can edit this method source.

5. In the Parameters group box of the reporting options, check the **Ignore Inherited Methods** check box if you want to exclude the parameters of any method that has reimplementations from the parameters unused list, regardless of whether the parameters are used. This covers the case where the superclass implementation of a method are essentially blank default implementations that are then properly defined in the subclasses.

   If you include the tag **[ExcludeFromUnusedParameterReport]** (which must be enclosed within **[ ]** bracket symbols) on the same line as an unused parameter, that parameter is not included in the report. For example, the first of the following code fragment comments does *not* exclude the unused parameter from the report, but the second example does.

   ```
   run(indx: Integer, str: String) updating; //
   [ExcludeFromUnusedParameterReport]

   run(indx: Integer, // [ExcludeFromUnusedParameterReport]
       str: String) updating;
   ```

   Tip The **UnusedParameterReport** global constant category contains the **"ExcludeFromUnusedParameterReport"** global constant, which you can use to locate the string tag in comments by using the local or global search functionality.

6. Click the **Search** button to initiate the search. Alternatively, click the **Cancel** button to abandon your selections.

When a scan is performed using the **Find Unused Local Variables/Parameters** command from the Schema menu and the located unused variables and parameters are displayed in the Unused Local Variables/Parameters form, double-clicking a method entry in the list for an unused local variable entry results in *all* listed unused local variables being removed from the method source and the method is then recompiled. All of the unused local variable entries in the list for that method are removed from the list except for the last entry, so that you can view the method changes. You can also undo the changes, if required. (Multiple undo actions may be required.)

Notes If there is no remaining text between the **vars** and **begin** statements, the **vars** section is removed. No comments are removed.

You can also remove unused local variables by selecting the **Remove Unused Local Variables** command in the Methods menu.

If you click the only remaining entry for a method again and there are no changes required, the entry is removed from the list.

The result of print preview search output to a print-ready report contains the:

- Schema of the method
- Class in which the method is defined
- Method name where the create was performed
- Line number in the method of the unused variable statement or parameter
- Variable name of the unused local variable or parameter
- Type of value; that is, **variable** or **parameter**

Removing Unused Local Variables

In addition to double-clicking a method entry in the list for an unused local variable entry in the Unused Local Variables/Parameters form, documented under "Displaying Unused Local Variables and Parameters" in the previous topic, you can also remove unused local variables from the selected method entry in the Unused Local Variables/Parameters form list by selecting the **Remove Unused Local Variables** command in the Methods menu.

All unused variables for that method are then removed from the method. which is displayed in the editor pane below the list. The method is then recompiled unless you have already modified the method and not yet saved your changes.

Note If there is no remaining text between the **vars** and **begin** statements, the **vars** section is removed. No comments are removed.

All of the unused local variable entries in the list for that method are removed from the list except for the last entry, so that you can view the method changes. You can also undo the changes, if required. (Multiple undo actions may be required.)

Locating Possible Transient Object Leaks

The Schema menu in the Schema Browser enables you to scan all methods in a schema or class for possible transient leaks.

The transient leak analysis can report potential leaks that are not in fact leaks, for reasons that are outside of the understanding of the analysis. You can mark such entries for exclusion from the report that is produced, by placing the following tag (which must be enclosed within **[ ]** bracket symbols) in a comment on the same line as the method call.

```
[ExcludeFromTransientLeakReport]
```

The following code fragment examples mark lines of code for exclusion from the transient leak analysis.

```
create cust transient; // create the customer [ExcludeFromTransientLeakReport]

cust := methodReturnsTransient(); // [ExcludeFromTransientLeakReport]
```

Note that for multiple-line function calls, the **// [ExcludeFromTransientLeakReport]** comment must be placed on the same line as the method call, instead of that of the method parameters. For example, the first of the following code fragment comments does *not* exclude the potential leak, but the second example does.

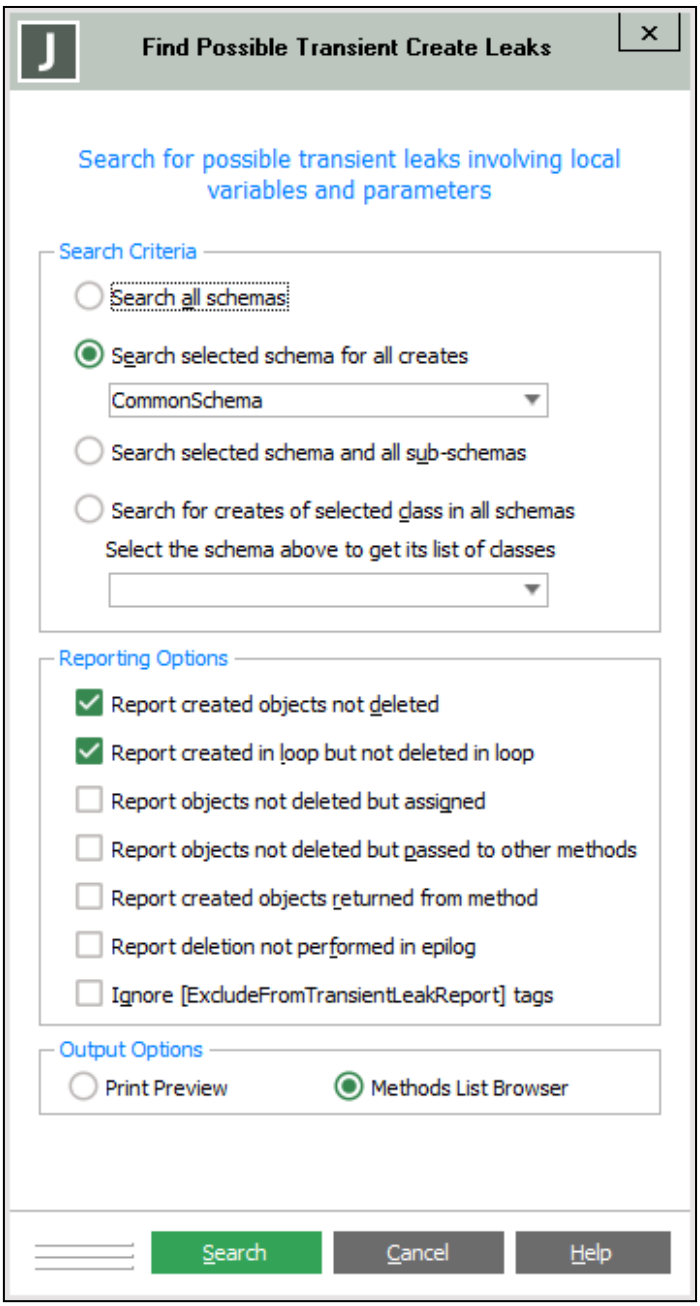
```
var := methodReturnsTransient(parameter1,
                              parameter2); // [ExcludeFromTransientLeakReport]

var := methodReturnsTransient(parameter1, // [ExcludeFromTransientLeakReport]
                              parameter2);
```

In addition, Jade provides a **PossibleTransientLeaks** global constant category that contains the **"ExcludeFromTransientLeakReport"** global constant.

To locate possible transient object leaks in the schema selected in the Schema Browser

1. Select the **Find Possible Transient Leaks** command from the Schema menu.

   The Find Possible Transient Create Leaks dialog, shown in the following image, is then displayed.



   The name of the schema selected in the Schema Browser is displayed in the first combo box in the Search Criteria group box.

2. If you do not want to search for possible transient leaks involving local variables and parameters in all methods in the schema selected in the Schema Browser (the default value), perform one of the following actions in the Search Criteria group box.

   - Select another schema in the combo box below the **Search selected schema for all creates** option button if you want to locate possible transient object leaks in a schema other than the one currently selected in the Schema Browser.
   - Select the **Search all schemas** option button if you want to locate possible transient leaks in all methods in all schemas.
   - Select the **Search selected schema and all sub-schemas** option button and in the combo box above the option button, select the schema in which you want to locate possible transient leaks in all methods in that schema and its subschemas.
   - Select the **Search for creates of selected class in all schemas** option button and in the **Select the schema above to get its list of classes** combo box below the option button, select the class of the schema selected in the combo box beneath the **Search selected schema and all sub-schemas** option button that has a local copy class whose methods will be scanned for possible transient leaks.

     All creates of that class in *all* schemas are searched; that is, the search is performed in the schema in which the class is defined and all subschemas. If the class is a **RootSchema** class, all schemas are searched.

     Tip For details about filtering the entries displayed in the list box associated with the combo box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.

3. Uncheck the **Report created objects not deleted** check box if you do not want to locate any transient **create** statement leaks on local variables or method parameters where there is no delete of that variable and that variable was not returned by the method and was not passed as a parameter to another method.

   By default, this check box is checked, indicating that created objects are checked for deletion.

4. Uncheck the **Report created in loop but not deleted in loop** check box if you do not want to locate possible leaks where a create is performed inside a loop but the delete is outside of the loop. This may not actually be a leak, because only one create may have been performed.

   By default, this check box is checked, indicating that objects created in a loop are checked for deletion within that loop.

5. Check the **Report objects not deleted but assigned** check box if you want to locate creates performed where there is no delete, but the variable was assigned to another property or variable. This may not represent a leak, as it is not known whether this value is subsequently deleted.

   By default, this check box is unchecked, indicating that assigned objects are not checked for deletion.

6. Check the **Report objects not deleted but passed to other methods** check box if you want to locate creates performed where there is no delete, but the variable was passed as a parameter to another method. This may not represent a leak, as it is not known what subsequently happens to that object.

   By default, this check box is unchecked, indicating that objects passed to other methods are not checked for deletion.

7. Check the **Report created objects returned from method** check box if you want to locate creates performed where the object is returned from the method. This may not represent a leak, as it is not known what the calling method will do with that object.

   By default, this check box is unchecked, indicating that objects returned from methods are not scanned for possible leaks.

8. Check the **Report deletion not performed in epilog** check box if you want to locate objects that have been created and deleted but where the object is not deleted in the epilog section of the method. This situation would result in a leak if an exception occurred during the method call.

   By default, this check box is unchecked, indicating that epilogs are not scanned for possible leaks.

9. Check the **Ignore [ExcludeFromTransientLeakReport] tags** check box if you want method lines containing this tag in a comment to be analyzed for possible transient leaks instead of being ignored.

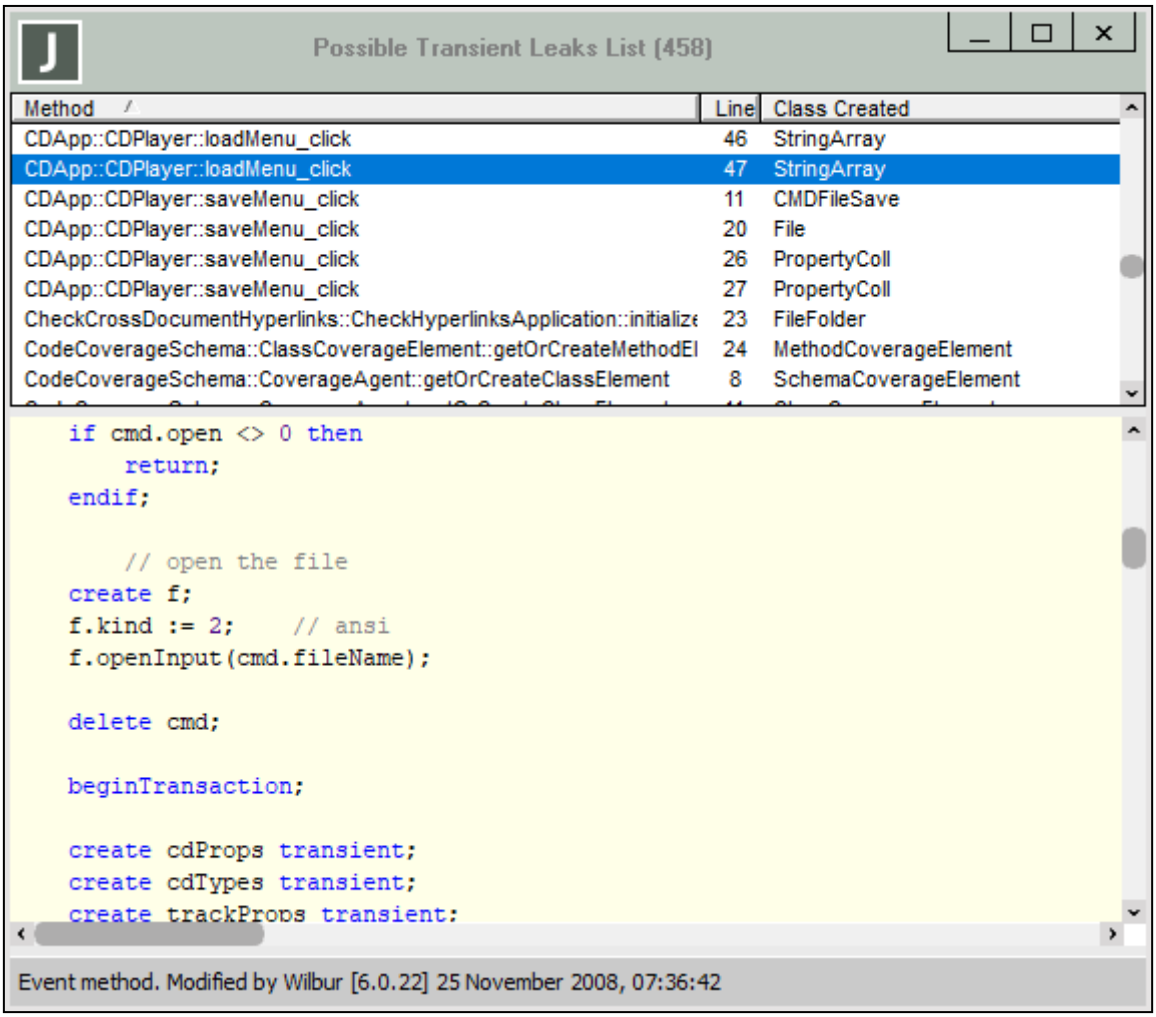
   By default, this check box is unchecked, indicating that method lines that contain the **[ExcludeFromTransientLeakReport]** tag are ignored during the possible transient leak analysis.

10. In the Output Options group box, select the **Print Preview** option button if you want to preview the possible transient leaks report output.

    By default, the **Methods List Browser** option button is selected, indicating that the methods in which there are possible transient leaks are displayed in a Methods Browser, shown in the following image, so that you can edit the method source whose transient leak list entry you have selected.

11. Click the **Search** button to initiate the search. Alternatively, click the **Cancel** button to abandon your selections.

The following image is an example of the possible transient leaks displayed in a Methods Browser.



When you press Tab to move focus from the table of methods at the top of the browser to the editor pane, the cursor is positioned at the left of the source line in which the possible transient leak has been identified.

From Jade version 2020, the possible transient leaks listed in the upper pane of the Methods Browser are now sorted in ascending order of the **Method** column (that is, by schema, class, and method). Left-click on the first row (the column header) of the table to sort the possible transient leaks in descending order (that is, **Z** through **A** or **9** through **1**) using the contents of the clicked column. Click the same column again to toggle the order of the entries. The column that is used for sorting displays an arrow icon to indicate whether the sorting is in ascending (^) or descending (v) order.

The result of your search is output to a report that is displayed in a Methods Browser or in print preview mode, depending on the option that you selected in step 10 of the previous instruction. This report contains the:

- Schema of the method
- Class in which the method is defined
- Method name where the create was performed
- Line number of the **create** statement in the method

- Type of the object being created
- Variable name used in the **create** statement
- Description of the possible leak

Note The report will also track calls to RootSchema methods that create and return a transient object that needs to be deleted (for example, **Class::allProperties**) and methods scanned that return a transient object that was created. If a possible leak is detected for such situations, the description will include the method name involved and the line number will be the statement in which the call was made.

Using a Method in a Workspace or the JadeScript Class

The Workspace or the **JadeScript** system class enables you to test a method. For example, you can:

- Use the **write** instruction to output results to the Jade Interpreter Output Viewer window
- Test that the common Open dialog for file selection is displayed
- Subclass the **JadeScript** class in your user schemas
- Run and debug a Jade script from any subclass of the **JadeScript** class

  Running a Jade script in a subclass of **JadeScript** using the **jadclient** non-GUI application requires the class to be specified in the **executeClass** command line parameter.
- Debug a Workspace method

The **JadeScript** class also enables you to execute and debug methods independently of your Jade application. (Running a **JadeScript** method always starts a new application copy.)

Notes You can execute a **JadeScript** method or Workspace code only from the current schema context (version)

The Refactor menu is disabled for a **Workspace** method because a **Workspace** has no **self** class object and therefore the menu is not relevant.

For details, see the following subsections.

Using a Workspace

The Workspace window consists of an editor pane, to enable you to write and execute a unit of code that is not to form part of your application. For example, you can use the Workspace window to test code (using the **write** or **read** Jade instructions) or to create, initialize, or delete objects.

If your method contains the Jade **write** instruction, the Jade Interpreter Output Viewer window is then displayed, showing the output from your executed method. (For details, see "Compiling and Executing Code in a Workspace", later in this chapter, and "Using the Jade Interpreter Output Viewer", in Chapter 1 of the *Runtime Application Guide*.)

A Workspace is available to all other user-defined schemas.

For details about debugging a Workspace, see "Debugging a Workspace Method".

Notes A Workspace method must not have a method signature.

Although the **Open Workspace** toolbar button and the File menu **Open** command are enabled from the **RootSchema**, the only Workspace that you can open in the **RootSchema** is an encrypted diagnostic Workspace supplied by Jade Support.

From version 2018.0.01, you can drag and drop a file in the following cases.

- Dropping one or more files onto the **New Workspace** or **Open Workspace** toolbar button opens a new workspace for each file that is dropped.

  Any folders that are dropped are ignored.

- Dropping a file onto the Workspace window displays the contents of the file in the workspace, replacing any previous content. If the previous workspace was unsaved, you are first prompted to save or discard the current workspace, or to cancel the display of the dropped file.

  Dropping multiple files is rejected.

To access a new Workspace window, perform one of the following actions

- Select the **New** command from the File menu.
- Click the **New Workspace** toolbar button.

To open an existing Workspace window

- Select the **Open** command from the File menu.
- Click the **Open Workspace** toolbar button.

Compiling and Executing Code in a Workspace

To compile and execute your Workspace code

1. Perform one of the following actions.
   - Select the **Select All** command from the Edit menu.
   - Press Ctrl+A.
2. Press F9.

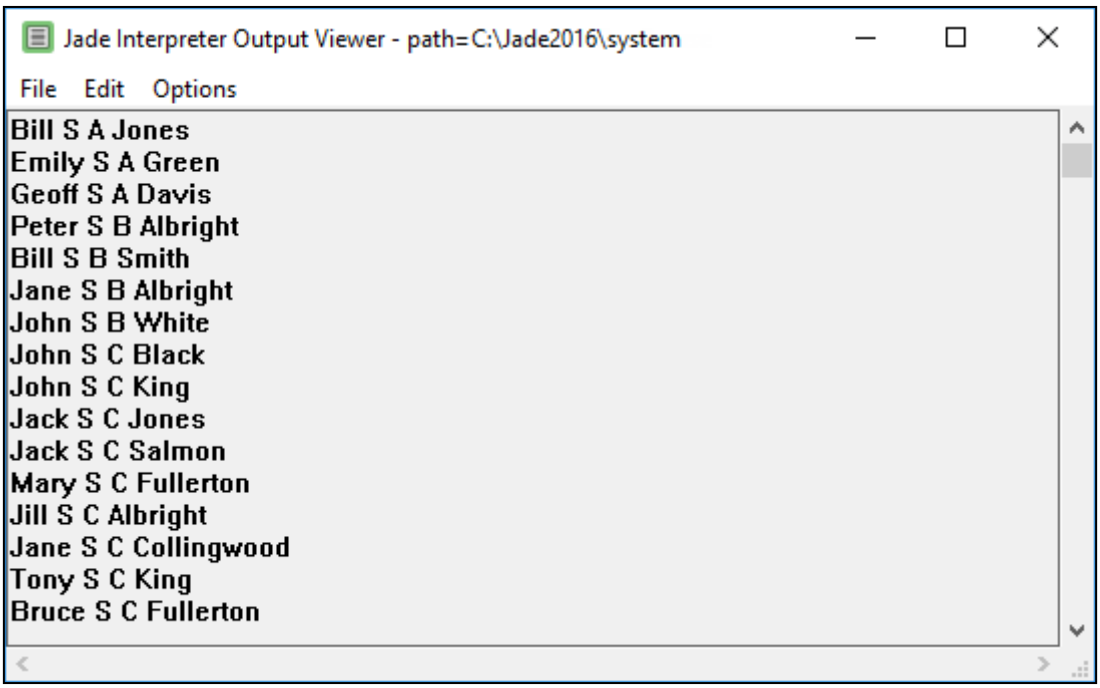
Notes If you have not selected any code in the Workspace, all code in the Workspace is compiled and executed.

You can execute Workspace text only in the current version of a versioned schema.

The code is then compiled and executed. If the compiler detects a syntax error, the error code and message are displayed in the status line. Amend your code logic, as required, before compiling and executing your code again.

When no errors are detected, the message *Execution complete* is displayed in the status line.

If your code contains the Jade **write** instruction, the Jade Interpreter Output Viewer window is then displayed, displaying the output from your executed code, as shown in the following image.



For details about:

- Viewing the output from your code, see "Using the Jade Interpreter Output Viewer", in Chapter 1 of the *Runtime Application Guide*.
- Disabling the output of **write** instructions to the Jade Interpreter Output Viewer, see "Jade Interpreter Section [JadeInterpreter]", in the *Jade Initialization File Reference*.

Saving Your Workspace

You can save your Workspace for later use.

» **To save your Workspace, perform one of the following actions**

- Select the **Save** command from the File menu.
- Click the **Save** toolbar button.
- Press F2.

Alternatively, select the **Save As** command from the File menu. The common File dialog is then displayed, enabling you to save your workspace code to a file.

Using the User Input Dialog

If your code contains the Jade **read** instruction, the User Input dialog is then displayed, prompting for the value of the primitive type variable that is required.

The local variable primitive type is displayed in the text box caption and the value of the variable is displayed in the text box. If the variable is of primitive type **Any**, the input is handled as a **String**.

To specify a variable value

1. In the text box, specify the value for the variable that is being read. For example, if your test code contains the following logic, specify the number that is being tested, as follows.

```
read numberToTest;
foreach count in 1 to numberToTest do
    ...
endforeach;
```

2. Click the **OK** button. Alternatively, click the **Cancel** button to assign a **null** value to the variable.

The action that follows the **read** instruction in your code is then performed.

Note The **write** and **read** Jade instructions are useful in a Workspace. A completed Jade application should use more-sophisticated user interface techniques for input and output of data.

Debugging a Workspace Method

To debug a Jade Workspace

1. In the Workspace window, select the text to be executed.

   If you do not select some text, the entire contents of the Workspace window are debugged.

   Notes You cannot set breakpoints in the original Workspace window.

   You can execute Workspace text only in the current version of a versioned schema.

2. Select the **Debug** command from the Jade menu. Alternatively, press F9.

   The **Debug** command is enabled only when a some or all of a Workspace window or a method in the **JadeScript** class is currently selected, to enable you to observe the behavior of your code at run time.

The selected text is then compiled as a method and executed in debug mode.

The debug execution initially stops on the first line of execution. You can use the Jade Debugger to set debug breakpoints in the method. You can then use the debugger to step through the logic or debug and run it to meet your requirements.

For details, see Chapter 7, "Using the Jade Debugger".

Using a JadeScript Method

To execute the current JadeScript method, perform one of the following actions

- Select the **Execute It** command from the Jade menu
- Press F9

A new application copy is started and the method is then compiled and executed. If a syntax error is detected, the error code and message are displayed in the status line. Amend your method logic, as required, before compiling and executing your method again.

Notes Only the first error in the logic is reported. When that error has been corrected and the method recompiled, any following error is reported individually or the compilation and execution is completed.

You can execute a **JadeScript** method only in the current version of a versioned schema.

When no errors are detected, the message *Compilation complete* is briefly displayed in the status line, followed by the message *Execution complete*.

If your method contains the Jade **write** instruction, the Jade Interpreter Output Viewer window is then displayed, showing the output from your executed method. (For details, see "Compiling and Executing Code in a Workspace" under "Using a Workspace", earlier in this chapter, and "Using the Jade Interpreter Output Viewer", in Chapter 1 of the *Runtime Application Guide*.)

Executing and Debugging a JadeScript Method

To execute and debug the current JadeScript method, perform one of the following actions

- Select the **Debug** command from the Jade menu in the Class Browser
- Press Shift+F9

A new application copy is started, the current method is compiled and executed, and the Debugger window is then displayed. For details, see "Starting an Application in Debug Mode", in Chapter 7.

The **Debug** command is enabled only when a some or all of a Workspace window or a method in the **JadeScript** class is currently selected, to enable you to observe the behavior of your code at run time. In addition, you can execute and debug a **JadeScript** method only in the current version of a versioned schema.

Running a Unit Test

Unit testing enables you to improve the quality and reliability of your applications by writing unit tests as your system is built and by performing extensive unit testing with minimal user intervention. You can build your own unit testing driver or you can use the default Jade unit test functionality.

Develop tests for a unit of code (which can be a fragment, a method, or a module) as the code is written, to ensure that quality is built into a project from the early stages.

Unit tests isolate each part of the program and show that the individual parts are correct. They provide a strict, written contract that the piece of code must satisfy. Use unit testing to:

- Enable refactoring without regression (that is, ensuring the module still works correctly)
- Eliminate uncertainty in the units
- Enable a bottom-up testing style approach
- Document the functionality provided by a unit and how to use it

The Jade menu displays the **Unit Test Debug** command if a user-defined schema with the **JadeTestCase** subclass, a **JadeTestCase** class or subclass, or a method in a **JadeTestCase** subclass is selected.

Note Visual Studio uses the F9 key to set breakpoints, by default, and uses the F5 key to run and continue debug execution. Jade uses the F5 key to set breakpoints and the F9 key to run and continue debug execution. Switching between the two development environments can lead to confusion and frustration when using the F5 and F9 keys. You can swap the F9 and F5 accelerator key bindings, by checking the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) accelerators** check box on the **Short Cut Keys** sheet on the Preferences dialog. This check box is unchecked by default (that is, false). For details, see "Maintaining Shortcut Keys", in Chapter 2.

To run a unit test and invoke the test dialog

- To invoke the test dialog, press F9 or select the **Unit Test** command from the Jade menu on one of the following.
  - A unit test method in a **JadeTestCase** subclass, to run the unit test for that method only.
  - A unit test class (the **JadeTestCase** class or a subclass of the **JadeTestCase** class), to run all unit test methods in that class and its subclasses.
  - A user-defined schema that has **JadeTestCase** subclasses, to run all unit test methods in all subclasses of the **JadeTestCase** class in the hierarchy of the schema; that is, it runs all unit test methods of all subclasses of **JadeTestCase** in the schema and all of its subschemas.
  - The **JadeTestCase** class in the **RootSchema**, to run all unit test methods in all subclasses of the class in all user-defined schemas.

To execute and debug a selected unit test, perform one of the following actions

- Select the **Unit Test Debug** command from the Jade menu.
- Press Shift+F9.

The Jade unit test framework is then initiated in Jade debug mode for the selected **JadeTestCase** class or selected method of the class if a method is selected. For details, see Chapter 17, "Running Unit Tests", in the *Developer's Reference*.

Viewing Code Coverage Results to Analyze Methods in Your Applications

The Code Coverage Results Browser enables you to view the contents of existing code coverage results output files. For details about generating a code coverage results file, see "Reporting Code Coverage Results", in Chapter 1 of the *Runtime Application Guide*.

You can also use the **View** command in the Jade User Interrupt Code Coverage submenu to view code coverage results in a deployed application. For details, see "Viewing Code Coverage Results", in Chapter 1 of the *Runtime Application Guide*.

Loading the results file into the Code Coverage Results Browser enables you to determine which blocks of code have executed in your application. You can load one or more code coverage output files into the browser.

To display code coverage results in the Code Coverage Results Browser

- Select the **Code Coverage** command from the File menu in the Jade Platform development environment.

The Code Coverage Results Browser is then displayed. For details, see "Code Coverage", in Chapter 17 of the *Developer's Reference*.

Refactoring a Jade Method

The **Refactor** command in the Edit menu provides commands that enable you to restructure a component of an existing method.

Notes The refactoring functionality is enabled only when the AutoComplete feature is turned on; that is, you have checked the **Use AutoComplete** check box in the Display Options group on the Preferences dialog **Editor Options** sheet. (For details, see "Using Jade AutoComplete Functionality", in Chapter 2.)

Ctrl+F2 displays the Refactor menu at the cursor position. This menu is also displayed when you select the **Refactor** command from the Edit menu or by right-clicking in the editor pane and then selecting the **Refactor** command.

The Refactor menu is disabled for a **Workspace** method because a **Workspace** has no **self** class object and therefor the menu is not relevant.

The result of a refactoring action is displayed in the status line; for example:

```
Local variable docExample : String was added to the method vars
```

For details, see the following subsections.

- Creating a New Class
- Declaring an Unknown Local Identifier as a Local Variable
- Extracting Existing Logic and Creating a New Method
- Generating a Method Stub from the Editor Pane
- Promoting a Local Variable to be a Property on the Current Class
- Promoting a Method Value to be a Local Method Constant
- Promoting a Local Variable to be a Parameter of the Current Method
- Promoting a Local Constant to be a Type Constant
- Renaming or Changing an Entity

Creating a New Class

When editing a Jade method, you can create a new class without leaving the editor pane, even if the current method has been changed and not saved.

To add a new class

1. In the editor pane, select the **Add Class** command from the Refactor submenu of the Edit menu.

   The Define Class dialog is then displayed. For details, see "Defining Your Own Classes", in Chapter 3.

2. As the **Object** class is selected by default in the **Subclass of** combo box on the **Class** sheet of the Define Class dialog, you must select the class that you require as the superclass of the new class.

The class is added to the current schema and is displayed in the Class List (you may have to expand the appropriate superclass). The currently selected class, property, and method remain selected.

Applies to Version: 2018.0.01 and higher

Declaring an Unknown Local Identifier as a Local Variable

When editing a Jade method, you can declare an unknown local identifier as a local variable.

To declare a local identifier as a local variable

1. Position the cursor on the identifier.
2. Right-click and then select the **Declare Local Variable** command from the Refactor submenu of the Edit menu.
3. If the identifier is assigned by an expression or it is used as an assignment value, the type of the local variable is determined from that expression.

The type is determined by the type of a **foreach** variable using the type of expression in the **in** section of the **foreach** instruction. If the **in** expression is:

- A collection, the membership of the collection is used to define the variable type.
- *Not* a collection, the expression type is used as the variable type; for example, **Integer**.

If the type of the expression cannot be determined, the Declare Local Variable Name dialog is displayed, prompting you to specify or select in the **Variable Type** combo box the primitive type of the variable and then click the **OK** button.

Text in the status line displays the result of the action; for example:

```
Local variable docExample : String was added to the method vars
```

A local variable of the determined or selected type is added to the **vars** section of the method, which is created if it does not exist. The cursor remains at its previous position.

Examples of an unknown **indx** local identifier in a method are as follows.

```
indx := 0;

posn := indx;
```

Positioning the cursor over **indx**, where this is undefined, would result in the following code fragment.

```
vars
...
indx : Integer;
begin
...
```

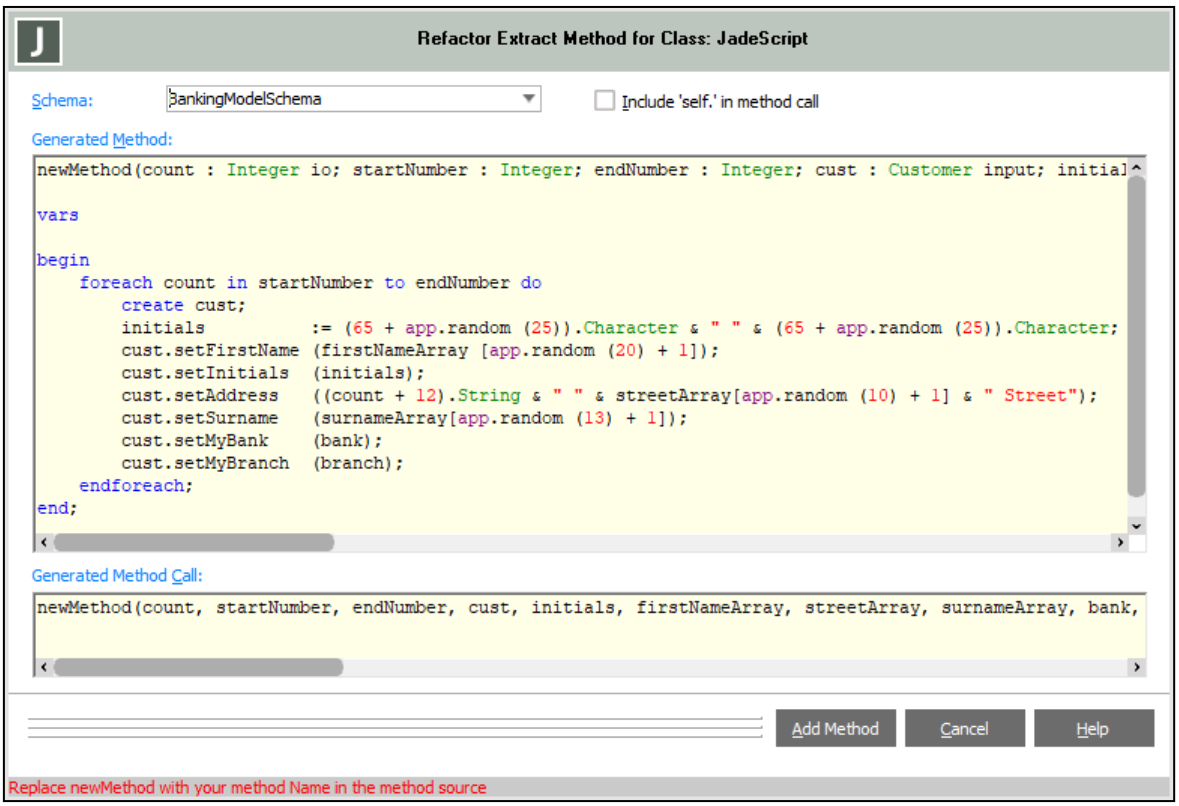
Extracting Existing Logic and Creating a New Method

When editing a Jade method, you can select logic within a method and create a new method in the currently selected class with that logic.

To extract existing logic and create a new method

1. In the editor pane, select the logic to be extracted.
2. Right-click and then select the **Extract Method** command from the Refactor submenu of the Edit menu. If the start of the selection includes a whole expression, the Refactor Extract Method for Class *class-name* dialog is then displayed. If it does not include a whole expression, an error message is displayed.

The **Generated Method** editor pane contains the source of the new method, consisting of the parameter signature determined by the analysis of the selected logic, the method template expansion if you have a template defined in your text templates preferences, and the selected logic, as shown in the following image.



The status line initially displays:

```
Replace newMethod with your method Name in the method source
```

3. The Enter and Esc keys initially control the dialog cancel and commit functions.

   - When the dialog is first displayed, a transparent text box containing the text **newMethod** overlays the method name in the **Generated Method** editor pane, and it has focus. This text box, which has a maximum length of 100 characters, must have a lowercase first character.
   - As you change the method name, the text box is updated and resized to fit, and the actual method source, template entry (if present), and method call text in the **Generated Method Call** editor pane are all adjusted.
   - The method name text is shown in red, which indicates that you can use the Esc and Enter keys to cancel the dialog or to perform the add method action, respectively.
   - You can use the right and down arrow keys, the Tab key, and the mouse to move to the **Generated Method** editor pane itself.
   - The text box is hidden as soon as it loses focus, the method name is displayed in its normal color in the editor pane, and the Esc and Enter keys no longer cancel or commit the dialog, because they are now editor-related.
   - You can still change the method name in the editor pane before you click the **Add Method** button.

Notes If you change the signature of the method, the text of the call that will replace the original selected text in the source method is not changed. You should also change this text in the editor pane of the Class Browser to meet your requirements.

If the text prior to the original text that you selected is a replacement statement and you selected one statement only, the return type of the method is set to the replacement type; for example, if the original method has the statement **int := (a * b + c);** and you select **(a * b + c)**, the result is a method return type of **Integer**.

4. In the **Schema** combo box, select the schema in which the method is to be created. This combo box, in which the current schema is selected by default, lists all of the schemas in which the method could be created.

5. If you want to ensure that **self.** prefixes the method call, check the **Include 'self.' in method call** check box. If **self.** exists in the method call, uncheck this check box to remove **self.** from the method call (the check box is checked if **self.** exists in the method call).

   The value of the **Include 'self.' in method call** check box is set automatically when the method name in the generated method is changed, based on whether **self.** prefixes the method call text.

6. Click the **Add Method** button, to add the new method and to replace the selected source with a call to the new method. Alternatively, click the **Cancel** button to cancel the current action and return focus to the original editor pane in the Class Browser.

Jade detects local variables used only in the selected logic of methods to be extracted. When the refactored method is created, they are defined as local variables in the new method rather than being passed as **io** parameters in the signature of the created method. (The definition of local variables is not affected in the original method.) If a variable is used in other logic not selected in the original method to be extracted, it is passed as an **io** parameter to the new method that is created.

When you extract a method for refactoring, a protected method is created if you have selected the **Protected** access option button in the Methods group box on the **Schema** sheet of the Preferences dialog. In the editor pane, manually remove the **protected** option from the signature of the created method if it is not required.

If the calling method text is changed to **self.*method-name***, the change is accepted, provided that the *method-name* in the call matches the generated method name when you click the **Add Method** button.

When you click the **Add Method** button, the new method is added to the currently selected class in the selected schema, using the method name specified in the **Generated Method** edit pane, and the selected content of the original source method is replaced with the contents of the **Generated Method Call** editor pane. This last change is not committed, so you can undo it by using the Ctrl+Z shortcut keys.

If the new method compiles successfully, the Refactor Extract Method for Class *class-name* dialog is closed and focus returns to the Class Browser editor pane of the original method.

If the new method fails to compile:

- The Refactor Extract Method for Class *class-name* dialog remains open and states that the new method was added but that the compile failed. The first error is displayed and the source generating the error is selected.
- The **Generated Method Call** editor pane is hidden and a message is displayed, stating that the original selection has been replaced by the text that was in the **Generated Method Call** editor pane.
- The **Add Method** button is renamed **Recompile**. Clicking this button attempts to recompile the method.

  If the recompile succeeds (following your changes to the method source), the dialog is closed.
- Clicking the **Cancel** button leaves the new method in error and the original method changed.

- The new method logic calls a method where the parameters are usage **io** or **output** and Jade has not detected that the parameter for the new method signature should be **io** or **output** (that is, no assignment is done on the variable in the selected logic or it is not used as the variable of a **foreach** instruction).

  Change the signature of the new method, to handle this situation.

The following is an example of a code fragment selected in the original source (**Generated Method**) editor pane.

```
d := (a * b + c);
```

The following example is the generated method in which the whole line is selected (where **doCalc** has been specified as the method name).

```
doCalc(d: Integer io; a: Integer; b: Integer; c: Integer);
vars
begin
    d := (a * b + c);
end;
```

The original statement then becomes:

```
doCalc(d, a, b, c);
```

If **(a * b + c)** were selected, the method would be:

```
doCalc(a: Integer; b: Integer; c: Integer): Integer;
vars
begin
    return (a * b + c);
end;
```

Generating a Method Stub from the Editor Pane

When editing a Jade method, you can generate a method stub (whose logic must be filled in later) from a call statement.

» To generate a method stub from the editor pane

1. Specify a method call with all of the required parameters.
2. Position the cursor over the new method name and then select the **Generate Method Stub** command from the Refactor submenu of the Edit menu.

A new method stub is then defined on the receiver of the method call. The types of the parameters are determined from the call statement. If the method call is assigned, the method return type is the type of the assignment variable.

If the receiver of the method is not in the current schema, a dialog is displayed, asking you to select the schema in which to add the method. For a class, the schema of the root of the class is selected by default. For a primitive method, the current browser schema is selected by default.

An example of an undefined method call is:

```
bool := cust.anewMethod(custName, bank.id, descStr);
```

This method call results in the creation of the **anewMethod** method stub on the **Customer** class, as follows.

```
anewMethod(custName: String;
           integer1: Integer;
           descStr:  String): Boolean;
vars
```

```
begin
    return null;
end;
```

When generating a method stub from the editor pane, note that:

- The AutoComplete feature must be turned on.
- The parameter can be an identifier or dot (**.**) expression only.

  The *Unable to determine the required method signature for <method name>. The type of a parameter cannot be determined.* message is displayed if the expression is invalid or an identifier is unknown.
- The method name must begin with a lowercase character.
- If successful, a message box displays the schema, class, and name of the method that was created. If rejected, a message box displays the reason why the method stub generation was not performed.
- All of the elements in the call statement, apart from the method name, must be known elements.
- Any template defined for methods in your user preferences is invoked.
- The Jade development security library is called, to ensure that you have access to the schema in which the method will be defined when it is not the currently selected schema.
- The generated method compilation could fail, and a warning message be displayed; for example, if the parameter types are not visible in a superschema.
- If the method is defined on a superclass or subclass, the generation will be rejected if the signature of the method does not match. A warning will also be displayed, advising you that the new method will be a re-implementation or superclass instance.
- If the method is defined on a superclass or subclass, the parameter names and return type is taken from the existing method.
- If the parameter of the method call is a local variable, parameter, or local constant, the name will be used as the parameter name in the method (with a numeric postscript, if the same variable is used more than once).
- Other types of parameters will use the type name of the parameter with a numeric postscript; for example, **integer1**.
- If the current unary expression selected in the editor pane is preceded by the **write** instruction, as shown in the following code fragment, a return type of **String** is assumed.

  ```
  write accountToReportOn.toString();
  ```

  The current unary expression preceded by ***unary-expression*** **:=** also handles the type of the expression.

Promoting a Local Constant to be a Type Constant

When editing a Jade method, you can promote a local method constant to a constant on the type in which the method is defined.

To promote a local constant to a type constant

1. Select the local constant reference in the method (in the logic or in the **constants** section).
2. Select the **Promote to Type Constant** command from the Refactor submenu of the Edit menu. The Add constant to *schema-name*::*type-name* dialog is then displayed. The name of the constant in the dialog is set to the local constant name. The value and the type of the constant are set in the editor pane.
3. Set any attributes that you require and then click the **OK** button.

The constant is then added to the type (that is, the class or primitive type) and the local constant definition is removed from the method.

If the line containing the constant definition will then be empty, the entire line is removed. If there is any other text on the line (including comments), only the constant definition portion is removed.

To undo the text change in the method, press Ctrl+Z. The constant definition is not returned.

Promoting a Local Variable to be a Parameter of the Current Method

When editing a Jade method, you can promote a local variable to be a parameter of the method.

To promote a local variable to a method parameter

1. Position the cursor on the name of a local variable (in logic or in the **vars** section).
2. Right-click and then select the **Promote to Parameter** command from the Refactor submenu of the Edit menu.

The local variable is then removed from the local variables (vars) section and added as the last parameter of the method. Any usages of the method must be edited to include the parameter. The signature change does not take effect until you compile the method. To undo the change, press Ctrl+Z.

If you attempt to promote a user identifier to be a method parameter and the identifier is not a local method variable, an error message of the following form is displayed.

```
Identifier identifier-name is a type-name not a local variable.
```

For example, if you attempt to promote an existing method parameter called **saleItem**, the following error message would be displayed.

```
Identifier saleItem is a method parameter not a local variable.
```

Promoting a Local Variable to be a Property on the Current Class

When editing a Jade method, you can promote a local variable to a property on the class in which the method is defined.

To promote a local variable to a property on the class

1. From the editor pane, select the local variable reference in the method (in the logic or in the **vars** section).
2. Select the **Promote to Class Property** command from the Refactor submenu of the Edit menu.

   If the caret is positioned in a local variable, the Define Attribute or Define Reference dialog is displayed, depending on the type of the local variable. The name of the property in the dialog is set to the local variable name and the dialog property type is set to the type of the local variable.
3. Set any attributes that you require and then click the **OK** button.

The property is then added to the class and the local variable definition is removed from the method.

If the line containing the variable definition will then be empty, the entire line is removed. If there is any other text on the line (including comments), only the relevant variable definition portion is removed.

To undo the text change in the method, press Ctrl+Z (the property definition will remain).

Promoting a Method Value to be a Local Method Constant

When editing a Jade method, you can promote a method string, numeric, or boolean value to a local method constant.

To promote a method value to a method constant

1. Click on a string, numeric, or boolean value within the logic of the method.
2. Select the **Promote to Method Constant** command from the Refactor submenu of the Edit menu.

   The Set name for Local Method Constant Value dialog is then displayed, showing the selected value.
3. In the **Constant Name** text box, specify the name that you require for the local method constant.
4. Click the **OK** button.

The constant is then added to the **constants** section of the method (which is created if it does not exist) and the selected instance of that constant in your method is replaced by the specified name.

Note The **Promote to Method Constant** command does not replace all instances of that constant, because the value can appear in unrelated contexts; for example, **false**, **0**, and so on.

To undo the change or changes, press Ctrl+Z to undo each change in that method.

Renaming or Changing an Entity

You can change or rename almost any identifier selected within the body of a method in the editor pane (for example, a local variable or a method) of the Class Browser, by positioning the cursor in the entity and then selecting the **Rename / Change** command from the **Refactor** submenu of the Edit menu (or you can press Shift+F2). This then attempts to process the identifier under the cursor to perform a rename or change. Alternatively, you can select the whole identifier.

Note The Jade development security library is called, to ensure that you have access to the schema in which the entity will be renamed when it is not defined in the currently selected schema.

The result of the refactor action depends on the type of the identifier, as follows. When the:

- Identifier is a local constant, variable, or method parameter, the Rename *entity-name* dialog is then displayed so that you can specify the required name in the **New Name** text box.

  When you specify the new name and click the **OK** button, the local identifier is renamed throughout the method and the cursor is positioned after the identifier.

  To undo a change, press Ctrl+Z for each identifier position in the method.

  Note The change or rename action is performed, even if the method has been modified and is unsaved.

- Variable is a user-defined global constant, the Global Constant maintenance dialog is displayed for that constant definition, regardless of the schema in which the global constant is defined. You can change or rename the global constant definition.

  After the change or rename action is committed, the cursor is positioned after the global constant name.

You cannot use Ctrl+Z to undo this change.

Notes This action is rejected if the method is changed or locked, because changes made will cause that method to be recompiled.

This action is equivalent to browsing to the global constants in the schema in which it is defined and then selecting the **Change** command from the Constants menu.

- Variable is a method, the Rename Method dialog is displayed, regardless of the class or schema in which the method is defined.

  After the rename action is committed, the cursor is positioned after the method name.

  You cannot use Ctrl+Z to undo this change.

  Notes This action is rejected if the method is changed or locked, because changes made will cause that method to be recompiled.

  This action is equivalent to browsing to the method in the Class Browser and then selecting the **Rename** command from the Methods menu.

- Variable is a class constant, the Define Constant dialog is displayed, regardless of the schema or class in which the class constant is defined. After the change or rename action is committed, the cursor is positioned after the constant name.

  You cannot use Ctrl+Z to undo this change.

  Notes This action is rejected if the method is changed or locked, because changes made will cause that method to be recompiled.

  This action is equivalent to browsing to the class constant in the Class Browser and then selecting the **Change** command from the Constants menu.

- Variable is a class property, the respective Define Attribute or Define Reference dialog is then displayed, regardless of the schema or class in which the property is defined. After the change or rename action is committed, the cursor is positioned after the property name.

  You cannot use Ctrl+Z to undo this change.

  Notes This action is rejected if the method is changed or locked, because changes made will cause that method to be recompiled.

  This action is equivalent to browsing to the property in a Class Browser and then selecting the **Change** command from the Properties menu.

  If you rename a property of a class that is versioned (either the current or latest version), the reorganization treats the property as being new and it deletes the prior property name content. To prevent loss of data, you should therefore rename properties when the class is not versioned.

- Variable is a class name, the Define Class dialog is displayed, regardless of the schema in which the class is defined. After the change or rename action is committed, the cursor is positioned after the class name.

  You cannot use Ctrl+Z to undo this change.

Notes This action is rejected if the method is changed or locked, because changes made will cause that method to be recompiled.

This action is equivalent to browsing to the class in a Class Browser and then selecting the **Change** command from the Classes menu.

If the cursor is not positioned on one of the variables in this list or it is a package or an interface entity, a message box is displayed advising you that the cursor is not positioned on an entity that can be changed or renamed in this way.

The action can also be rejected because:

- The entity cannot be changed for a variety of standard reasons; for example:
  - The new name already exists as a local entity such as a parameter, local variable, or constant
  - A property, method, or constant already exists in the selected class or its superclasses
- You are not entitled to change that entity (that is, security library rules apply)

Notes Normal versioning and patch versioning rules apply, based on the schema of the entity that is changed.

Modifying a type constant in the *current* version of a versioned schema is disallowed.

Extracting a Method Selected in a Methods List

You can extract a single method from a list (for example, after a references request of an entity) in the same way you can when you select a method in the Methods List of the Class Browser and then select the **Extract** command from the Methods menu.

» To extract a single method selected in a methods list

1. Select the method from a list of methods (for example, in the list showing the search results in the Global Find/Replace Results form).
2. Select the **Extract Method** command from the Methods menu. The Extract dialog is then displayed, to enable you to select the options that you require. For details, see "Specifying Your Extract Options", in Chapter 10. The common Save As dialog is then displayed.

   Note The **Extract Method** command is disabled if the current method has been modified but not saved. Moving the cursor over the disabled command displays *Not available - Method has been modified but not saved*.
3. Click the **Browse** button at the right of the **Schema File Name** text box and then specifySpecify the name and location of your method file, as follows.
   - The file name is constructed as *class-name_method-name.file-suffix*, defaulting to the name of the current type and method, with an .**mth** suffix (for example, when extracting the **Customer** class **getAddress** method, the default file name is **Customer_getAddress.mth**).
   - The file suffix is as defined in the **Method** text box of the File Suffixes group box on the **Miscellaneous** sheet of the Preferences dialog.
   - The directory to which the extracted method is saved defaults to your Jade working directory.
4. Complete the **Selective** and **Schema Options** sheets on the Extract dialog, as required. For details about completing the **Selective** and **Schema Options** sheets, see "Specifying Selective Options" and "Specifying Your Schema Options", respectively, in Chapter 10.

Applies to Version: 2016.0.01 and higher

Applies to Version: 2022.0.01 and higher

Defining Properties

You can add a property to a user-defined class at any time. When you have defined a property, it can be accessed for any instance of the class in which it is defined or for any instance of a subclass of that class. (See also "Promoting a Local Variable to be a Property on the Current Class", earlier in this chapter.)

Attribute properties are primitive types such as strings or collections of primitive types (for example, a string array) that are characteristics (or features) of an object. Attributes do not possess identity but the objects to which they apply determine their individuality.

Reference properties contain references to other objects; that is, they are end points in one- or two-directional relationships.

When you add a new property to a class and the Properties List displays the **Attributes**, **References**. **Controls**, or **Menu** sheet and the new property does not match the type that is currently displayed, focus changes to that, the **All** sheet and the new property is visible and selected.

Notes Properties cannot be added to, changed, or removed from system classes. Adding, changing, or deleting a property may cause the class to be marked for reorganization if instances of that class already exist. (A class is not normally marked for reorganization when you add a reference that is an exclusive collection.)

For details about defining references to ActiveX automation events, see "Using Automation Events", in Chapter 4 of the *External Interface Developer's Reference*.

For details about:

- Renaming a property selected within the body of a method in the editor pane, see "Renaming or Changing an Entity", later in this chapter.
- Creating a method mapping for a property selected in a hierarchy browser, see "Adding a Method Mapping for a Property", later in this chapter.
- Dynamic design-time and runtime properties, see "Dynamic Clusters and Properties", later in this chapter.
- The scope of design-time dynamic properties, including a table providing usage decisions, see "Design-Time Dynamic Property Scope and Notes", later in this chapter. See also "Runtime Dynamic Properties", in Chapter 21 of the *Developer's Reference*.

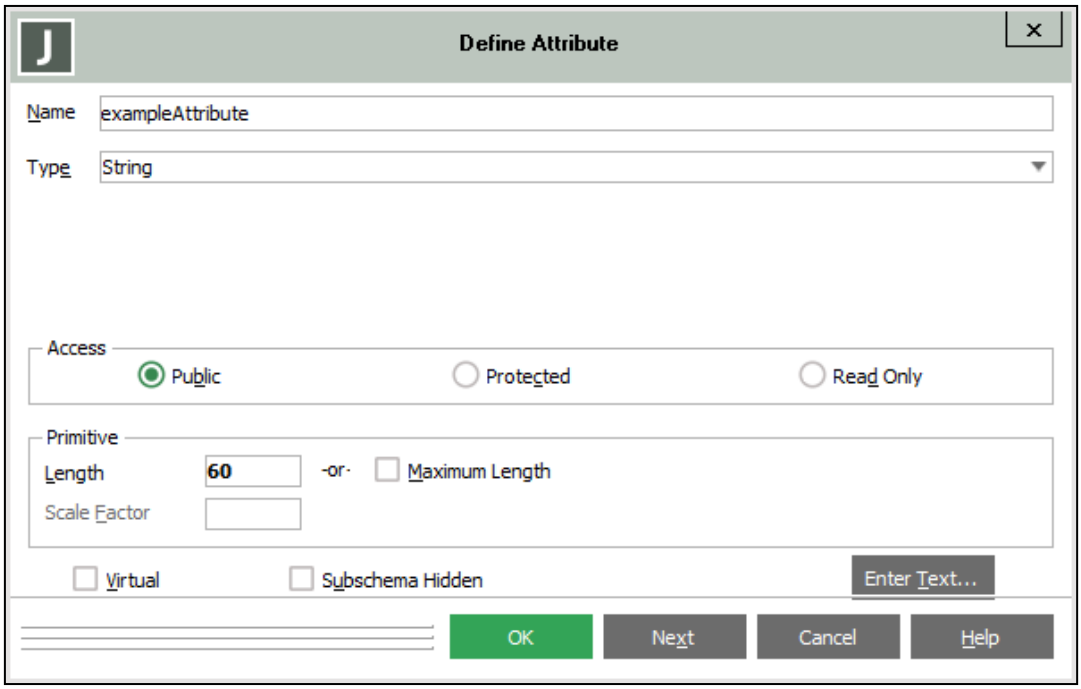
Adding an Attribute Property

Use the **Add Attribute** command from the Properties menu to add a static attribute property to the class currently selected in the Class Browser. (For details about adding a dynamic attribute, see "Defining an Attribute Property" under "Dynamic Clusters and Properties".)

» To add an attribute property

1. In the Class Browser window, select the class to which the attribute is to be added.
2. Select the **Add Attribute** command from the Properties menu.

The Define Attribute dialog, shown in the following example, is then displayed.



The Define Attribute dialog enables you to define your new attribute property for the selected class.

3. In the **Name** text box, specify the name of the attribute property that you want to define.

   The attribute name must be unique to the class to which it is being added and it cannot exist in any of its superclasses or subclasses.

4. In the **Type** combo box, perform one of the following actions.

   - Select the required type in the Type list.
   - Specify the first character or the first few characters of the type in the text box and then select the required type from the Type list. The selected type is then displayed in the **Type** combo box.

   The type can be any primitive type or any primitive array class defined in the schema.

5. In the Access group box, select the appropriate type of access if you do not want access to the property to be protected. The **Protected** option button is selected by default, so that the property can be accessed only by methods defined in the selected class and its subclasses. (A protected property is displayed in the Properties List of the Class Browser with a padlock icon to the left of the property.)

   Select the **Public** option button if you want the property to be read and modified by any other method in the schema. (A property that has public access is displayed in the Properties List of the Class Browser with the public icon to the left of the property.)

   Select the **Read Only** option button if you want the property to be read by other methods in the schema but not updated. (A read-only property is displayed in the Properties List of the Class Browser with the read-only icon to the left of the property.)

Note If the property is of type **StringArray**, **IntegerArray**, or one of the other primitive array types, the access mode setting applies to the array object rather than to the elements within the array; that is, a setting of read-only does not prevent array elements being added, deleted, or updated.

6. If you have specified a **Decimal**, **Binary**, or **String** primitive type for the property, specify the length of the property in the **Length** text box of the Primitive group box. Alternatively, check the **Maximum Length** check box if you want the length of the property to be an unbounded string of the maximum length. (The **Length** text box is disabled when you check the **Maximum Length** check box.)

   The default and maximum lengths are listed in the following table.

Type	Default Length	Maximum Bounded Length	Maximum Unbounded Length
Decimal	12	23	Not applicable
String	30	100,000-1	**Max_UnboundedLength** in the **SystemLimits** global constants category
Binary	30	100,000-1	**Max_UnboundedLength** in the **SystemLimits** global constants category

7. If you have specified a **Decimal** primitive type for the property, specify the number of decimal places for the property in the **Scale Factor** text box of the Primitive group box.

8. Check the **Virtual** check box if you want the value of the property to be derived at run time and never stored in the database. (A virtual attribute cannot be a collection.)

   A virtual attribute property requires a defined mapping method that is activated when the value of the attribute is set or retrieved.

9. Check the **Subschema Hidden** check box if you want to specify that the attribute property is available only in the local schema; that is, it is not available for use in any subschemas.

   In a subschema copy class, you cannot define a mapping method for a **subschemaHidden** property that is defined in a superschema. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

10. Click the **Enter Text** button if you want to specify or maintain descriptive text for the attribute property as part of the definition or maintenance of the attribute. For details, see "Specifying Text for a Schema Element", in Chapter 3.

    Tip You can also specify descriptive text for the attribute at any time, by selecting the **Text** command from the Properties menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

11. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another attribute property.

The specified attribute property is then displayed in the Properties List of the Class Browser and the values that you defined for the property displayed in the editor pane.

If a property template has been defined for all new properties in your Jade development database or for your own Jade properties, the property template is displayed below the attribute property details in the editor pane when you click the **OK** button on the Define Attribute dialog. For details, see "Maintaining Text Templates", in Chapter 2.

Adding a Reference Property

Use the **Add Reference** command from the Properties menu to add a new static reference to the selected class. You can add a reference to the selected class at any time. (For details about adding a dynamic reference, see "Defining a Dynamic Reference Property" under "Dynamic Clusters and Properties".)

A *reference* is a property that contains a reference to another object; that is, it is an end point in a one- or two-directional relationship. The two types of reference are:

- An *implicit* reference, in which an object references another object and either of the following is true:
  - The reference object does not contain a reference back to the first object.
  - The referenced object contains a reference to the first object but the two properties have not been defined as end points in a two-way relationship by using the **Define Inverse** button on the Define Reference dialog.
- An *inverse* (or *explicit)* reference, in which two objects reference each other and the two properties have been defined as end points in a two-way relationship by using the **Define Inverse** button on the Define Reference dialog.

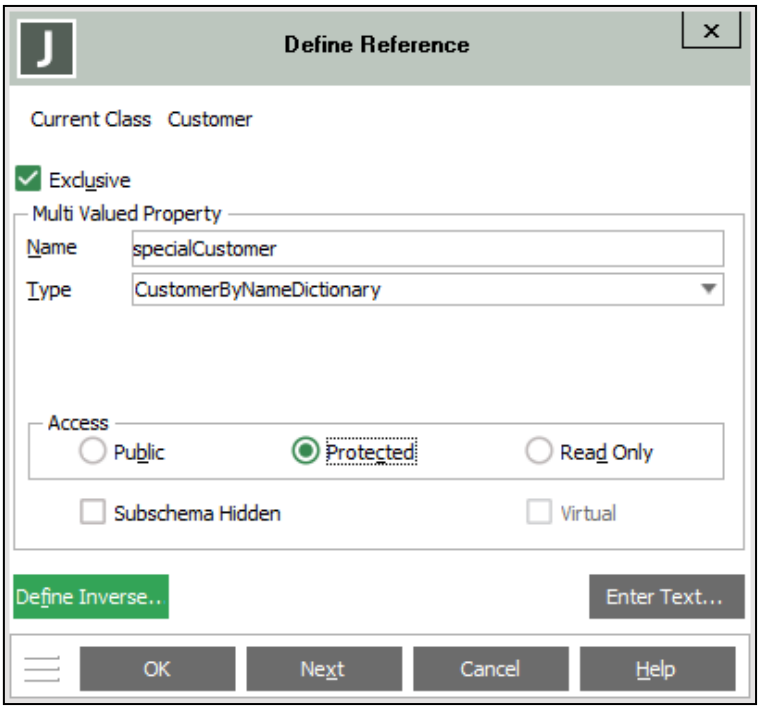
Tips The advantage of using an inverse relationship instead of an implicit relationship is that one side of an inverse relationship can be maintained automatically by Jade.

For details about using condition constraints to maintain automatic inverse references when the specified conditions are satisfied, see "Defining and Compiling Jade Methods and Conditions" and "Adding Conditions to Classes or Primitive Types", earlier in this chapter.

To add a reference property

1. In the Class Browser, select the class to which the reference is to be added.
2. Select the **Add Reference** command from the Properties menu.

The Define Reference dialog, shown in the following image in an example of its initial form, is then displayed.



An extended form of the dialog is displayed when you specify that you want to define an inverse reference. For details, see "Defining an Inverse Reference Property", later in this chapter.

Defining an Implicit Reference

The Define Reference dialog enables you to define a new reference property for the selected class.

To define a reference property

1. If you want the reference property to be exclusive to the current class only, check the **Exclusive** check box. Exclusive reference properties can be specified when the type of the property is a collection subclass.

   An exclusive reference property is not embedded in the class structure. The collection is created and destroyed automatically when the parent object is created and destroyed.

2. In the **Name** text box, specify the name of the property that you want to define. The name cannot be the same as the name of any existing property or non-mapping method in the class, its superclasses, or subclasses.

3. In the **Type** combo box, perform one of the following actions.

   - Select the required type in the Type list.
   - Specify the first character or the first few characters of the type in the text box, and then select the required type from the Type list. The selected type is then displayed in the **Type** combo box.

   The type can be any class in the current schema or its superschemas. The **Define Inverse** button is disabled if the selected type is defined in a superschema, as inverses cannot be defined between classes in different schemas (the superschema class does not have visibility to the subschema class).

   The type can be any primitive array class defined in the schema.

4. In the Access group box, select the appropriate type of access if you do not want access to the property to be protected. The **Protected** option button is selected by default, so that the property can be accessed only by methods defined in the selected class and its subclasses. (A protected property is displayed in the Properties List of the Class Browser with a padlock icon to the left of the property.)

   Select the **Public** option button if you want the property to be read and modified by any other method in the schema. (A property that has public access is displayed in the Properties List of the Class Browser with the public icon to the left of the property.)

   Select the **Read Only** option button if you want the property to be read by other methods in the schema but not updated. (A read-only property is displayed in the Properties List of the Class Browser with the read-only icon to the left of the property.)

   Note If the property is of type **StringArray**, **IntegerArray**, or one of the other primitive array types, the access mode setting applies to the array object rather than to the elements within the array; that is, a setting of read-only does not prevent array elements being added, deleted, or updated.

5. Check the **Virtual** check box if you want the value of the implicit reference property to be derived at run time and never stored in the database.

   A virtual reference property must have a defined mapping method that is activated when the value of the property is set or retrieved.

   If you check the **Virtual** check box, the **Define Inverse** button is disabled.

6. Check the **Subschema Hidden** check box if you want to specify that the reference property is available only in the local schema; that is, it is not available for use in any subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

7. Click the **Enter Text** button if you want to specify or maintain descriptive text for the reference property as part of the definition or maintenance of the reference. For details, see "Specifying Text for a Schema Element", in Chapter 3.

   Tip You can also specify descriptive text for the reference at any time, by selecting the **Text** command from the Properties menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

8. Click the **OK** button. (Alternatively, click the **Define Inverse** button if you want to define an inverse for this property. Inverse reference properties are optional.)

If a property template has been defined for all new properties in your Jade development database or for your own Jade properties, the property template is displayed below the reference property details in the editor pane when you click the **OK** button on the Define Reference dialog. For details, see "Maintaining Text Templates", in Chapter 2.

Using Inverse Reference Properties

An inverse property is one in which two objects reference each other. One end of this inverse reference property can be automatic; that is, the Jade Object Manager establishes the return reference for you.

For example, if you have two classes called **Driver** and **Car**, if the **Driver** class contains a property **myCar**, the inverse of that property could be the property **myDriver** in the **Car** class. Inverse reference properties can be:

Reference	Usage
one-to-one	myDept of **Manager** = myManager of **Department** In this example:

Reference	Usage
	■ *myDept* is a property of type **Department** in the **Manager** class ■ *myManager* is a property of type **Manager** in the **Department** class
one-to-many	allEmployees of **Department** = myDept of **Employee** In this example: ■ *allEmployees* is a set with membership of **Employee** objects in the **Department** class ■ *myDept* is a property of type **Department** in the **Employee** class
many-to-many	allMailGroups of **Employee** = allMembers of **MailGroup** In this example: ■ *allMailGroups* is a collection containing **MailGroup** objects in the **Employee** class ■ *allMembers* is a collection containing **Employee** objects in the **MailGroup** class

For inverse reference properties, you must specify one of the update modes listed in the following table.

Mode	Description
Automatic	The property cannot be updated in user logic but is automatically maintained by the system in response to the user updating the inverse or deleting the parent of the inverse.
Manual / Automatic	Either the property being maintained or the inverse can be updated in user logic. The other side of the reference is automatically maintained by the system.
Manual	The property being maintained is always updated in user logic, except when the parent object of the inverse is deleted. The inverse is always maintained by the system.

A reference property is a *one* relationship and a collection property is a *many* relationship. If a property defining the relationship is not a collection, only one reference is allowed. The collection can be a set or a dictionary.

To add the same property on both sides of an inverse relationship:

1. The relationship type must be peer-to-peer.
2. The update mode must be **Manual / Automatic**.
3. The relationship must be one-to-one or many-to-many.

Only dictionaries with member keys can be used to define relationships.

Notes Mapped properties (that is, properties that have a mapping method) from a **RootSchema** class cannot be used as dictionary keys.

A reference on a class can have itself as an inverse reference if the inverse is defined as a **peer-to-peer** relationship, the update mode is **Manual** / **Automatic**, and the reference is **one-to-one** or **many-to-many**. Applications that require two objects of the same class can therefore have a reference between themselves so they can use the same **myObject** reference with the inverse being the **myObject** reference (for example, in a double-ledger accounting system with two equal and opposite transactions so that you do not have to define two attributes so that you can define the inverse reference).

For more details, see "Automatic Key Maintenance" and "Resolving Exceptions", under "Maintaining Dictionary Key Paths", in Chapter 3.

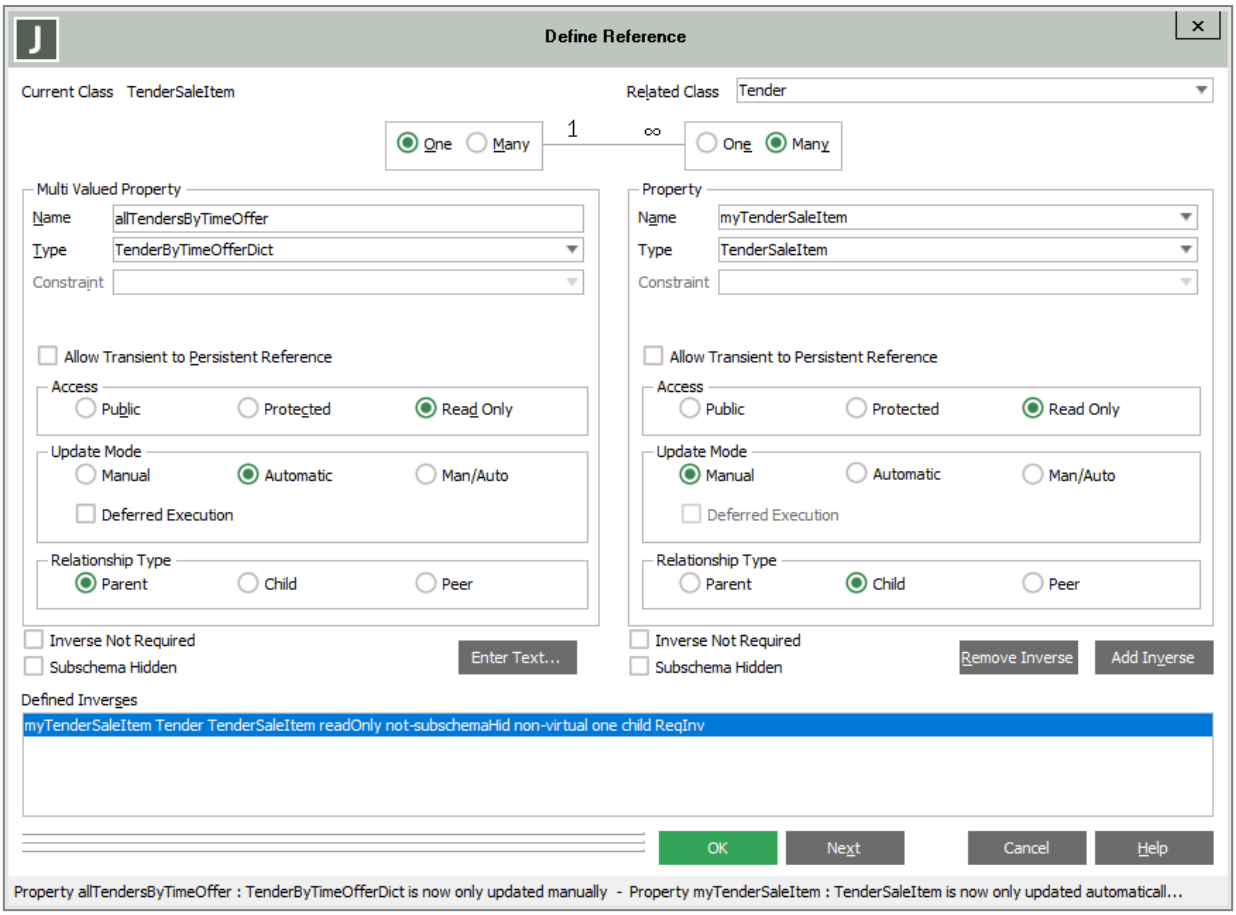
For details about using constraints to maintain automatic inverse references when the specified conditions are satisfied, see "Defining and Compiling Jade Methods and Conditions" and "Adding Conditions to Classes or Primitive Types", earlier in this chapter.

Defining an Inverse Reference Property

Use the extended Define References dialog, displayed when you select the **Define Inverse** button in the initial Define Reference dialog, to specify the inverse reference. When you specify an inverse property, you are defining a relationship between two classes. The two properties represent the *end points* of the relationship.

The values and options that you specified in the initial Define Reference dialog are displayed in the Property group box at the left of the extended dialog. The related class is also displayed, as well as existing inverse reference properties for the class.

The following is an example of the extended Define Reference dialog.



The status bar at the bottom of the dialog displays the exact type of the relationship that is being created when you select an access, update mode, or relationship type option button.

Check boxes and access, update mode, and relationship type option buttons display bubble text indicating their function when you hover over the control.

Values that cannot be altered are disabled. The default values that are set for an inverse reference property are:

- If the property on the left side of the dialog is a reference to a collection, the One / Many option on the right side of the dialog is set to **Many**.
- The **Type** list contains only those classes that are valid end points for the relationship.

To define an inverse reference property

1. Check the **Subschema Hidden** check box if you want to specify that the reference property is available only in the local schema; that is, it is not available for use in any subschemas. This check box is checked if you checked it on the initial form of the Define Reference dialog, documented under "Defining an Implicit Reference", earlier in this chapter. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.
2. Specify the cardinality of the relationship between the two classes, by selecting the **One** or the **Many** option for the class on the left side of the dialog. This may change the classes listed in the Type list on the right side of the dialog and the caption of the group box.

   When you select a property on the left side of the dialog that is a reference to a collection, the **Many** option on the right side of the dialog is automatically set, the group box caption is displayed as Multi-Valued Property, and the **Type** list is populated only with collection classes.

   Tip For details about filtering the entries displayed in the list box associated with the **Type** combo box for the current class on the left of the form, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.

   When you select the **One** option on the right side of the dialog, the caption is displayed as Property.
3. The **Name** combo box contains any properties in the related class that are valid end points for the relationship. If you select an existing property from the **Name** list box, an inverse is added between the property in the current class and the property selected in the related class. If you specify a name for the property in the **Name** text box of the related class, a new property is added to the related class, and an inverse is added between the two properties.

   Note If the required inverse property is not available for selection, you *may* be able to establish the inverse relationship from the other side (that is, define the required inverse property by opening the Define Reference dialog for the inverse property you require for the end-point and then add the inverse relationship from *that* property).
4. If you added a new property in the **Name** list box, perform one of the following actions in the **Type** combo box.
   - Select the required type in the Type list. (The list displays only the types that are valid for the inverse.)
   - Specify the first character or the first few characters of the type in the text box, and then select the required type from the Type list. The selected type is then displayed in the **Type** combo box.
5. If a condition method has been defined for the class on which an automatic update mode is defined or a superclass of that class, select the condition that you require from the **Constraint** list box. This list box is enabled only for a property that has the **Automatic** option button selected in the Update Mode group box, and it does not contain any constraints if you have not defined any for that class or superclass or there are no conditions without parameters.

   When the manual side of an inverse reference is set, the property on the automatic side of the reference is set only when the selected constraint is satisfied. For details about using condition constraints to maintain automatic inverse references when the specified conditions are satisfied, see "Defining and Compiling Jade Methods and Conditions" and "Adding Conditions to Classes or Primitive Types", earlier in this chapter.

6. In the Access group box, select the appropriate type of access if you do not want access to the property to be protected. The **Protected** option button is selected by default, so that the property can be accessed only by methods defined in the selected class and its subclasses. (A protected property is displayed in the Properties List of the Class Browser with a padlock icon to the left of the property.)

   Select the **Public** option button if you want the property to be read and modified by any other method in the schema. (A property that has public access is displayed in the Properties List of the Class Browser with the public icon to the left of the property.)

   Select the **Read Only** option button if you want the property to be read by other methods in the schema but not updated. (A read-only property is displayed in the Properties List of the Class Browser with the read-only icon to the left of the property.)

7. Check the **Allow Transient to Persistent Reference** check box if you want the current class (left side) or the related class (right side) to reference a persistent object from a transient object without its inverse being maintained. For new references, this check box on both the left and the right sides of the relationship is unchecked by default, indicating that transient to persistent references are not allowed.

   When transient to persistent references are allowed, no inverses are updated when a persistent object is referenced from a transient object. This enables you to create transient structures that replicate persistent data for querying and analysis or for editing purposes without having to define multiple reference properties (one with an inverse and one without) and the associated code to maintain the appropriate reference based on the lifetimes of the objects involved. For example, given an inverse relationship between two classes **A.myB inverseOf B.myA**, this enables you to specify that where **myB** on a transient instance of **A** is updated to reference a persistent instance of **B**, **myA** on the persistent instance of **B** is *not* updated to reference the transient instance of **A**.

8. In the Update Mode group box on the right side of the dialog, select the type of update required for your property. (The update mode on the left side for the current class is set automatically; for example, setting **Manual** on the right side always results in the left side being **Automatic** unless the definition is for a condition.)

   The **Manual** option specifies that the property is maintained by your Jade logic. If you select the **Automatic** option, the property is maintained by Jade whenever the property on the other side of the relationship is updated. Automatic references cannot be updated in user logic. The **Manual / Automatic** option specifies that either side of the relationship can be updated in user logic. The other side of the relationship is automatically maintained by Jade.

9. In the Update Mode group box, check the **Deferred Execution** check box (it is unchecked by default) if you want to defer execution of an automatic or multi-valued inverse property.

   Note You can check or uncheck this check box only when the **Automatic** or **Man/Auto** option button is selected (that is, it is disabled when the **Manual** update mode is selected or the corresponding **Type** is not a **Collection** or a **Collection** type that does not support deferred update modes).

10. In the Relationship Type group box on the right side of the dialog, select the type of relationship required. (The type of relationship on the left side for the current class is set automatically.)

    A peer-to-peer relationship is an equivalent relationship between two objects. A parent-child relationship is a relationship in which the child object belongs to or is subsidiary to the parent object. When a parent object is deleted, any child objects are automatically deleted. The relationship is between instances of two classes. For example, if you select the **Parent** option in the Relationship Type group box on the right side of the dialog, the class displayed in the **Related Class** text box is the class of the parent object in the relationship. (The class displayed in the **Current Class** list box is the class of the child object.)

11. Check the **Inverse Not Required** check box for the current class or the related class if you want to specify that no exception is raised when:

    - No inverse is set during automatic maintenance because the type of the object is not compatible with any of the inverse references
    - No inverse is set during automatic maintenance because the constraint fails

    The last inverse is removed as the result of a change to a property. For details about raising exceptions when specific conditions are not met, see "Defining and Compiling Jade Methods and Conditions", in Chapter 4. These check boxes are not checked, by default.

12. Check the **Subschema Hidden** check box for the current class or the related class if you want the inverse to be available only in the local schema. This check box is unchecked by default; that is, the inverse is available in subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

    Inverses that are subschema-hidden are not available for use in any subschemas. They cannot be referenced by subschema code, and they are not displayed in the Properties List in subschema Class Browsers.

13. If you want more than one inverse reference for the property on the left side of the dialog, click the **Add Inverse** button to add the specified inverse reference to the list of defined inverse references. (Conversely, you can remove an existing inverse reference by selecting the reference listed in the **Defined Inverses** list box and then clicking the **Remove Inverse** button.)

14. Click the **OK** button.

The specified property is then displayed in the Properties List of the Class Browser and the attributes that you defined for the property are displayed in the editor pane.

Dynamic Clusters and Properties

You can:

- Add dynamic clusters to a class from the Jade Platform development environment or at run time.
- Add design-time dynamic properties to a class from the Jade Platform development environment.
- Extract from and load design-time dynamic properties to a schema (.**scm**) file.
- Reference design-time dynamic properties directly in methods.

Add dynamic properties to existing classes to implement new or changed functionality without requiring a database reorganization of installations where the changes are installed. These dynamic properties are displayed and can be used in the same way as static properties, including as key elements in **MemberKeyDictionary** classes.

You can declare one or more property clusters on a class. Dynamic properties can be added rather than static properties, to avoid the need for a reorganization.

When a dynamic property has been defined, it can be used in most places where static properties can be used; for example, in packages, ODBC, keys, and exposures (but not in RPS, Jade audit access, or ad hoc indexes). The two types of dynamic properties are:

- Runtime dynamic properties, which:
    - Are created, changed, or deleted only at run time by user logic.
    - Do not require any reorganization when added, changed, or deleted.

- Are displayed in the Jade Platform development environment for a class (the default text color is magenta) but cannot be changed or deleted using the development environment.
- Are not included in schema (.**scm**) files with the class.
- Cannot be directly referred to in Jade methods. Access is by calls to the **getPropertyValue** and **setPropertyValue** methods using the property name.
- Reorganization is not available to handle any impact caused; for example, an added inverse is not populated.
- Cannot be used in exposures, ODBC, packages, RPS, as keys, in an inverse with a static or design-time property on the other side, Jade audit access, or a delete command action in a Jade Control File (JCF) **commandFile** parameter in the batch Schema Load utility (**jadloadb**).

For details, see "Runtime Dynamic Properties", in Chapter 21 of the *Developer's Reference*.

- Design-time dynamic properties, which:
  - Can be created and maintained using the Jade Platform development environment.
  - Are displayed in the development environment for a class (the default text color is maroon).
  - Can be directly accessed in Jade methods.
  - Even if a reorganization is not required when a design-time dynamic property is added, changed, or deleted, you are given the option of versioning the class and the property if the change was not made in the latest version of a versioned schema.
  - The addition, change, or deletion of a primitive type, exclusive primitive array, or ordinary object reference does not require a reorganization except for the deletion of a non-embedded property.

    Note This is the main benefit of using design-time dynamic properties, as a mutate reorganization is not required in most cases. If versioning is used, the reorganization process needs only to instantiate the new class definition.

  - Reorganization is required when inverses are added and therefore there is really no advantage in adding a design-time dynamic inverse versus using a static property, unless both sides of the new inverse are new design-time dynamic properties.
  - Can be used in exposures, ODBC, packages, as keys, in an inverse with a static property on the other side, and in a delete command action in a Jade Control File (JCF) **commandFile** parameter in the batch Schema Load utility (**jadloadb**).
  - Cannot be used in RPS, Jade audit access, or as keys in an ad hoc index.

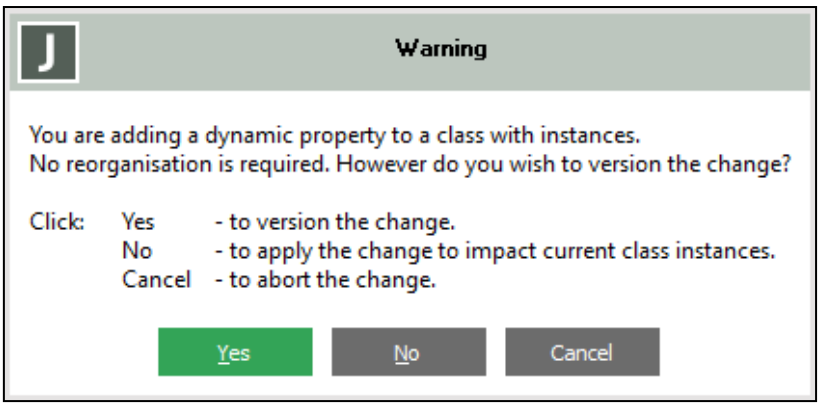
The **Window** sheet of the Preferences or Jade Installation Preferences dialog enables you to change the color with which the text of dynamic design-time and runtime property names are displayed in the Properties List of a hierarchy browser. By default, design-time dynamic properties are displayed in maroon and runtime dynamic properties in magenta.

For details about defining dynamic property clusters, see "Defining Dynamic Clusters", in Chapter 3. For details about defining dynamic attribute and reference properties, see "Defining a Dynamic Attribute Property" and "Defining a Dynamic Reference Property", respectively, in the following subsections.

The following limitations apply when using dynamic clusters and properties.

- Dynamic properties cannot be added to **Collection** classes.
- A dynamic property cannot be a subobject (an exclusive collection).
- The transaction that adds a new runtime dynamic property must be committed before the new property can be used. This allows the notification that the property has been added to be sent and processed, and avoids multiple updater conflicts over cluster definition changes.
- A class can have a maximum of 255 local dynamic property cluster definitions.
- A class can have a maximum of 65,535 local properties, both static and dynamic.
- Dynamic property clusters and dynamic properties cannot be added to system schema classes.

When you click the **OK** button on the Define Dynamic Property or Define Dynamic Reference dialog, a message box similar to that shown in the following image is displayed if the class is not versioned or the selected schema is not the latest.



Clicking:

- **Yes** versions the schema, class, and cluster if required, and adds the new property to the latest version.
- **No** adds the property to the current version and the property immediately becomes live and available.
- **Cancel** aborts the add process and closes the dialog.

A similar message box is displayed if the class is not versioned or the selected schema is not the latest, so that the change can be versioned, directly applied, or aborted.

Deleting a design-time dynamic property also displays a similar message box if the class is not versioned or the selected schema is not the latest, allowing the delete action to be versioned, directly applied, or aborted. Note that the deletion of a non-embedded design-time dynamic property (exclusive, slob, or blob) requires a reorganization to perform the delete action.

The process of changing a dynamic property is the same as that for a static property, except that:

- You cannot change the defined dynamic property cluster.
- You cannot change the type from a primitive type to a collection type, or the reverse.
- You cannot change a collection type if the class has instances, unless you change it to a superclass.

- Primitive type data changes are handled at run time, converting the previous stored type into the new type on a *get* method call and storing the data in the new format on a *set* method call.

For details about the scope of design-time dynamic properties, including a table providing usage decisions, see "Design-Time Dynamic Property Use" and "Deciding Between Static and Design-Time Dynamic Properties".

Design-Time Dynamic Property Use

The following table summarizes the conditions that require a database reorganization when using design-time dynamic properties. Reorganization is required only when a class has instances, so an implied condition for the table is that there are instances of the class containing the design-time dynamic property.

When there are no instances, there is no advantage in using a design-time dynamic property.

Action on Design-Time Dynamic Property	Condition	Reorg is Required
Adding an attribute		No
Changing an attribute	When the attribute is a primitive type; for example: ■ **String** to **Integer** ■ **String[30]** to **String[50]** **Note** The type of a primitive-array attribute cannot be changed.	No
Deleting an attribute	When the attribute is not a slob, blob, or primitive array; for example: ■ **String[30]** ■ **Date**	No
	When the attribute is a slob, blob, or primitive array; for example: ■ **String[1000]** ■ **IntegerArray**	Yes
Adding a reference	When the reference does not have an inverse	No
	When the reference is made an inverse of an existing reference	Yes
	When the reference and its inverse are dynamic, and both are added at the same time	No
Changing a reference	When the reference is changed to a superclass type (the only change of type that is allowed)	No
Deleting a reference	When the reference does not have an inverse	No
Deleting a reference	When the reference is an inverse reference	Yes
Changing a condition used as a constraint		Yes

One reason for preferring design-time dynamic properties over static properties is to avoid a lengthy database reorganization. For more details, see "Deciding Between Static and Design-Time Dynamic Properties", later in this chapter.

Deciding Between Static and Design-Time Dynamic Properties

As a design-time dynamic property can be used wherever a static property can be used, there are a number of factors in deciding the type of property to use.

- If there are instances of the class containing the property and you want to avoid a reorganization, use a design-time dynamic property because:
  - A database reorganization is not required when a design-time dynamic property is added, changed, or deleted. However, a database reorganization would be required for a static property.
- If there are no instances of the class containing the property, use a static property because:
  - Accessing a static property in an application is faster than accessing a design-time dynamic property.
  - A database reorganization is not required when a static property is added, changed, or deleted. There is no advantage in using a design-time property.

Defining a Dynamic Attribute Property

Use the **Add Dynamic Attribute** command from the Properties menu to add a dynamic attribute property to the class currently selected in the Class Browser.

» To add a dynamic attribute property

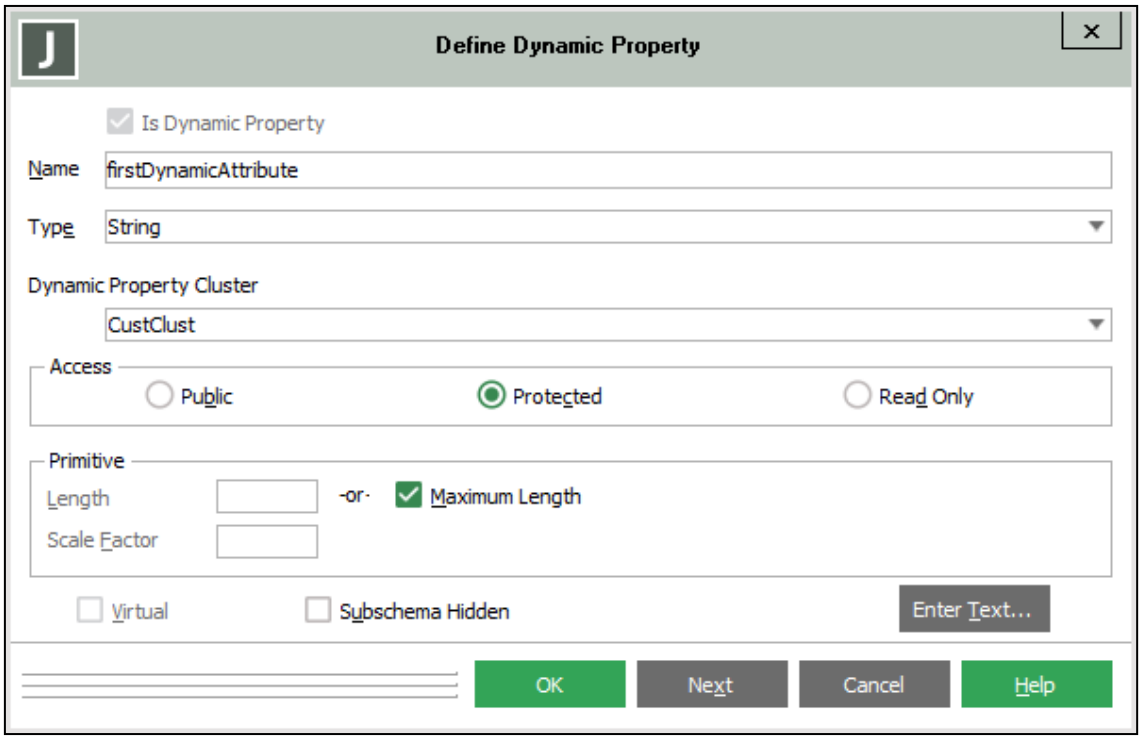
1. In the Class Browser window, select the class to which the dynamic attribute is to be added.

   Notes You can add a dynamic attribute property only to a non-collection class in which a dynamic cluster has been defined.

   Design-time attribute properties cannot be added, changed, or deleted if the owner class is currently in use.

2. Select the **Add Dynamic Attribute** command from the Properties menu.

The Define Dynamic Property dialog, shown in the following example, is then displayed.



The read-only **Is Dynamic Property** check box is checked, and cannot be unchecked. (Use the Define Attribute dialog to define a static attribute property.)

3. In the **Dynamic Property Cluster** combo box, perform one of the following actions.

   - Select the dynamic property cluster in which you want to define the dynamic attribute property.
   - Specify the first character or the first few characters of the dynamic property cluster in the text box and then select the required cluster from the list. The selected cluster is then displayed in the **Dynamic Property Cluster** combo box.

   The type can be any primitive type or any primitive array class defined in the schema.

All other values are the same as those on the Define Attribute dialog for static properties. For details, see "Adding an Attribute Property".

For details about the message box display when you click the **OK** button, see "Dynamic Clusters and Properties".

Defining a Dynamic Reference Property

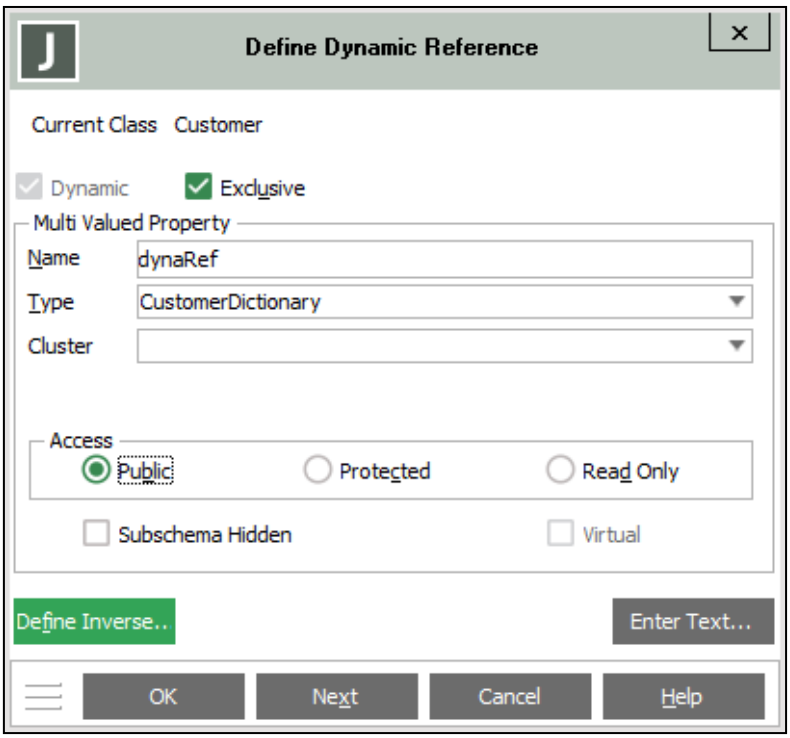
Use the **Add Dynamic Reference** command from the Properties menu to add a new dynamic reference to the selected class. You can add a reference to the selected class at any time.

Notes You can add a dynamic reference property only to a non-collection class in which a dynamic cluster has been defined.

Design-time reference properties cannot be added, changed, or deleted if the owner class is currently in use.

To add a reference property

1. In the Class Browser, select the class to which the reference is to be added.
2. Select the **Add Dynamic Reference** command from the Properties menu. The Define Reference dialog, shown in the following image in an example of its initial form, is then displayed.



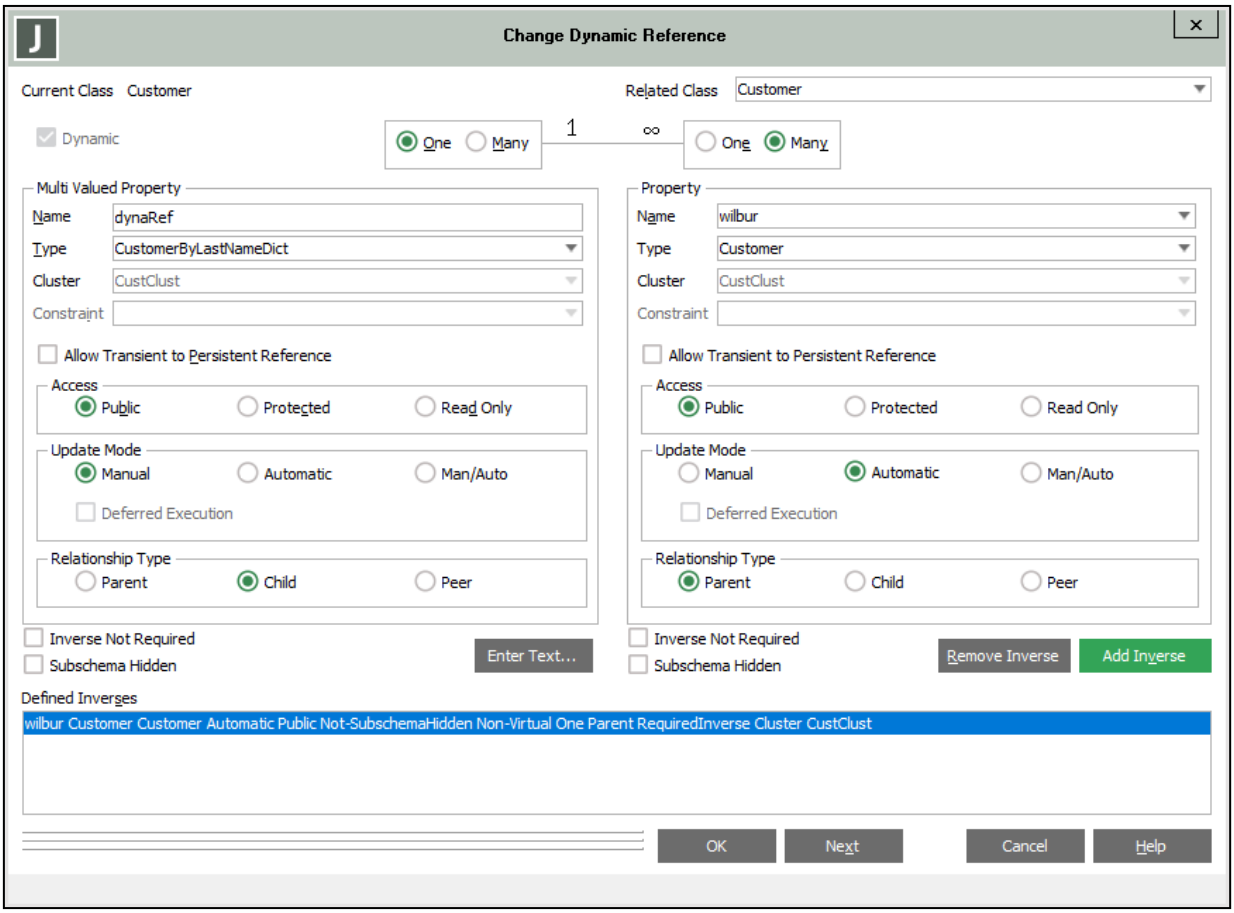
The read-only **Dynamic** check box is checked, and cannot be unchecked. (Use the Define Reference dialog to define a static reference property.)

You can change the type to a superclass type only if instances of that class exist.

3. In the **Cluster** combo box, perform one of the following actions.
   - Select the dynamic property cluster in which you want to define the dynamic reference property.
   - Specify the first character or the first few characters of the dynamic property cluster in the text box and then select the required cluster from the list. The selected cluster is then displayed in the **Cluster** combo box.

All other values are the same as those on the Define Reference dialog for static properties. For details, see "Adding a Reference Property".

The following image is an example of the dynamic reference dialog expanded for inverse definition.



You must select a dynamic cluster for a right-hand side property that is to be added. (It will be added as a design-time property.)

Adding, changing, or deleting an inverse involving design-time properties requires a reorganization to check and populate the associated inverses.

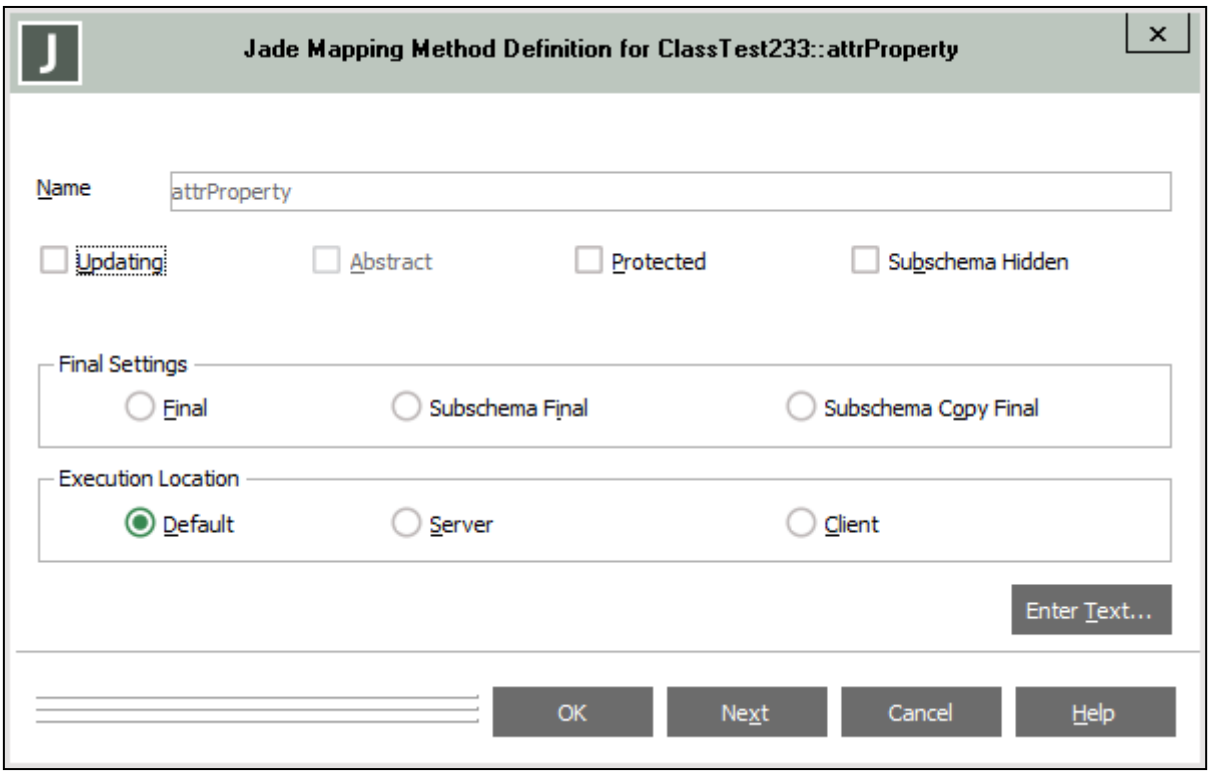
For details about the message box display when you click the **OK** button, see "Dynamic Clusters and Properties".

Adding a Method Mapping for a Property

Use the **Create Mapping Method** command from the Properties menu to add a method mapping for the property selected in the Properties List of a hierarchy browser (for example, the Class Browser). (For details about mapping methods, see "mapping Option", in Chapter 1 of the *Developer's Reference*.

The **Create Mapping Method** command is enabled if a mapping method can be created for the selected property, in which case the menu command changes to **Create Mapping Method For Property '*property-name*'**. If a mapping method already exists or a mapping method cannot be created for a selected property, the command is disabled.

When you select the command, the Jade Mapping Method Definition dialog, shown in the following image, is then displayed, with the selected property displayed and disabled in the **Name** text box. In addition, the dialog title is displayed as Jade Mapping Method Definition For *class-name*::*property-name*.



For details about the controls on this dialog, see "Adding Jade Methods to Classes or Primitive Types", earlier in this chapter.

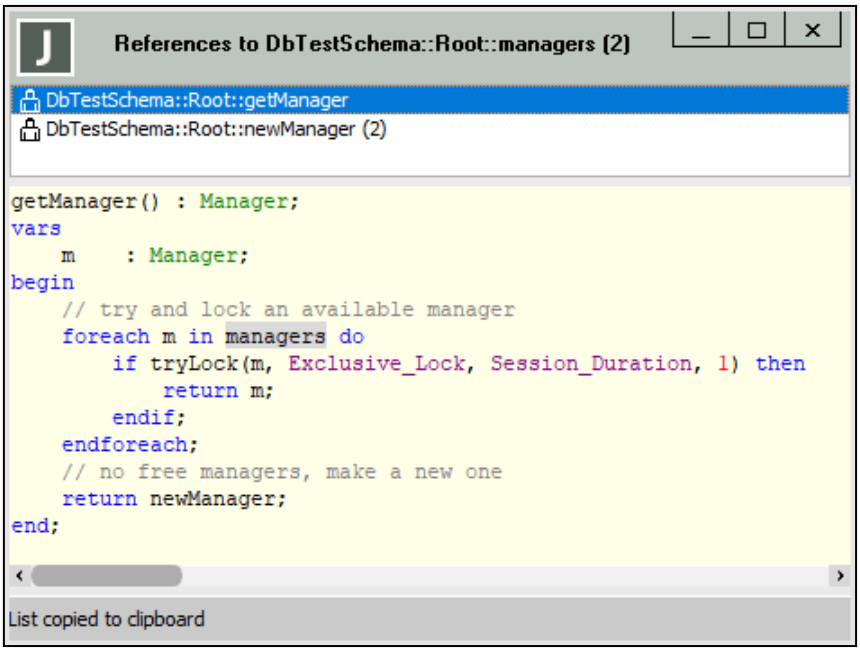
You can change the mapping method options and then click the **OK** button. If the method is added, the appropriate mapping method source is constructed. (Alternatively, click the **Cancel** button to abandon the addition of the mapping method.)

Applies to Version: 2016.0.02 (Service Pack 1)

Displaying all References to the Selected Property or Constant

Use the **References** command from the Properties menu or Constants menu to display all references in the selected schema to the property selected in the Properties List of the Class Browser or the constant selected in the Constants List of the Class Browser, Primitive Types Browser, or Interface Browser.

The References window for the selected property or constant is then displayed, as shown in the following image.



This window lists all methods and their classes that reference the selected property or constant, and enables you to view specific references to the property or constant.

In addition, it displays all constants in the current schema that reference the constant selected in the Constants List of the Class Browser, Primitive Types Browser, or Interface Browser.

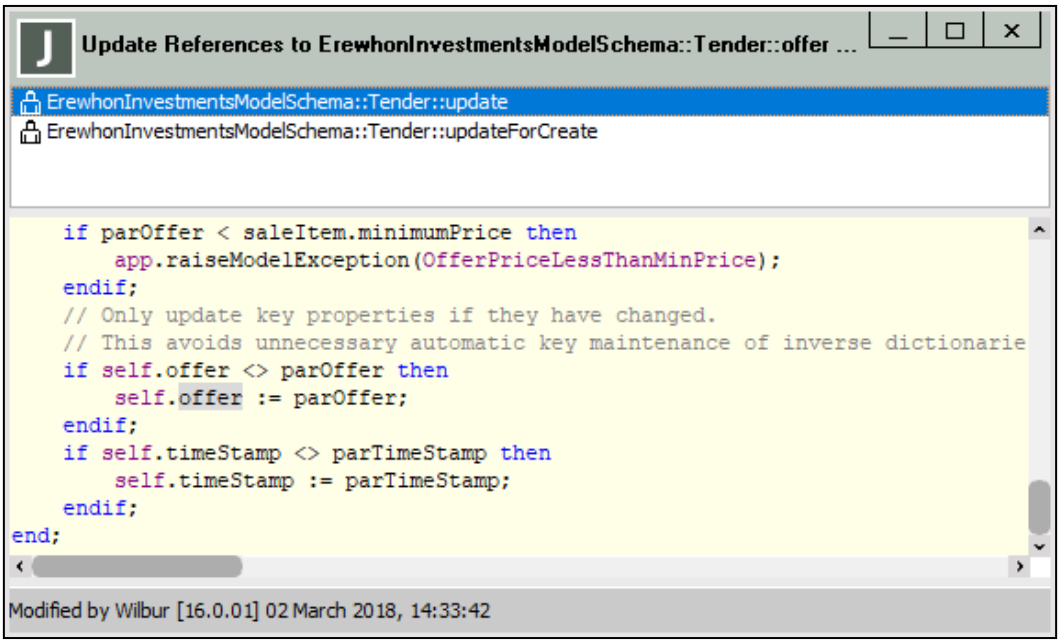
In the References window, select the method whose reference to the selected property or constant you want to view. The details of the selected method are then displayed in the editor pane.

You can maintain a method displayed in the editor pane of the References window. (For details, see "Compiling Methods", earlier in this chapter.)

Displaying all References that Can Update the Selected Property

Use the **Update References** command from the Properties menu to display all references in the selected schema that can update the property selected in the Properties List of the Class Browser.

The Update References window for the selected property is then displayed, as shown in the following image.



This window lists all method references and their classes that can update the selected property, and enables you to view specific update references to the property.

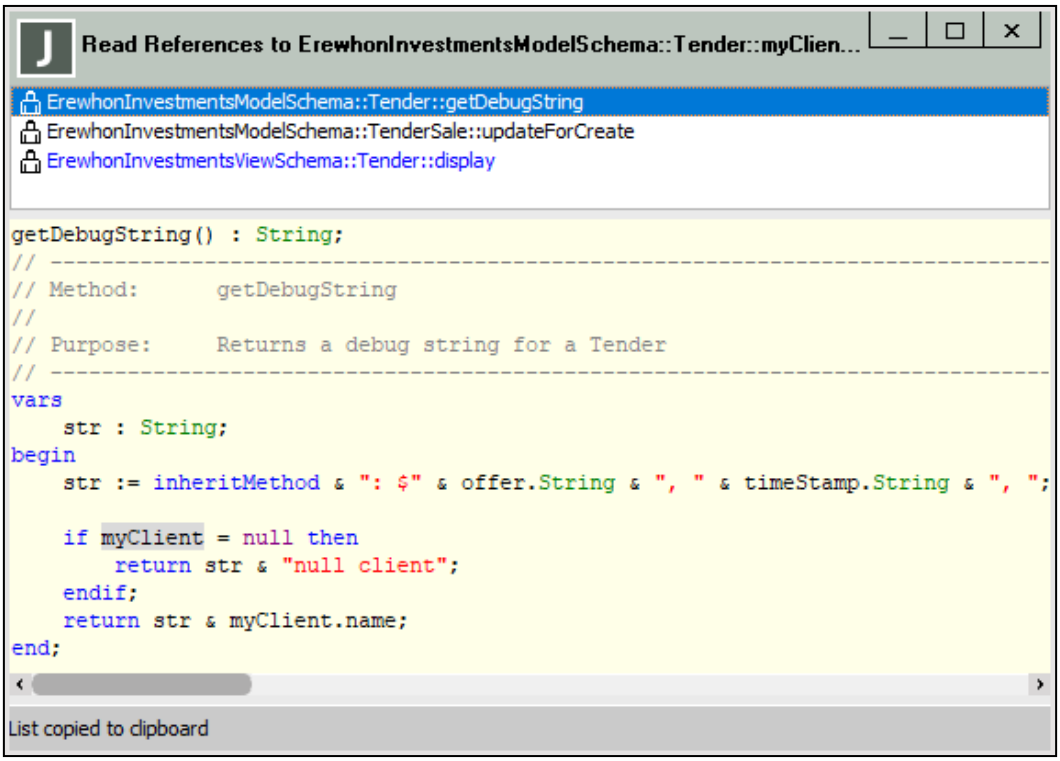
In the Update References window, select the method whose update reference to the selected property you want to view. The details of the selected method are then displayed in the editor pane.

You can maintain a method displayed in the editor pane of the Update References window. (For details, see "Compiling Methods", earlier in this chapter.)

Displaying all References that Can Read the Selected Property

Use the **Read References** command from the Properties menu to display all references in the selected schema that can read the property selected in the Properties List of the Class Browser.

The Read References window for the selected property is then displayed, as shown in the following image.



This window lists all method references and their classes that can read the selected property, and enables you to view specific read references to the property.

In the Read References window, select the method whose read reference to the selected property you want to view. The details of the selected method are then displayed in the editor pane.

You can maintain a method displayed in the editor pane of the Read References window. (For details, see "Compiling Methods", earlier in this chapter.)

Displaying all Local References to the Selected Property

Use the **Local References** command from the Properties menu to display all references in the selected class and its subclasses that reference the property selected in the Properties List of the Class Browser.

The Local References window for the selected property is then displayed, listing all methods in the class and its subclasses that reference the selected property. In the upper pane of the Local References window, select the method whose reference to the selected property you want to view. The selected method is then displayed in the editor pane, with each local property reference highlighted.

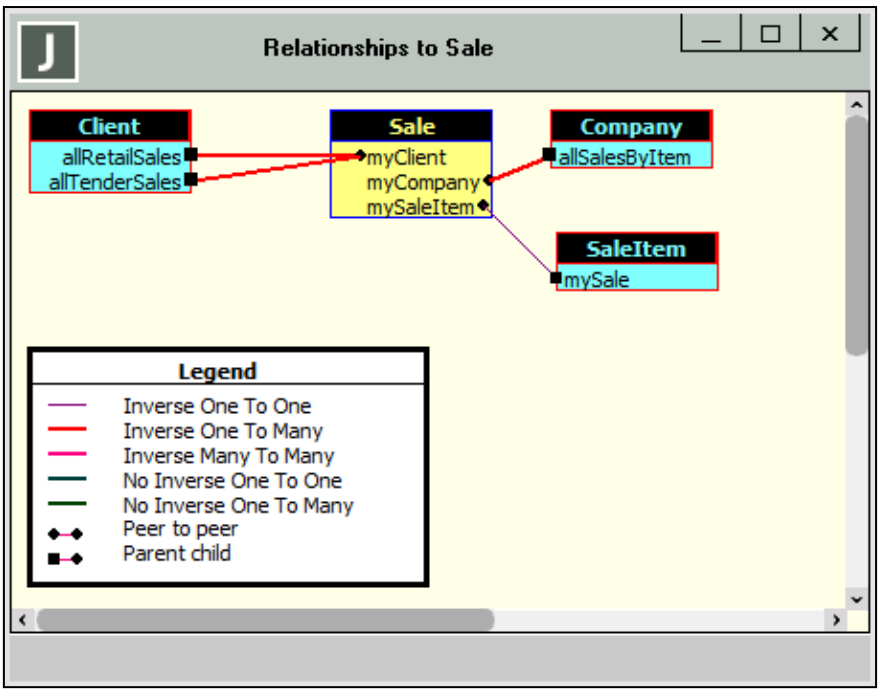
You can maintain a method displayed in the editor pane of the Local References window. (For details, see "Compiling Methods", earlier in this chapter.)

Displaying all Relationships to a Class

Use the **Relationships** command from the Classes menu to display a graphical representation of all relationships of the class currently selected in the Class List of the Class Browser to other classes in the schema. The types of relationships that can be displayed are:

- Inverse one-to-one
- Inverse one-to-many
- Inverse many-to-many
- No inverse one-to-one
- No inverse one-to-many
- Parent-child
- Peer-to-peer

The Relationship View window for the current class, shown in the following image, is then displayed.



This window displays inverse (explicit), no inverse (implicit), parent-to-child, and peer-to-peer relationships.

Note You can view the displayed relationship model only; you cannot modify it by adding, changing, or deleting relationships using this graphical representation.

You can print the relationship model, if required.

Using the Relationship View Window

For inverse relationships, the relationship line is drawn between the two properties that define that relationship.

For relationships with no inverse, the relationship line is drawn between the property and the class.

The one-to-many and many-to-many relationships are displayed with a line thicker than that between one-to-one relationships.

In the Relationship View window, you can:

- Move any rectangle other than that representing the selected class to suit your own requirements; for example, to make the display more aesthetic for printing purposes.

  Note The new positions are not saved, and the positions are recalculated every time the graph is redrawn.

- Double-click on a rectangle other than that representing the selected class, to open a new Relationship View window displaying the relationships for the class that represents that selected rectangle.
- Print the relationship graph, by using the **Print Selected** command from the File menu. The output is printed in landscape format and may be split across multiple pages, if necessary.
- Change the graphical display, by using the View menu commands.

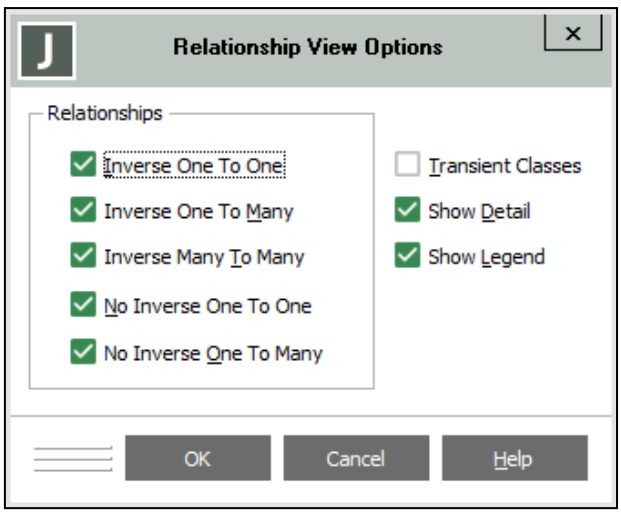
Specifying Your Relationship View Window Options

You can change the display of relationships in the Relationship View window to meet your requirements.

To select your relationship options

1. Select the **Options** command from the View menu in the Relationship View window.

   The Relationship View Options dialog, shown in the following image, is then displayed.



2. To hide the display of a relationship, uncheck the check box in the Relationships group box of the relationship that you do not want displayed. By default, all types of relationship are displayed in Relationship View window.

   When a check box is checked, the relationship line is drawn between the two properties in the Relationship View window that define the relationship.

   The relationship types whose display you can hide or make visible are as follows.

   - **Inverse One To One**, showing or hiding the display of inverse (explicit) one-to-one relationships and draws a relationship line between the two properties that define the relationship when checked.

- **Inverse One To Many**, showing or hiding the display of inverse (explicit) one-to-many relationships and draws a relationship line between the two properties that define the relationship when checked.
- **Inverse Many To Many**, showing or hiding the display of inverse (explicit) many-to-many relationships and draws a line between the two properties that define the relationship when checked.
- **No Inverse One To One**, showing or hiding the display of one-to-one relationships that have no inverse relationship (that is, implicit relationships) and draws a relationship line between the property and the class that define the relationship when checked.
- **No Inverse One To Many**, showing or hiding the display of one-to-many relationships that have no inverse relationship (that is, implicit relationships) and draws a relationship line between the property and the class that define the relationship when checked.

3. If you want to display both transient and persistent classes in the relationship, check the **Transient Classes** check box. By default, only persistent instances of classes are displayed.
4. If you do not want the properties involved in a relationship to be displayed, uncheck the **Show Detail** check box. By default, properties involved in a relationship are displayed.

   If you uncheck this check box, no properties are displayed and a single line is displayed to indicate the relationship between two classes.
5. If you do not want the relationship legend displayed at the bottom of the Relationship view window, check the **Show Legend** check box. By default, the relationship legend is displayed.

   The relationship legend displays the line indicating each type of relationship in the Relationship View window.
6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The current Relationship View window then has the focus, and displays your selected options.

Zooming in to and out from the Relationship View Window

You can enlarge or reduce the rectangles and font in the current Relationship View window to meet your requirements.

The font size is enlarged or decreased in increments of two points, up to a maximum size of 30 points or down to a minimum of two (2) points.

To zoom in on the current Relationship View menu

- Select the **Zoom In** command from the View menu.

The rectangles and the text within rectangles in the area of the current Relationship View window in which the caret is currently positioned are then enlarged; that is, zoomed in.

You may have to scroll horizontally or vertically to view the full relationship graph.

To zoom out from the current Relationship View menu

- Select the **Zoom Out** command from the View menu.

The rectangles and the text within rectangles in the area of the current Relationship View window in which the caret is currently positioned are then decreased; that is, zoomed out.

Defining Class, Primitive Type, and Interface Constants

You can add a constant to a class, primitive type, or interface at any time. Constants defined in methods are available only in the method in which they are defined.

Notes Global constants are associated with a schema and its subschemas. Class constants are associated with a class and its subclasses. Primitive type and interface constants are associated with a primitive type or interface, respectively.

For details about renaming a user-defined global or class constant selected within the body of a method in the editor pane, see "Renaming or Changing an Entity", later in this chapter.

Adding a Class, Primitive Type, or Interface Constant

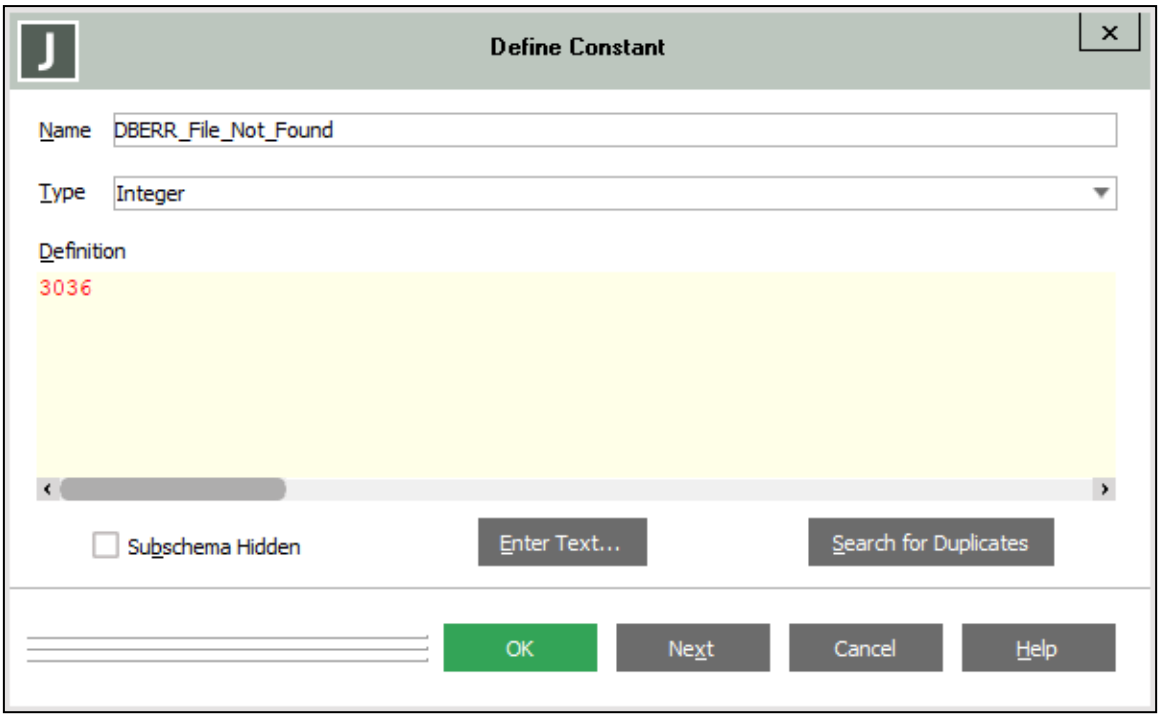
To add a class, primitive type, or interface constant

1. In the Class Browser, Primitive Types Browser, or Interface Browser, select the class, primitive type, or interface to which the constant is to be added.
2. Select the **Add** command from the Constants menu.

Tips The Constants menu is available only when the tab of the **Const** folder is selected in the Properties List of the Class Browser or when the Primitive Types Browser or Interface Browser is the current window. (The Properties List folders are shown in the image under "Using the Class, Primitive Types, or Interface Browser", in Chapter 3.)

See also "Promoting a Local Constant to be a Type Constant", earlier in this chapter.

The Define Constant dialog, shown in the following image, is then displayed.



Defining a Class, Primitive Type, or Interface Constant

The Define Constant dialog enables you to define your new constant for the selected class, primitive type, or interface.

To define a constant

1. In the **Name** text box, specify the name of the constant that you want to define.

   The constant name starts with an initial capital letter; for example, **ExampleConst**. The name must be unique to the class, primitive type, or interface to which it is being added.

2. In the **Type** combo box, select the required primitive type in the Type list or specify the first character or the first few characters of the primitive type in the text box, and then select the required primitive type from the Type list. The selected primitive type is then displayed in the Type text box.

   The type can be any primitive type to which you want to assign a constant value; for example, an integer value of **1** or a string value of **"samples\sharedem"**.

3. In the **Definition** editor pane, specify the definition of your constant. The definition can be a simple literal value or a more-complex expression. The following example shows the editor pane definition of a **File** subclass constant of primitive type **String**, named **DefaultDirectory**.

   ```
   "samples\sharedem"
   ```

4. Check the **Subschema Hidden** check box if you want to specify that the class constant is available only in the local schema; that is, it is not available for use in any subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

5. Click the **Enter Text** button if you want to specify or maintain descriptive text for the class, primitive type, or interface constant as part of the definition or maintenance of the constant. For details, see "Specifying Text for a Schema Element", in Chapter 3.

   Tip You can also specify descriptive text for the constant at any time, by selecting the **Text** command from the Constants menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

6. Click the **Search for Duplicates** button if you want to determine if a constant with an identical definition value exists in any superschema, the current schema, or any superclass, or a global constant with the same definition exists in the current schema or any superschema.

   Note Jade can locate duplicate global constants only if you selected a type in the **Type** combo box. If the type of global constant is not specified, no duplicate definitions are found.

   If an existing definition value is located, a message box displays the name of the constant containing that definition and the class, primitive type, or interface, or global constant category and schema in which it is defined.

   Tip This enables you to use an available existing constant without duplicating an identical definition in a constant category, class, primitive type, and interface in the same schema, for example.

7. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another constant.

The specified constant is then displayed in the Constant List of the Class Browser, Primitive Types Browser, or Interface Browser, with the values that you defined for the constant displayed in the editor pane. (A class, primitive type, or interface constant has public access and is displayed in the Constants List with the public icon to the left of the constant, by default.)

If a constant template has been defined for all new constants in your Jade development database or for your own class constants, the constant template is displayed below the class constant in the editor pane when you click the **OK** button on the Define Constant dialog. For details, see "Maintaining Text Templates", in Chapter 2.

Constant Definition Tips

When defining a constant value, the value of a constant can be a simple literal value or an expression constructed using literals and other constants. For details about literal types, see "Literals", in Chapter 1 of the *Developer's Reference*.

You can define the value for a constant whose primitive type is not a specific literal format by using a typecast of a **String** literal or in the case of a **Byte**, a small **Integer** literal, as shown in the examples in the following table.

Primitive Type	Value Expression
Date	"31/12/2007".Date
Time	"14:34:23.123".Time
TimeStamp	"31/12/2007, 14:34:23:123".TimeStamp
Point	"1,7".Point
Byte	0.Byte

For details about typecasting, see "Type Casts", in Chapter 1 of the *Developer's Reference*.

Sorting Constants by Value

Use the **Sorted By Value** command from the Constants menu or the Categories menu in the Class, Primitive Types, or Interface Browser to sort all constants in the current class, primitive type, interface, or global constants category in ASCII collating sequence, based on the value of the constants.

When you select this command, a check mark is displayed to the left of the command in the Constants or Categories menu. (To change the option back again and display the constants in alphabetical order of name, repeat the operation. The check mark is then no longer displayed at the left of the command in the Constants or Categories menu.)

The constants are first sorted into type and they are then sorted appropriately. Numeric types are sorted numerically. The sort order is as follows.

1. Character (ASCII sequence)
2. String (ASCII)
3. Boolean (ASCII)
4. Integer (numeric)
5. Decimal or Real (numeric value)
6. Date (date order)
7. Time (time order)
8. TimeStamp (date and time order)
9. Point (numeric order, *x* and then *y* values)

10. Binary (ASCII)
11. MemoryAddress (ASCII)

Promoting Class Constants to Global Constants

Use the **Promote** command from the Constants menu in the Class Browser to promote a class constant to a global constant.

To promote a class constant to a global constant

1. In the Constants list of the Class Browser, select the class constant that you want to promote.
2. Select the **Promote** command from the Constants menu. (The **Promote** command is disabled if no class constant is selected.)

   The Promote Constant dialog is then displayed, showing the name of the constant you selected for promotion.
3. In the drop-down list of the **Select Category** combo box, select the global constant category to which the selected constant is to be promoted. All global constant categories in the current schema and its superschemas are displayed for selection.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

The class constant is then promoted to your selected category.

Notes As global constant names must be unique to the schema in which they are defined and cannot have the same name as a class constant, the constant is *moved* from the class in which it was defined to the global constant category.

You can promote only class constants. (You cannot demote a global constant to a class.)

Printing Class, Primitive Type, or Interface Constants

To print class, primitive type, or interface constants

1. Select the **Print Selected** command in the File menu from the Class Browser, Primitive Types Browser, or Interface Browser.

   The Print Class or Print Options is then displayed. The **Constants** check box in the Options group box is checked (that is, selected) by default.
2. If you do not want your class, primitive type, or interface constants output to the printer, select the **Print Preview** option button to preview the output on your monitor, or the **RTF File** option button to output your selections to a Rich Text Format (RTF) file in your working (**bin**) directory.
3. Click the **Print Setup** button if you want to change the setup of your printing. (Print setup details are retained for subsequent print requests.)
4. Click the **OK** button to confirm your selection. Alternatively, click the **Cancel** button to abandon this selection.

Each constant in the selected class, primitive type, or interface is then output to the medium that you select (that is, preview, printer, or RTF file).

Using Class, Primitive Type, or Interface Constants in Your Methods

If you were to define a **ColorCerulean** constant with an integer definition of **98765** for a **Form** class, you can refer to that constant in a number of ways in a Jade method.

- In methods of the **Form** class or its subclasses, refer to the **ColorCerulean** constant as follows.

```
setForeColor(ColorCerulean);
```

- In instances of type **Form**, refer to the **ColorCerulean** constant in methods as follows.

```
backColor := f.ColorCerulean;
```

- In any method, refer to the **ColorCerulean** constant as follows.

```
color := Form.ColorCerulean;
```

Note Remember to enclose a string constant definition in single quotes (**''**) or double quote (**""**) characters; for example, **"This is an example"**.

Defining Global Constants

Global constants provide a more-meaningful representation than simply using literal values or numbers.

You can define constants of any primitive type; for example, **String**, **Integer**, or **Character**. A number sign (#) precedes a hexadecimal (hex) value. Hexadecimal strings in Jade are represented as a sequence of space-separated hexadecimal characters; for example:

```
WhiteSpace = #"09 0A 0B 0C 0D 20";
```

Constants in expressions can be user-defined constants or they can be one of the predefined system constants. As the name implies, global constants cannot be updated or modified in any way within the body of a method.

Jade provides system global constants in the **RootSchema** that are available for all of your schemas. For details, see "Global Constants Reference", in Appendix A of the *Encyclopaedia of Primitive Types*.

Note Only one Global Constants Browser for the current schema can be open at any time. If a Global Constants Browser is already open for that schema, it is brought to the top when you select the **Global Constants** command from the Browse menu.

You can have concurrent open Global Constants Browsers for different schemas in the current development environment session.

Accessing the Global Constants Browser

The Browse menu **Global Constants** command from the Schema Browser accesses the Global Constants Browser window, to enable you to define and maintain global constants in your schemas.

» To open the Global Constants Browser, perform one of the following actions

- Select the **Global Constants** command from the Browse menu
- Press Ctrl+G

The Global Constants Browser window is then displayed. The global constant categories defined for the current schema are displayed in the Global Constants list when the Global Constants Browser is opened, and nodes are collapsed. To view global constants and categories defined in superschemas, see "Customizing the Class Browser of the Current Schema", in Chapter 3.

You can change your default browser options, if required, by using the Browser Options from the Options menu **Preferences** command. (For details, see "Setting User Preferences", in Chapter 2.)

To maintain a user-defined:

- Global constant, select the global constant that you want to modify or delete, and then select the appropriate command from the Constants menu.
- Global constant category, select the category that you want to modify or delete, and then select the appropriate command from the Categories menu.

To add a global constants category to your schema, see "Adding a Global Constant Category", elsewhere in this section.

Note You cannot modify system-defined categories or global constants, which are displayed in red by default.

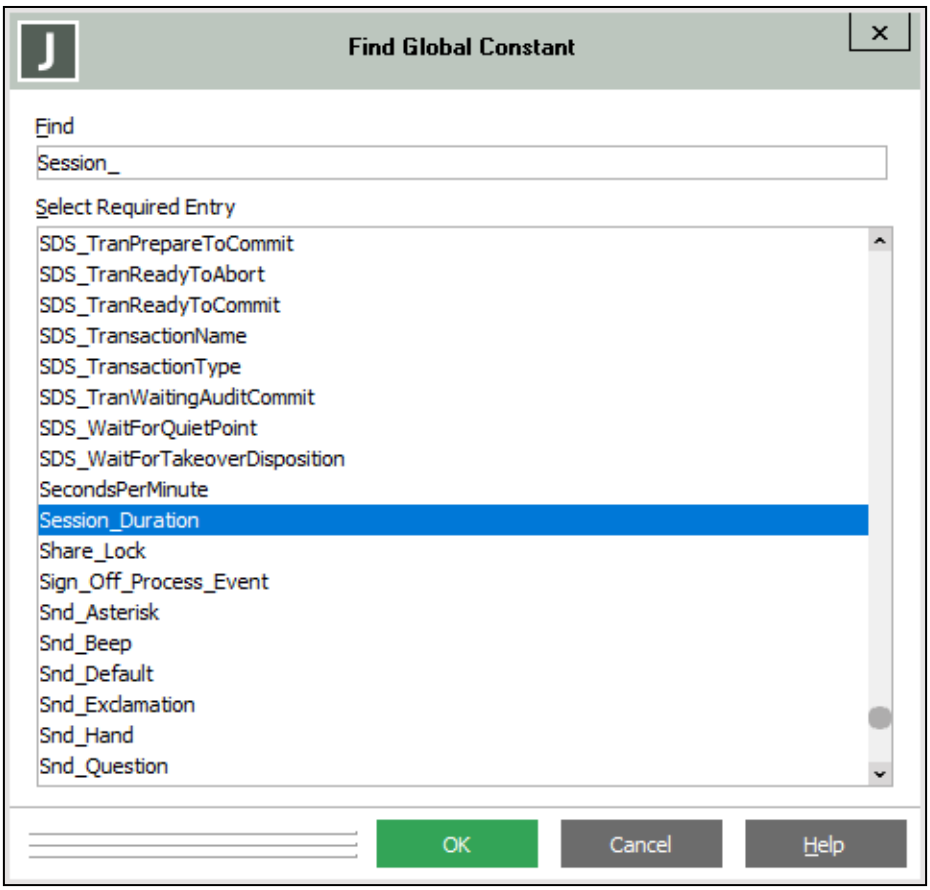
Finding a Global Constant

The Find Global Constant dialog enables you to search for a global constant in all categories in your current schema and its superschemas (including the **RootSchema**). Use this dialog to determine if a global constant exists in your schema or any of its superschemas that you could use instead of defining a new one.

To find a global constant

1. In the Global Constant Browser, select the **Find** command from the Constants menu or press F4.

   The Find Global Constant dialog, shown in the following image, is then displayed.



   All global constants in the current schema and all its superschemas are displayed in the **Select Required Entry** list box.

2. To select the global constant that you require, perform one of the following actions.

   - Select the appropriate global constant from the list box.
   - Type at least the first few characters of the global constant name in the **Find** text box. (The list display starts with the global constant matching the value that you specify.)

3. When you have selected the required global constant, click the **OK** button.

A maximum of 32,000 entries is loaded into the list box. If the specified text does not match any of the entries in the list and there are more than 32,000 entries, the rest of the entries are searched for the specified text. If a global constant is found, the list is cleared and repopulated with this entry and entries that follow this entry (again up to the 32, 000 limit).

The selected global constant, the category in which it is contained, and any other global constants in that category are then displayed in your Global Constants Browser.

By default, global constants and categories inherited from superschemas are displayed in blue and the user-defined global constants and categories in the current schema are displayed on black. You can change these display colors, by using the **Window** sheet from the Preferences dialog, accessed from the Options menu **Preferences** command. (For details, see "Maintaining Window Options", in Chapter 2.)

Adding a Global Constant Category

The **Add** command from the Categories menu in the Global Constants Browser enables you to add a new global constant category to the current schema.

The global constant categories enable you to group your global constants into meaningful groups. Global constants provide a more-meaningful representation than simply using literal values.

To add a global constant category

1. In the Global Constants Browser, select the **Add** command from the Categories menu. The Define Constant Category dialog is then displayed.
2. In the **Name** text box, specify the name of the category that you want to define. The name can contain any letter or number but cannot contain punctuation characters.
3. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The specified category is then displayed in the Global Constants List of the Global Constants Browser, to enable you to add global constants to the new category.

The actions that you can perform using the mouse from the Global Constants List window are listed in the following table.

Action	Result
Click	The global constants provided by the selected category populate the Global Constants List window. The declarations defined for the selected global constants are displayed in the global constants definition pane. A selected collapsed node is expanded when a node displaying a plus sign (+) is clicked, or an expanded node is collapsed when a node displaying a minus sign (-) is clicked.
Right-click	Displays the Constants menu, to enable you to maintain the selected global constant.
Double-click	Expands a collapsed node for the selected global constant, if applicable.

Use the scroll bar to scroll up and down the Global Constants List window, if required.

Adding a Global Constant

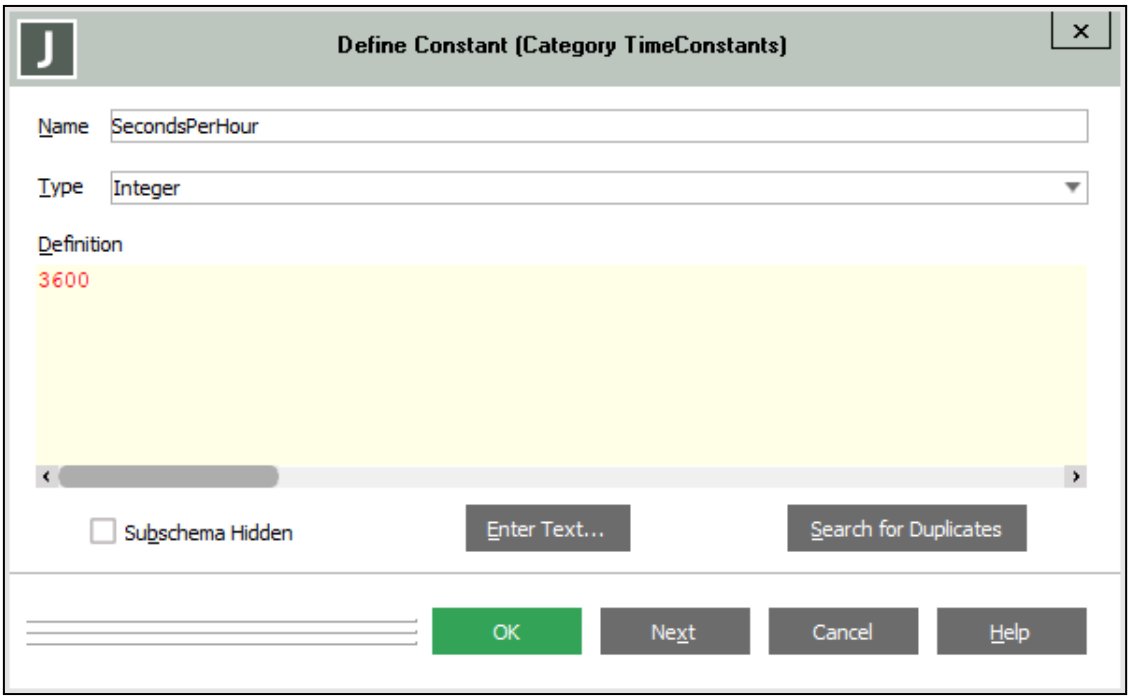
You can add a global constant to an existing category or to a new global constant category.

Note You cannot define a global constant that conflicts with a global constant defined in a superschema.

To add a global constant

1. In the Global Constant Browser, select the category to which the global constant is to be added.
2. Select the **Add** command from the Constants menu.

The Define Constant dialog, shown in the following image, is then displayed.



3. In the **Name** text box, specify the name of the global constant that you want to define.

   The constant name starts with an initial capital letter (for example, **TestConst**) and it must be unique to the schema to which it is being added.

4. In the **Type** combo box, select the required primitive type in the Type list, or specify the first character or the first few characters of the primitive type in the text box and then select the required primitive type from the Type list. The selected primitive type is then displayed in the **Type** text box.

   The type can be any primitive type to which you want to assign a constant value; for example, an integer value of **1** or a string value of **"cerise"**.

5. In the **Definition** editor pane, specify the definition of your global constant. The definition can be a simple literal value or a more-complex expression.

   Note For tips about defining global constant values, see "Constant Definition Tips", earlier in this chapter.

6. Check the **Subschema Hidden** check box if you want to specify that the global constant is available only in the local schema; that is, it is not available for use in any subschemas.

   For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

7. Click the **Enter Text** button if you want to specify or maintain descriptive text for the global constant as part of the definition or maintenance of the constant. For details, see "Specifying Text for a Schema Element", in Chapter 3.

   Tip You can also specify descriptive text for the global constant at any time, by selecting the **Text** command from the Constants menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

8. Click the **Search for Duplicates** button if you want to determine if a global constant with an identical definition value exists in the current schema or any superschema.

   If an existing definition value is located, a message box displays the name of the global constant containing that definition, category, and schema in which it is defined.

9. Click the **OK** button.

   Alternatively, click the **Cancel** button to abandon your selections or the **Next** button to define another global constant.

The specified global constant is then displayed in the Global Constants List of the Global Constants Browser, with the value that you defined for the global constant displayed in the definition pane.

Printing Global Constants

» To print global constants from the Global Constants Browser

1. Select the **Print Selected** command in the File menu. The Print Global Constants dialog is then displayed.

   As print options only apply to the printing of global constants, only the Print Options group box is enabled.

2. If you do not want your global constants output to the printer, select the **Print Preview** option button to preview the output on your monitor or the **RTF File** option button to output your selections to a Rich Text Format (.**rtf**) file in your working (**bin**) directory.

3. Click the **Print Setup** button if you want to change the setup of your printing. (Print setup details are retained for subsequent print requests.)

4. Click the **OK** button to confirm your selection. Alternatively, click the **Cancel** button to abandon this selection.

Each category and the global constants in those categories are then output to the medium that you select (that is, preview, printer, or rich text format file).

Chapter 5 Using the Painter and Form Wizard

This chapter covers the following topics.

- Using the Jade Painter
  - Accessing the Painter
  - Creating a Form
  - Testing Your Form
  - Selecting a Skin for Painted Forms
  - Saving Your Form
  - Editing an Existing Form
  - Painter Menus
  - Painter Toolbars
- Using Form Wizard to Create a Form
  - What Form Wizard Produces
  - Accessing the Form Wizard
  - Specifying Your Form Name and Title
  - Specifying the Target Class
  - Specifying the Visible Instances
  - Selecting the Presentation Style for the Instances
  - Selecting the Functionality of Your Form
  - Selecting Browse Functionality for Your Form
  - Selecting Find Functionality for Your Form
  - Selecting the Features Displayed on Your Form
  - Removing Features Selected for Display
  - Changing the Sequence of Displayed Features
  - Selecting Captions and Control Names
  - Incorporating a Toolbar, Menu, and Status Line
  - Initiating the Form Build
- Copying a Form into another Locale

Using the Jade Painter

The Jade Painter enables you to create and maintain screen, printer, and Internet forms for runtime Jade applications. The Painter is a graphical user interface (GUI) painter that supports bitmaps, list boxes, button controls, and so on. A *form* is a window that acts as a container for controls that display information and that permit user input. Forms have properties that determine aspects of their appearance (for example, position, size, or color) and aspects of their behavior (for example, whether they can be resized).

Notes Only one Painter window for the current work session can be open at any time. If a Painter window is already open for that work session, it is brought to the top when you select the **Painter** toolbar button or the File menu **Painter** command from the Jade Platform development environment. You can have concurrent open forms for different schemas in the Painter window, if required.

If you have mapping logic on subclassed controls, other processes such as the Jade Painter, Translator utility, or the loading of schemas may also execute that logic. The logic therefore may need to perform checks to determine if it is running in the user application environment, to ensure that exceptions are not generated in these other situations.

You can access the Jade Painter from a user-defined schema only; that is, you cannot access it from the **RootSchema**.

For details about the Jade Painter, see the following subsections.

Accessing the Painter

To access the Jade Painter, perform one of the following actions

- Select the **Form Painter** option button from the Jade sign-on dialog
- Select the **Painter** command from the File menu of a browser window (or press Ctrl+P)
- Click the **Painter** toolbar button in the Jade Platform development environment
- Select the **Form Wizard** command from the File menu of a browser window

The Jade Painter, shown in the following image, is then displayed.

The title bar displays the database path, user identifier and process identifier, and the skin (if selected), in the following format.

```
Jade Painter : (database-path : user-identifier_painter-process-identifier :
singleUser or multiUser) schema-name::form-name skin-selected
```

The **Control** and **Alignment and Size** palettes, or toolbars, are always hidden when you start a Painter session and they are displayed when you open a form for edit or you start to paint a new form. When all open forms are closed, the display of these palettes is then hidden.

The status line is divided into three parts. The first part displays the status, which is most often **Ready**. The second part displays the three-letter abbreviated country code for the locale of the form being edited (for example, **USA** or **NZL**). The third part displays the form name, size, and position or the control name, size, and position if a control is selected.

Creating a Form

This subsection covers the following topics.

- Specifying Your Form Preferences
- Adding a New Form
  - Printer-style Forms
- Adding Controls to Your Form
- Displaying a Hierarchical List of Controls on a Form

- Adding Container Controls
- Allowing Control Docking
- Initializing the JadeRichText Control
- Defining the Layout of Your Form
- What Happens Next
- Maintaining Properties for Your Form or Control
- Designing Menus
- Defining Methods for Your Form

Specifying Your Form Preferences

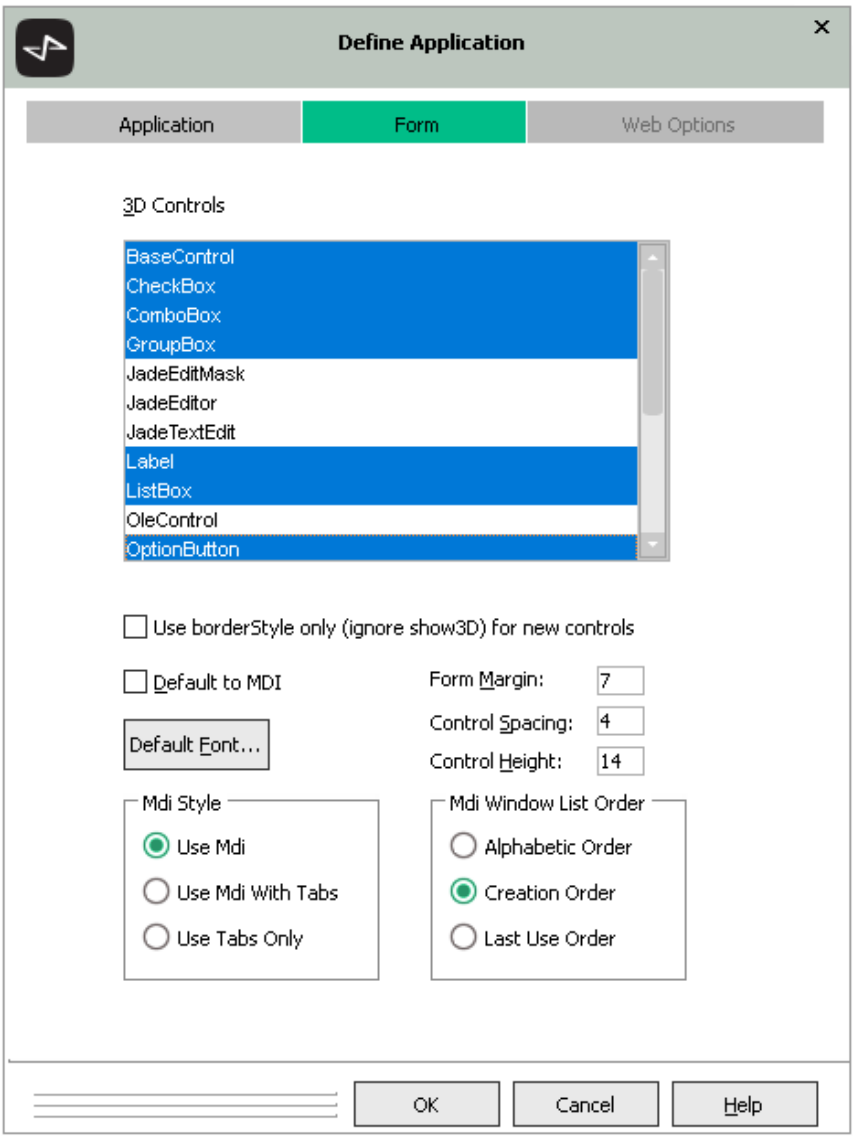
You can specify your form preferences when you first define your application or you can access the Define Application at any time to maintain your application preferences.

To access the Define Application dialog

1. Perform one of the following actions to open an Application Browser window.
   - Click the **Browse Applications** button from the Browser toolbar
   - Select the **Applications** command from the Browse menu
   - Press Ctrl+L
2. From the Application menu of the Application Browser, select either:
   - The **Add** command, to add a new application to your current schema
   - The **Change** command, to maintain the preferences of your current application

   The Define Application dialog is then displayed.
3. Click the **Form** tab.

The **Form** sheet of the Define Application dialog, shown in the following image, is then displayed.



The **Form** sheet of the Define Application dialog enables you to specify the attributes for your forms. The form margin, control spacing, and control height preferences are specified in Dialog Units, so that you do not need to specify both vertical and horizontal spacing. (Pixel spacing is then a function of the Dialog Units and the application font.)

To specify your form preferences from the Form sheet

1. In the **3D Controls** list box, select the controls to which three-dimensional effects are added.
2. Check the **Use borderStyle only (ignore show3D) for new controls** check box if you want to ignore the default value of the **show3D** property for new controls and use only the value of the **borderStyle** property.
3. Check the **Default to MDI** check box if you want your forms created as Multiple Document Interface (MDI) forms. By default, forms are not created as MDI forms; that is, this check box is unchecked.

   Note This is an application default, which you can override for individual forms.

4. Click the **Default Font** button if you want to change the default font for the forms in your application. The default font for your forms is Tahoma, regular style, and 8.25 points. (This applies only if you have not set a specific application font.)

   The common Font dialog is then displayed, to enable you to make your font selections. When you have made your font selections, focus is then returned to the **Form** sheet of the Application dialog.

5. In the **Form Margin** text box, specify the number of dialog units that you require as a margin around the edge of forms (windows) in your Jade application.

   Form margins enable you to position controls so that they do not intrude into the margin of the form. The default value is **7** dialog units.

   When the **Show Alignment Hairs** command from the Painter Options menu is set, a margin of the specified number of dialog units is drawn around the inside of the form. No alignment grid lines are drawn in the margin of a form.

6. In the **Control Spacing** text box, specify the number of dialog units that you require between controls on forms in your Jade application.

   The default value is **4** dialog units.

7. In the **Control Height** text box, specify the number of dialog units that you require as the uniform height of button, label, and text box controls on forms in your Jade application.

   The default value is **14** dialog units.

8. If you do not want standard MDI child forms (the default), in the Mdi Style group box, select the:

   - **Use MdiWidth Tabs** option button if you want MDI child forms in the application displayed in the MDI frame with a tab for each child form. The child forms can be maximized, minimized, and restored.
   - **Use Tabs Only** option button if you want a tab only for each MDI child form on the MDI frame. Only the top MDI child form is visible and it is always maximized.

   For details, see the **Application** class **mdiStyle** property in Volume 1 of the *Encyclopaedia of Classes*.

9. If you do not want MDI child forms shown in creation order (the default) in the Window menu list of child forms and in the MDI Tabs down arrow list, in the Mdi Window List Order group box, select the:

   - **Alphabetic Order** option button if you want the Window list menu and the MDI tabs down arrow to show the list of MDI child forms in alphabetic order of the caption.
   - **Last Use Order** option button if you want the Window list menu and the MDI tabs down arrow to show the list of MDI child forms in the order in which they were last used.

   For details, see the **Application** class **mdiWindowListOrder** property, in Volume 1 of the *Encyclopaedia of Classes*.

10. Click the **OK** button.

    Alternatively, click the **Cancel** button to abandon your selections.

Adding a New Form

You can create forms only in the Jade Painter. A form can be a subclass only of another user-defined form class or of the **Form** system class. Existing forms are displayed in the **Existing Forms** list box, to enable you to select the form that is to be subclassed.

A subform inherits all of the methods and properties of its parent, or superform. All forms are subclasses of the **Window** class.

Use a form to create an interface for your Jade applications. A form is a window on which you paint controls for the running of your application. Controls are objects such as text boxes, list boxes, buttons, check boxes, and so on. You also use a form to define a print layout.

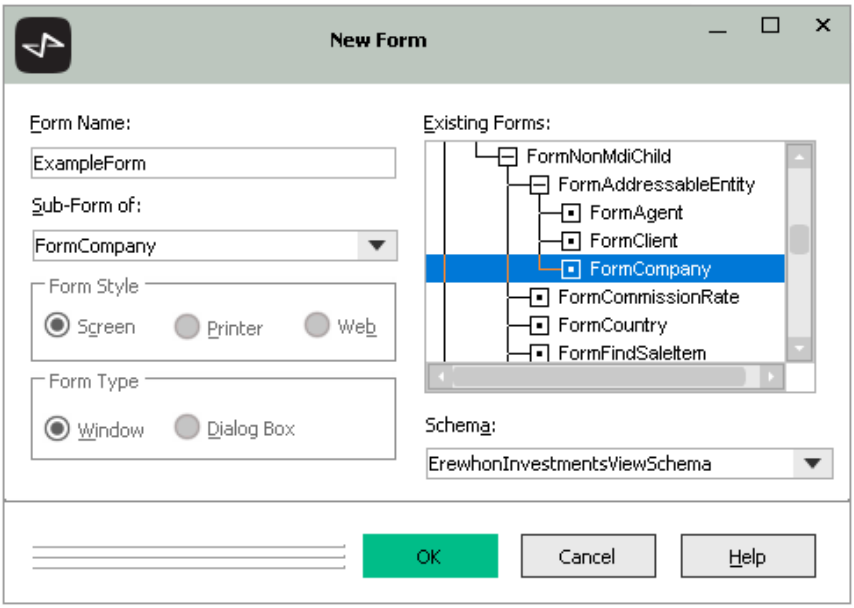
When you add the first form to your schema, it is automatically created as a *subclass* of the **Form** system class (*superclass*), inherited from the **RootSchema**.

The first form that you add to a schema becomes the default start-up form (the form that is initially created and displayed automatically when the application is started up).

To create a new form, perform one of the following actions

- Select the **New Form** command from the File menu.
- Click the **New Form** toolbar button.

The New Form dialog, shown in the following image, is then displayed, to enable you to define your new form.



To define your new form

1. If you do not want to define a form for the Painter's current schema, select the schema to which you want to add the form in the **Schema** combo box at the lower right of the dialog. The **Existing Forms** list box then displays the forms defined for that schema. (For details about editing existing forms, see "Editing an Existing Form", later in this chapter.)
2. Specify a form name in the **Form Name** text box. You must specify a form name, which cannot have the same name as:
   - Another form
   - An application name within the current schema
   - Another class in the current schema (or any of its superschemas)
   - Predefined Jade classes or interfaces

The form name must start with a letter. It can include numbers and underscore characters, but cannot include punctuation or spaces. As forms are defined in the Jade database as classes, the first letter of the name is converted to an uppercase character, if it is lowercase.

3. If your form is a subform, perform one of the following actions.
   - Specify its superform in the **Sub-Form of** combo box.
   - Select its superform from the **Sub-Form of** combo box.
   - Select its superform from the **Existing Forms** list box.

   A subform inherits all the methods and properties of its parent or superform.

4. In the Form Style group box, select the **Printer** or the **Web Page** option button if you do not want to define a screen form (the default). You can define your form as a screen, printer, or web page form. The controls on a printer form do not accept input at run time.

   If you select the **Screen** default option, use the Form Type group box to define the type of screen entity.

   For details about creating a form as a Printer form, see "Printer-style Forms", in the following subsection. For details about creating a web page form, see "Creating Web Pages", in Chapter 1 of the *Web Application Guide*.

5. In the Form Type group box, select the **Dialog Box** option button if you do not want to define a window form (the default). The **Dialog** option button automatically provides you with control buttons for **OK**, **Cancel**, and **Help** functions.

6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Printer-style Forms

When you create a Jade form as a printer-style form (that is, you select the **Printer** option button in the Form Style group box on the New Form dialog in the Jade Painter), you are declaring that the form will be used for printing. It will affect the way the form and controls are displayed in the Jade Painter, some default property values when the form and controls are created, and it will alter the appearance of the form and some controls when used as a web form.

Note The form can still be created and displayed as a normal GUI form.

When a form is created as a printer-style form, it has the following effects.

- In the Jade Painter, if the value of the **borderStyle** property is **BorderStyle_None** (**0**), the following controls are drawn with a light border to show the extent of the control.
  - Label
  - Frame
  - JadeRichText
  - TextBox
- Default properties are changed when the form is created or when a new control is added to the form, as follows.
  - Form
    - backColor = White
    - scaleForm = true
    - mdiChild = MdiChild_NotMdi (1)

- Control
  - show3D = Show3D_Not3D (1)
  - foreColor = Black
  - fontName = Tahoma
  - fontSize = 10
- Frame
  - backColor = White
  - borderStyle = BorderStyle_None (0)
  - bevelInner = 0
  - bevelOuter = 0
  - boundaryWidth = 0
  - caption = ""

- If the form is displayed or printed as a Jade web form, the HTML generated for the form does not include a border, a style, or a caption.
- If the form is displayed or printed as a Jade web form, the HTML generated for a **Frame**, **JadeDockContainer**, or **JadeDockBar** control allows overflow if the contents do not fit inside the container.

You can use the **Form** class **isPrinterForm** method at run time to test whether form was created as a printer-style form.

To set default properties on controls when a control is added to a form in the Jade Painter, re-implement the **Window** class **setDefaultPainterControlProperties** method in a user schema. For example, to change the default font for printer form controls, re-implement the **setDefaultPainterControlProperties** method on the **Control** class, as follows.

```
setDefaultPainterControlProperties();
begin
    if self.form.isPrinterForm() then
        self.fontName := "Arial";
    endif;
end;
```

Applies to Version: 2018.0.02 (Service Pack 1) and higher

Adding Controls to Your Form

Use the **Add Control** command from the Controls menu to add a new control to your current form.

Notes You can also add a control by using palette buttons. A description of each palette button is displayed in bubble help when you move the mouse over each palette button.

It not possible to handle the painting of transparent controls in the correct **zOrder** when it involves a mixture of controls that can be directly painted by Jade and those that can only be painted separately by Windows. As a result, transparent sibling controls are always be painted before any **JadeRichText**, **MultiMedia**, **JadeXamlControl**, **Ocx**, **OleControl**, **JadeDotNetVisualComponent**, or **ActiveXControl** controls, regardless of their **zOrder** settings.

For details about editing a form, see "Editing an Existing Form", and for details about designing a menu, see "Designing Menus", later in this chapter. See also "Allowing Control Docking" and "Suspend Parent Alignments Command", later in this chapter.

To add a new control to your form

1. Select the **Add Control** command from the Controls menu.

   The **Add Control** dialog is then displayed.

   Tip Double-click a button in the Painter **Control** palette to add multiple instances of the selected control. For example, if you double-click the **CheckBox** palette button, you can then click the cursor in your form the number of times required to add new check boxes in the required positions.

   You must click the **Pointer** button ( ) at the far left of the **Control** palette to terminate the add control mode, return to painter mode, and continue painting your form.

2. Select the type of control that you want to add from the list of controls in the **Add Control Type** list box.
3. Click the **Add** button. When you now move the cursor over unfilled parts of your form, the cursor becomes a cross.
4. To position your control, move the cursor, and then click to gain the default control size or drag it out to the required size.
5. To size the control, drag the sizing handles (the eight small boxes that surround the border of a selected control).
6. To finely position the control, click inside the control, and then drag it to the required location on your form.
7. Apply properties to your control by using the **Show Properties Dialog** command in the Window menu, by pressing F4, or by double-clicking the form. For details, see "Maintaining Properties for Your Form or Control", later in this chapter.

All controls are children of the form or container control on which they are drawn. Being a child control affects the placement of the control within the parent form or container.

The left and top properties of a control are relative to the parent control. Moving the form or container also moves the controls.

You can display a hierarchical list of all controls painted on the currently active form. For details, see "Displaying a Hierarchical List of Controls on a Form", later in this chapter.

Tips Use the **Grid** command or **Show Alignment Hairs** command from the Options menu to assist you in positioning your controls on the form, if required. (For details, see "Using Grids to Position Controls on Forms" and "Using Alignment Hairs to Position Controls on Forms", later in this chapter.)

To access the online help topic for the current control, click the right mouse button and select the **Help on control** command from the popup menu that is then displayed. Online help that relates directly to the control that has focus on a form in the Jade Painter is then displayed. For example, if a **TextBox** control is selected, the "TextBox Class" Jade online help topic is displayed.

Alternatively, if the selected control is an ActiveX control that was imported into Jade, the help file associated with that ActiveX control is then opened. (Online help for an ActiveX control is a separate help file provided by the organization that wrote the control, it relates only to that control, and it is not part of the Jade online help.)

For details about changing the type of an existing control on a form (for example, changing a **GroupBox** control that does not have children to a **Frame** control), see "Changing a Control Type" under "Editing an Existing Form", later in this chapter.

Adding Container Controls

Container controls represent containers for other controls. The container controls are as follows.

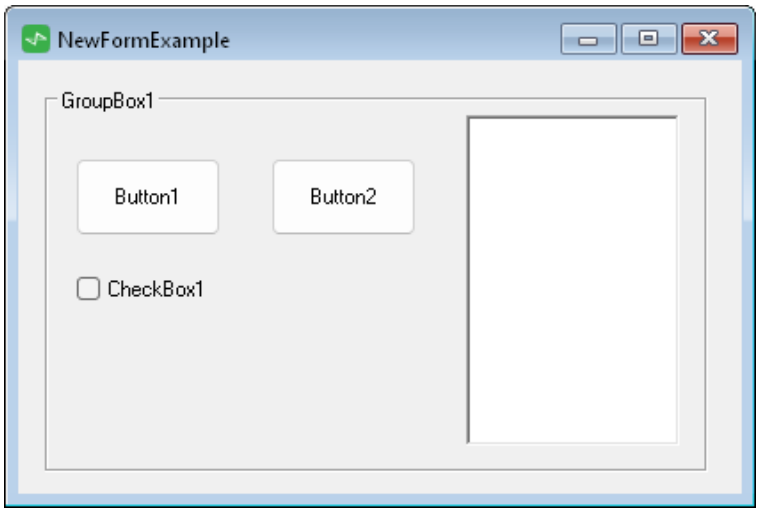
- GroupBox
- Folder
- Frame
- JadeDockBar
- JadeDockContainer
- Other user-defined controls (for details about creating your own controls, see Chapter 5 of the *Developer's Reference*)

A container control must be added before the controls that it is to contain. When a container control is in place, other controls can be added within its bounds.

Note You cannot use the Ctrl+drag keyboard action to move or copy selected controls that have different parents. You can, however, copy and paste controls with different parents.

For details about docking of **Frame** and **JadeDockContainer** controls, see "Allowing Control Docking", later in this chapter.

The following image shows an example of **Button**, **CheckBox**, and **ListBox** controls within a **GroupBox** control.



To move an existing control in a container control, use the mouse to drag the control.

Pushing a Control to the Bottom

When two or more controls overlap, you can select which control appears on top or behind the other control or controls.

The **Push to Bottom** command from the Controls menu pushes a selected control behind the other overlapping controls.

Pushing all Parents and the Control to the Bottom

When two or more controls overlap and a control is selected, you can push the control and all of its parents to the bottom of the displayed zOrder.

The **Push all Parents and Control to Bottom** command from the Controls menu pushes the selected control and all of its parents behind the other overlapping controls.

Applies to Version: 2020.0.01 and higher

Bringing a Control to the Top

When two or more controls overlap, you can select which control appears on top or behind the other control or controls. The **Bring to Top** command from the Controls menu causes the selected control to sit on top of any overlapping controls.

Bringing all Parents and the Control to the Top

When two or more controls overlap and a control is selected, you can bring the control and all of its parents to the top of the displayed zOrder.

The **Bring all Parents and Control to Top** command from the Controls menu causes the selected control and all of its parents to sit on top of any overlapping controls.

Applies to Version: 2020.0.01 and higher

Selecting all Controls on Your Form

Use the **Select All** command from the Edit menu to select all of the controls on your painted form; for example, to reposition all your controls at once. When you have selected all controls, if you position the caret in any of the controls, hold down the left mouse button, and then drag the control, all the controls move in formation.

Locating a Control on Your Form

The **Find** command from the Edit menu finds a specified control and enables you to optionally bring the specified control to the top of any overlapping sequence of controls when it is found.

Obtaining Help for a Control Type

Use the **Help on control** command from the popup menu to access the page of the PDF file that provides information about the control type that has focus; for example, right-click on a combo box and then select the **Help on control** command from the popup menu that is displayed. Online help that relates directly to the control that has focus on a form in the Jade Painter is then displayed. For example, if a **TextBox** control is selected, the "TextBox Class" Jade online help topic is displayed.

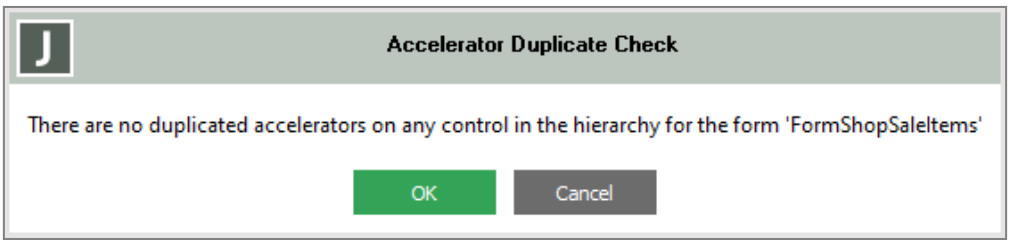
Checking for Duplicate Control Accelerators

» **To check for duplicate control accelerators in the full hierarchy of the currently selected form**

- Select the **Check For Duplicate Control Accelerators** command from the Controls menu.

The accelerators on all controls in the hierarchy of the currently selected form are then examined. This includes all superclass forms and all subclass forms of the currently selected form.

All of the accelerators on all forms in the hierarchy of the form are searched, and a message box is displayed, as shown in the example in the following image.



After selecting the **Check For Duplicate Control Accelerators** command in the Controls menu, the Accelerator Duplicate Check message box displays one of the following.

- There are no duplicated accelerators on any control in the hierarchy for the form *selected-form-name* [including sub-classes]
- There is a duplicate use of a control accelerator as follows: '*accelerator-character*' [*form-name*::]*control-name* [*form-name*::]*control-name*
- There are duplicate uses of control accelerators as follows: *'accelerator-character'* [*form-name*::]*control-name* [*form-name*::]*control-name accelerator-list*

The menu command is disabled if a form is selected or if the selected control does not have a caption property.

Applies to Version: 2020.0.01 and higher.

Finding the Available Control Caption Accelerator Characters

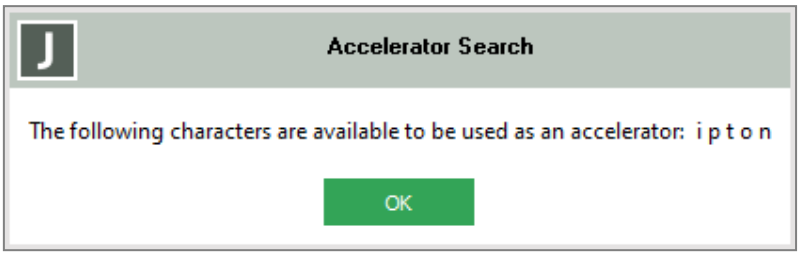
To find the available caption accelerators for a control

1. Select the control whose available caption accelerators you want to view.

   This process includes all superclasses and all subclasses of the current form.

2. Select the **Find Available Caption Accelerator Characters** command from the Controls menu.

All of the used accelerator characters for other controls on the form, all superclasses, and all subclasses of the current form are gathered and removed from the caption string of the currently selected control, leaving the list of available characters in the control caption that are available to be used as a unique accelerator, as shown in the following image.



After selecting a control and clicking the **Find Available Caption Accelerator Characters** command, the Accelerator Search message box displays one of the following.

- All alpha/numeric characters in the caption have already been used as accelerators on other controls on this form [and sub-class forms]

- The character *character* is available to be used as an accelerator
- The following characters are available to be used as an accelerator: *character-list*

The menu command is disabled if a control is not selected or if the selected control does not have a **caption** property.

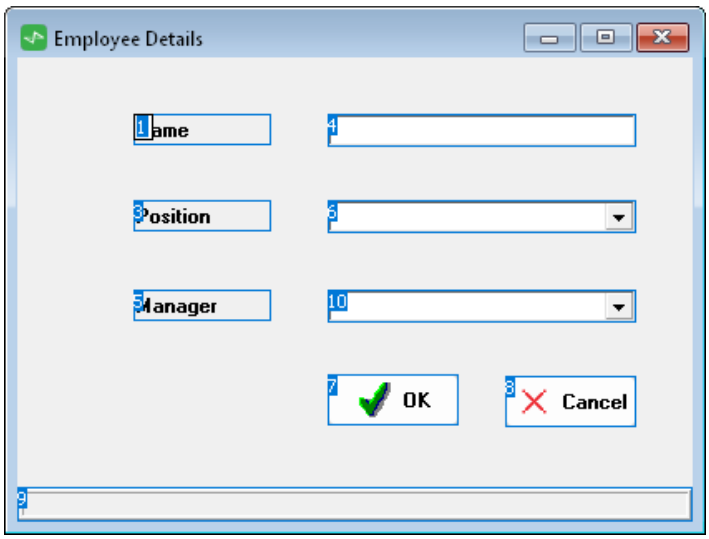
Applies to Version: 2020.0.01 and higher

Changing the Runtime Tab Sequence

To change the tab sequence that the user experiences at run time

- Select the **Tab Ordering** command from the Controls menu.
- Click **Tab ordering** button on the Tools toolbar.

The tab indexes are then displayed in the top left of each control of your form, as shown in the following example.



By default, Jade assigns a tab order to controls as you define them on your form.

Each new control is placed last in the tab order. The valid range is any integer value.

Note Controls are pasted on the form in ***controlList*** order (that is, the order in which you add them to the form). The order in this list can change during editing, because parent controls must always precede their children. Controls are copied to the Painter clipboard in **controlList** order. When pasted, each control is added to the form and a new **tabIndex** property is assigned. If the tab index of a control has not been used in the new form (the form on which you paste the control), the pasted control retains its tab index value. A control pasted on to a form can therefore have a tab index order that differs from the one that it had on the original form, although in most cases, the tab order corresponds to that of the original form (particularly when pasting controls on to new forms).

To change tab indexes

1. Click the tab index that you want to change.
2. Enter the new tab index value (in the range 1 through 9999).
3. Press the Enter key or the TAB key.

4. Repeat steps 1 through 3 for each tab index that you want to change.
5. To return to normal Painter mode, select the **Tab Ordering** command from the Controls menu once more. The tab indexes are then no longer displayed on your form.

When you change the tab order by using the **Tab Ordering** command, you change the value of the **tabIndex** property of the control. If the **tabIndex** already exists for an existing control, the tab index of that control (and all other controls also affected) is increased by one. You can also change the **tabIndex** property by using the Properties window, or at run time in logic.

Jade automatically renumbers the **tabIndex** value of other controls to reflect insertions, deletions, and changes that would result in a tab number conflict.

All controls except menus are included in the tab order. At run time, invisible or disabled controls and controls that cannot receive the focus (that is, frames and labels) remain in the tab order but are skipped during tabbing.

Note You can neither change the tab order of controls that belong to a superform nor use the tab number of a subform.

When changing the tab number to a number that already exists, Jade automatically increments the tabs of all other controls. Each control must have a unique tab number.

Control Order on Touchscreens

When a device is running with accessibility options enabled, Jade changes the z-order of controls within each parent to match the **tabIndex** property value order when the form is created so that screen-reading software such as JAWS or Microsoft Narrator read the correct text associated with controls. These software utilities assume that the z-order defines the control associations; for example, the label associated with a text box or combo box.

This means, however, that any arrangement of the z-order of the controls within the Jade Painter could be lost unless the **tabIndex** property values match the required arrangement of the controls.

Notes Changing the tab index of controls at run time does not have any impact on the z-order of controls within the parent.

Calling the **Window** class **zOrder** method at run time also overrides the z-order established when the form was created, regardless of whether accessibility is active or not.

When Accessibility is enabled, Jade cannot be sure whether screen-reading software is active. Narrator, for example, does not set a system flag to indicate it is active, and other software can confusingly claim that screen reading is active. This problem has become more visible with the arrival of touchscreen PCs, where accessibility is usually active by default.

As a result, when defining forms where the **zOrder** of controls is established within the Jade Painter, ensure that the **tabIndex** ordering of the affected controls matches the required order, so that the same behavior occurs on PCs regardless of the accessibility state.

Alternatively, add calls to the **Window** class **zOrder** method to establish the z-order requirements at run time.

Adding or Maintaining Parameters for WebJavaApplet Controls

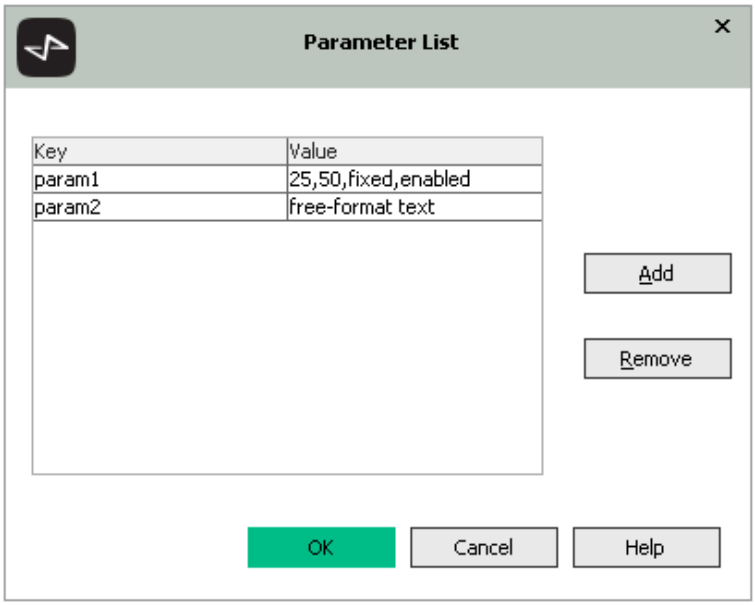
Although you can use the **WebJavaApplet** class **parameters** property to dynamically define one or more parameters to insert in generated HTML for web-enabled sessions, you can define a list of parameters for the **WebJavaApplet** control at design time so that they are inserted in the control when the form is run.

To add or maintain parameters for a WebJavaApplet control

1. Add a **WebJavaApplet** control to the current form or select an existing **WebJavaApplet** on the form.
2. Click the **Specific** icon on the Properties dialog.

   The **Specific** sheet of the Properties dialog is then displayed. (For details, see "Maintaining Properties for Your Form or Control" and "Maintaining Specific Control Properties", later in this chapter.)
3. Select the **parameters** property and then select the **Change** value from the drop-down list in the right column of the **parameters** property row.

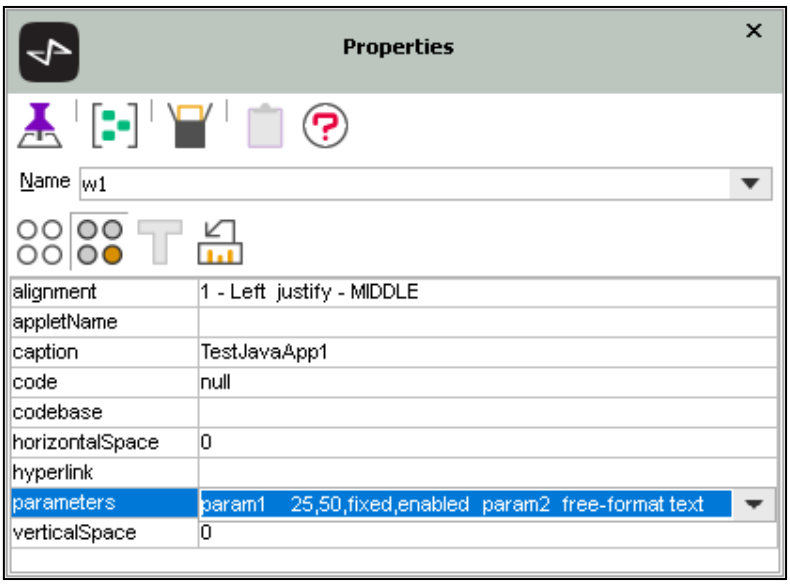
   The Parameter List dialog, shown in the example in the following image, is then displayed.



4. To add a parameter, click the **Add** button and then specify the parameter name in the **Key** column of the row that is then displayed and the value for that parameter key in the **Value** column. Repeat this action until you have added all parameters that you require to the **WebJavaApplet** control. Alternatively, perform one of the following actions.
   - To change an existing parameter, click in the appropriate cell of the table and edit the parameter key or value to meet you requirements.
   - To delete an existing parameter from the control, click on the row that you want to delete and then click the **Remove** button. That row is then removed from the table of parameters.
5. When you have added, changed, or deleted parameters to meet your requirements, click the **OK** button. Alternatively, click the **Cancel** button to abandon you actions.

Any parameters that you added or modified are then displayed in the drop-down list in the right column of the **parameters** property row on the **Specific** sheet of the Properties dialog above the **Change** value.

The parameters in the table in the Parameter List dialog is displayed in one row of the drop-down list in the right column of the **parameters** property row in the Properties dialog, separated by tabs, as shown in the following image.



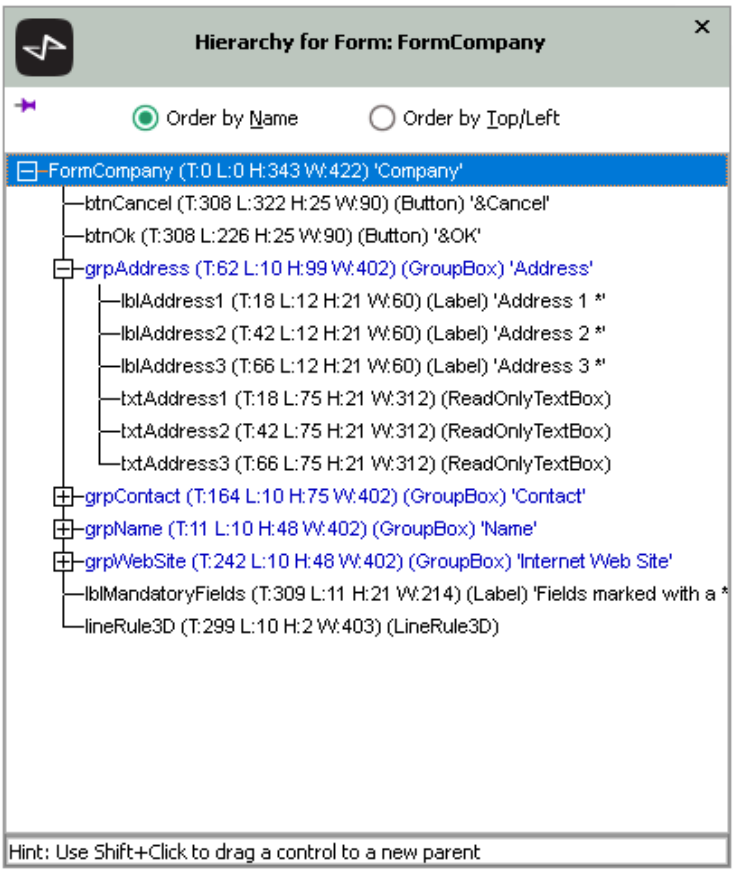
Displaying a Hierarchical List of Controls on a Form

You can display a hierarchical list of all controls painted on the currently active form; for example, if you want to inspect the controls painted on a complex form.

» To access the Hierarchy for Form dialog for the current form, perform one of the following actions

- Select the **Show Control Hierarchy Dialog** command from the Window menu
- Press F5

The Hierarchy for Form dialog, similar to that shown in the following image, is then displayed.



If you do not want the control children ordered by name under their parent (the default value), click the **Order by Top/Left** option button. The control children are then ordered by the top position and then the left position when two controls have the same top position, providing a visual association with the form layout.

The list box displays a hierarchical list of parent-child relationships of each control painted on the form. Each control is listed alphabetically by control name under its control parent entry by default. The description for each displayed form and control includes the top and left positions and the height and width sizes in pixels. Each entry is displayed as *control-name* (T:*top* L:*left* H:*height* W:*width*) (*control-class-name*) '*caption*'; for example:

```
lblWebSite (T:18 L:12 H:21 W:60) (Label) 'URL'
```

Note Click the **Stay on top of Painter** icon at the top left of the dialog or select the **Control Hierarchy on Top** command from the Options menu to keep the Hierarchy for Form dialog on top of (overlay) the Painter. Conversely, repeating these actions toggles the pinning of the dialog on top of the Painter and the check status of the menu command.

To save the current form when the Hierarchy for Form dialog has focus, press F2.

The caption value is displayed only if a **caption** property value has been defined for the control and it is not null.

Each entry that has children is displayed in blue text, while each leaf child entry is shown in black text.

The Windows system font is used to display the entries so that any caption text in other languages can be displayed.

Each time you select a control in the painted form, that entry is selected in the Hierarchy for Form dialog. The hierarchy is expanded, if required, and the entry is centered in the list box, if possible.

The list is updated whenever a control is added to or deleted from the painted form, if the type of a control is changed, the parent of a control is changed, a caption is changed, or if another form being painted is activated.

Clicking on a control entry in the list box on the Hierarchy for Form dialog selects that entry in the Painter and the Painter is activated if it is not already active. In addition:

- Pressing F4 when the Hierarchy for Form dialog has focus causes the Properties dialog to be displayed.
- Double-clicking a control name in the list box of the Hierarchy for Form dialog causes the Properties dialog to be displayed after the control is selected in the Painter.
- Pressing F5 when the Properties dialog has focus displays the Hierarchy for Form dialog.

The Hierarchy for Form dialog is unloaded if the currently selected form being painted is unloaded or if the Jade Painter is closed.

The window state and size of the dialog and the checked status is saved when you close the dialog. When the dialog is next displayed, the state of the form that is displayed is determined by the last saved state.

Applies to Version: 2016.0.01 and higher

» To change the parent of a control from the Hierarchy for Form dialog from version 2018.0.01 and higher

1. While holding the Shift key down, left-click the name of the control to be re-parented in the list of control names.
2. While holding down the mouse button, drag the mouse to the new parent control in the Hierarchy for Form dialog and then release the mouse.
3. A message box is then displayed with the following message, prompting you to confirm the parent change.

```
Make 'control-name' ('control-class-name') a child of 'new-parent-name'
('parent-class-name') and preserve left and top?
```

The message box contains three buttons. Click:

- **Yes**, if you want to change the parent and retain the current top and left position within the new parent.
- **No**, if you want to change the parent and retain the same screen position within the new parent.
- **Cancel**, if you want to abort the parent change process.

Note that while dragging a control, moving the mouse:

- Above the list box entries containing the control and form names scrolls the list up.
- Below the list box entries containing the control and form names scrolls the list down.
- Over the expanded node image causes the list item to be expanded one level.
- Over an item that cannot have control children displays a no-drop allowed style cursor image.
- Over an item that can host the control being dragged as a child displays a hand with pointed finger cursor image, indicating that you can drop the dragged control.

The status text of the completed drag and drop process is displayed at the bottom of the Hierarchy for Form dialog when you release the mouse over an entry in the list box.

Defining Control Name Prefixes

You can define a prefix for a **Control** subclass (for example, a prefix of **lbx_** for a **ListBox** control) so that when you add a control to a form in Painter, Jade inserts the appropriate prefix when prompting you for the name of the control.

To prefix a control, you must first define a **controlNamePrefix** method for the appropriate control subclass. You can add this method, which returns a string containing the appropriate prefix for that control, to the **Control** class or to any of its subclasses.

The following example shows a Jade method defined for the **Button** class in a user-defined schema.

```
controlNamePrefix(): String;
vars
begin
    return "btn_";
end;
```

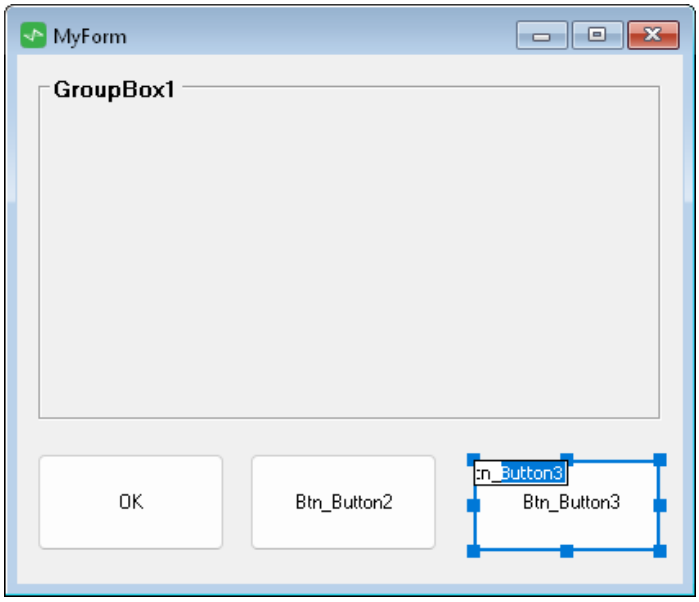
The **controlNamePrefix** method is used only by the Jade Painter. No check for a control name prefix is made when you use the Class Browser or logic to add controls.

If you do not define a valid **controlNamePrefix** method for a control class, the Jade Painter uses the default control naming rules, which may be overridden.

If the control class has a **caption** property, the initial value for the caption is the control name (with a capitalized first letter and without the control name prefix applied) if you accept the default name when prompted to do so, or the actual text (with a capitalized first letter) that you specified as part of the name. For example, if the Painter detects you have changed the control name caption (for example, from **Btn_Button1** to **Btn_OK**), the **caption** property is made equal to the caption name with the prefix removed so that the caption displays only the value that you specified (in this example, **OK**).

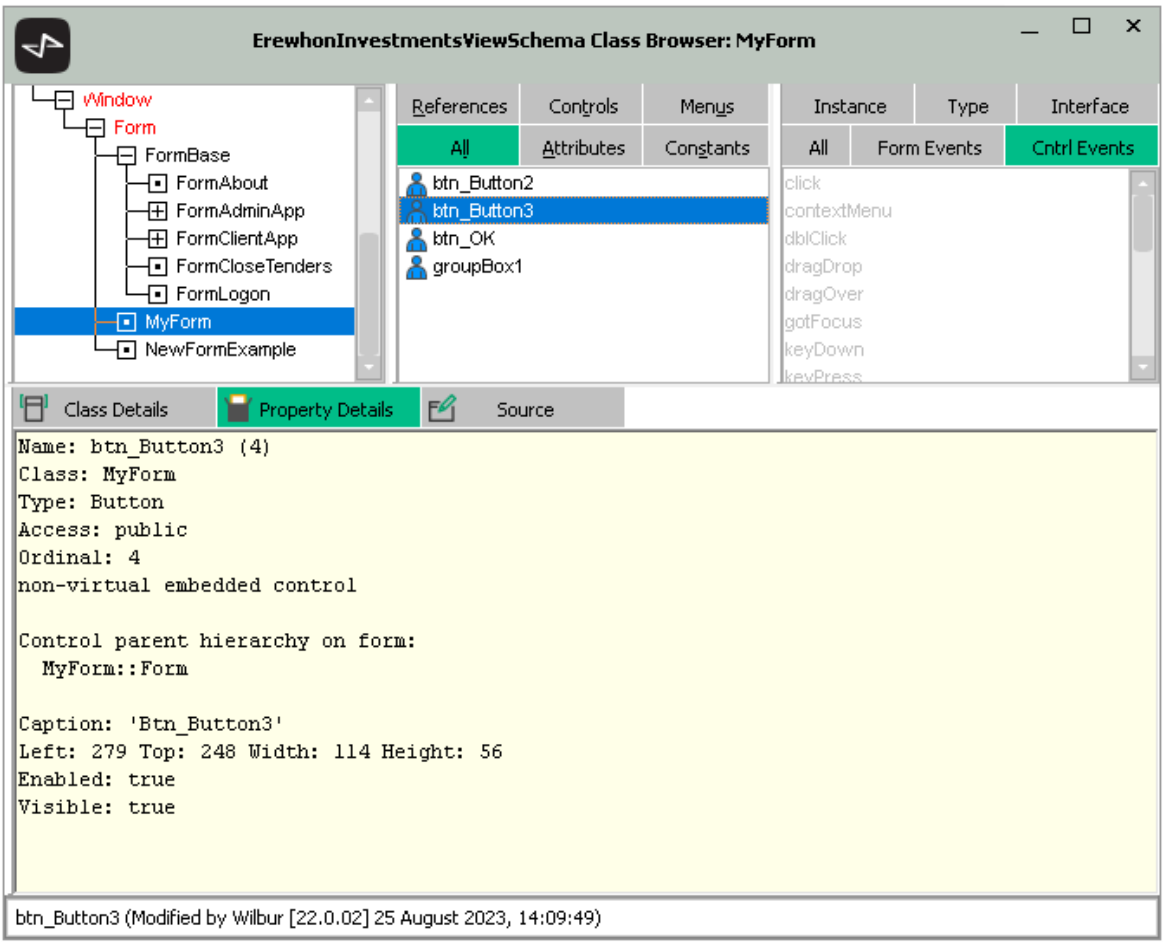
The **name** property for the control displays the full name (**btn_OK** in this example) in the Properties dialog.

The following image is an example of a form containing a group box that has no defined prefix and three buttons that have a defined prefix. This illustrates that only the non-prefixed portion of the control name is automatically selected so that you can change it to a caption of your choice, if required.



Now when you select your form in the Class List of the Class Browser, the controls listed in the Properties List display the prefixes that you defined for the controls in your schema.

The following image shows the controls specified for the **MyForm** form specified in the previous example.



Notes The **controlNamePrefix** method can be defined once only for any class in a schema branch. You can therefore not override a prefix if the control is in a superschema and it has already implemented the **controlNamePrefix** method. In this case, set the superschema to the current schema and then change the method.

Control prefixes are not applied when you create a form using the Form Wizard, which prefixes all controls with **ctl_**. (The type of control is not known until the form is built after you have specified the name of the control.)

Translating Control Properties

When the forms management style of the schema is **FormsMngmt_Single_Multi (2)** and you press F6 or Shift+F6 in the right column of the row of one of the properties listed in the following table, the Add String dialog or Select String dialog, respectively, is then displayed, which enables you to select the translatable string to apply to the control selected on the form.

Property	Class and its Subclasses in which the Property Value Can be Translated
bubbleHelp	Window

Property	Class and its Subclasses in which the Property Value Can be Translated
caption	Button, CheckBox, JadeDockBase, Form, Frame, GroupBox, JadeMask, Label, MenuItem, OptionButton, Sheet, StatusLine
helpKeyword	MenuItem, Window
mask	JadeEditMask
text	JadeEditMask, TextBox

The name of the string prefixed with a dollar sign (**$**), rather than the string definition itself, is then displayed in the property value column; for example, **$Agent_Commission_Rates**.

When the **Schema** class **formsManagement** property is set to **FormsMngmt_Single_Multi (2)**, forms are loaded from and saved to the schema default locale, regardless of the current locale. When the value of a translatable property is changed and the value looks like the name of a translatable string, a lookup is performed for a matching translatable string.

If a valid translatable string is located, the value of it is painted on the form instead of the property text. If the translatable string is not valid (that is, it is not found or it takes parameters), the property text is painted; for example, **$Agent_Commission_Rates**.

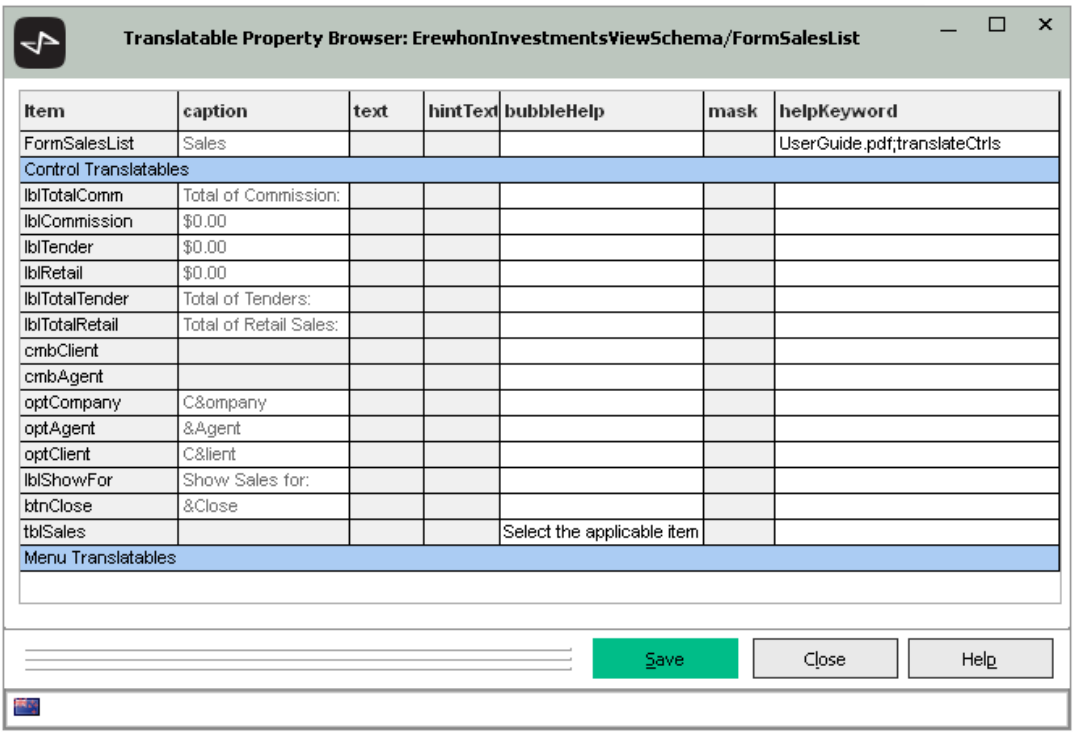
For details about defining and selecting translatable strings, see "Defining Translatable Strings from the Editor Pane" and "Inserting a Translatable String into an Existing Jade Method", respectively, in Chapter 4. For details about forms management styles, see "Forms Translation Styles", in Chapter 11.

Using the Translatable Property Browser

When a form defined in a schema with the forms management style of **FormsMngmt_Single_Multi (2)** is displayed in the Jade Painter and you select the **Translate Properties** command in the Window menu, the Translatable Property Browser is then displayed. This browser provides you with a quick mechanism to set up or assign translatable strings to the properties on a control.

Although you can use the Translatable Property Browser to specify new values for a control property, it is most useful when setting up properties with translatable strings so that different locales can have an easy translation mechanism for a single form.

The following is an example of the Translatable Property Browser.



Inherited controls are not shown in the browser.

The **Item** column at the left of the browser lists the names of the controls whose properties you can translate. The properties that are available for translation are displayed with a white background in cells for that control row; properties that are not available for translation for a control have a gray background in the cell of that control row.

Use the String Browser to maintain translatable strings associated with the current schema. (For details, see "Using the String Browser", in Chapter 11.) When you change a string definition in the String Browser of a control property displayed in the Translatable Property Browser (for example, from French to Japanese), the new translatable string definition is then displayed in the appropriate cell.

To change a property to use an existing translatable string

1. Perform one of the following actions.
   - Double-click the appropriate cell on the table
   - Select the cell and then press Shift+F6

   The Select String dialog is then displayed. (For details about using this dialog, see "Inserting a Translatable String into an Existing Jade Method", in Chapter 4.)
2. In the Select String dialog, select the string that you want to use in the translation and then click the **OK** button.

   The selected translatable string is then displayed with a green background in the appropriate cell on the Translatable Property Browser.
3. To update the control properties for the form, click the **Save** button. (You must still save the form itself in the Jade Painter.)

The form is then displayed in the Jade Painter with the translated string values for the currently selected locale. (The current locale is displayed in the Jade Painter status line.)

To add a new translatable string to a property

1. Select the cell and then press F6.

   The Add String dialog is then displayed. (For details about adding a string, see "Adding Strings", in Chapter 11.)

2. In the Select String dialog, select the string that you want to use in the translation and then click the **OK** button.

   (For details about using this dialog, see "Inserting a Translatable String into an Existing Jade Method", in Chapter 4.)

The selected translatable string is then displayed with a green background in the appropriate cell on the Translatable Property Browser.

Allowing Control Docking

You can use control docking on your forms if you want to allow users to drag a docking control (and all of its children) to a new position on the form or to cause that control to float in an independent floating window.

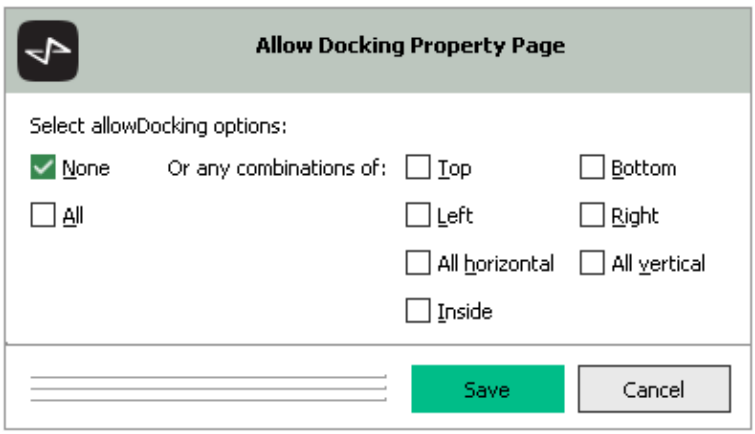
The changes required to implement docking controls enable you to have multiple status lines, multiple frames aligned to the top or bottom of the form, to align all controls that use the space remaining on the parent, and to align all multiple controls in the same parent so that they share the space remaining after other siblings have been aligned.

To specify that a form, frame, or docking container control allows docking within it

1. Select a **Form**, **Frame**, or **JadeDockContainer** control on the form.

2. Click the **Specific** icon on the Properties dialog. The **Specific** sheet of the Properties dialog is then displayed. (For details, see "Maintaining Properties for Your Form or Control" and "Maintaining Specific Control Properties", later in this chapter.)

3. Select the **allowDocking** property and then select the **Change** value from the drop-down list in the right column of the **allowDocking** property row. The Allow Docking Property Page dialog, shown in the following image, is then displayed.



4. To specify the docking that you require for the control, perform one of the following actions.

   - Clear all docking options by checking the **None** check box
   - Set all docking options by checking the **All** check box

- Specify individual options by checking or unchecking the values that you require or do not want to apply, respectively

The default value of the **allowDocking** property for **Form** and **Frame** classes is **None** (that is, the **None** check box is checked when you access the Allow Docking Property Page dialog and a **Form** or **Frame** control has focus).

The default value for **JadeDockContainer** controls is **Inside** (that is, only the **Inside** check box is checked when you access the dialog when a docking container has focus in the Painter). The **Inside** value means that the control is docked without any special alignment applying, and is intended for use with a **JadeDockContainer** control that has multiple **JadeDockBar** controls as children (for example, a toolbar that has several **JadeDockBar** controls, each holding a logical grouping of controls).

For docking examples and details about floating and docking controls, see the JadeDockBar, JadeDockBase, and JadeDockContainer classes, in Chapter 2 of the *Encyclopaedia of Classes*.

Initializing the JadeRichText Control

When you insert or paste the **JadeRichText** control onto a form in the Jade Painter, it is not running in a mode that allows text to be entered in the control. Use the **JadeRichText** class **initialContent** property, accessed from the Properties dialog, to initialize the control.

If the text of this property starts with a valid RTF header sequence (for example, **"{\rtf"**), the RTF reader loads that text when the form is run. All other text is displayed as plain text.

Although you would normally type the text with which the control is initialized into the Properties dialog **initialContent** property in the Jade Painter, this requires you to have some knowledge of RTF codes (that is, control words and symbols). For details about using the Properties dialog, see "Maintaining Properties for Your Form or Control", later in this chapter.

You could use the **JadeRichText** control itself to supply the RTF string, by performing the following actions.

1. Create another form (for example, **TempRTF**) containing a **JadeRichText** control and a **Button** control.
2. Add the following code to the **click** event of the **Button** control of the **TempRTF** form.

```
begin
    // rtfControl is value of the name property for the RTF control
    app.copyStringToClipboard(rtfControl.selTextRTF);
end;
```

3. In the Jade Painter, run the second form (**TempRTF**, in this example) and then enter the text, formatting it to meet your requirements.
4. When the appearance of the control meets your requirements, press Ctrl+A to select all text and then click the form button. The **click** event then copies the selection to the clipboard as encoded text that contains RTF control words and symbols.
5. In the Properties dialog for the RTF control on your original form, press Ctrl+V in the **initialContent** text box so that the RTF text string, including its control words and symbols, is then displayed.

Defining the Layout of Your Form

The Layout menu commands enable you to fine-tune the sizing and placement of controls on your painted form. For example, if you have painted a row of buttons, you may want to space them neatly on your form and make them all the same size.

Notes Some sizing and alignment subcommands cannot be selected if you have not selected a group of two or more controls (that is, they are disabled). All alignments are relative to the whole form unless you have selected controls within a container control, in which case alignments are relative to the container. For details, see "Adding Container Controls", earlier in this chapter.

Some alignment commands require you to select a group of controls. In those commands that align relative to a master control, the last control that you select in a group becomes the master. The master control is displayed with sizing handles.

The sizing and placement commands are:

- Alignment
- Spacing
- Size
- Suspend Parent Alignments

Tip You can use the **Undo Last Layout** command from the Edit menu to undo the last command that you actioned.

Sizing Your Controls

The **Size** command from the Layout menu accesses a submenu of commands that enable you to standardize the size of the controls on your form. You can also use the **Size** command to scale a control so that it fits the current contents of the control.

Tip When laying out your form, use the **Size** submenu commands before using the **Alignment** and **Spacing** layout commands, as the **Size** commands can alter the spacing of your controls.

Before you align and space the controls on your form, you may want to make some controls identical in size; for example, button controls. You may also want to standardize the width or the height of other controls.

For more details about the sizing controls, see "Size Command", later in this chapter.

The **Size** submenu commands are listed in the following table.

Command	Toolbar Button	Description
Same Width		Makes all selected controls the same width
Same Height		Makes all selected controls the same height
Same Height and Width		Makes all selected controls the same height and width
Standard Size		Resizes all selected controls to their default sizes

Alignment Controls

The Layout menu **Alignment** command accesses a submenu of commands that enable you to align selected groups of controls.

The **Alignment** submenu commands are listed in the following table.

Command	Toolbar Button	Description
Left		Aligns to the left edge of the master control
Right		Aligns to the right edge of the master control
Top		Aligns to the top edge of the master control
Bottom		Aligns to the bottom edge of the master control
Centre Horizontally		Centers the selected controls horizontally
Centre Vertically		Centers the selected controls vertically

You cannot select the **Left**, **Right**, **Top**, and **Bottom** commands if you have not yet selected a group of controls (that is, they are disabled). You can align controls across frame container controls, if required. For more details, see "Alignment Command", later in this chapter.

Spacing Your Controls

The Layout menu **Spacing** command accesses a submenu of commands that enable you to neatly and logically space your controls across or down the form or container control.

The **Spacing** submenu commands and the corresponding toolbar buttons are listed in the following table.

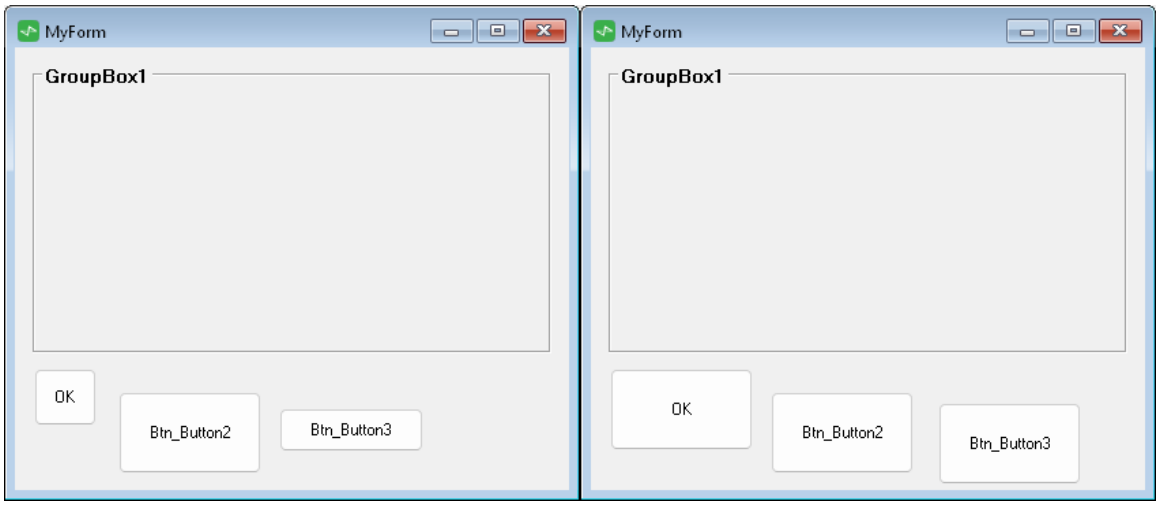
Command	Toolbar Button	Description
Space Evenly Down		Spaces the selected controls evenly down the container
Space Evenly Across		Spaces the selected controls evenly across the container
Spread Down Container		Spreads the selected controls evenly down the entire container
Spread Across Container		Spreads the selected controls evenly across the entire container
Vertically Adjacent		Makes the selected controls vertically adjacent
Horizontally Adjacent		Makes the selected controls horizontally adjacent
Vertical Standard Spacing		Automatically spaces controls vertically
Horizontal Standard Spacing		Automatically spaces controls horizontally
Top/Right Command Buttons		Positions command buttons at top right
Bottom/Right Command Buttons		Positions command buttons at bottom right

You cannot select the **Space Evenly Across**, **Space Evenly Down**, **Horizontally Adjacent**, and **Vertically Adjacent** commands if you have not yet selected a group of controls (that is, they are disabled). To enable the **Space Evenly Across** and **Space Evenly Down** commands, you must select at least three controls.

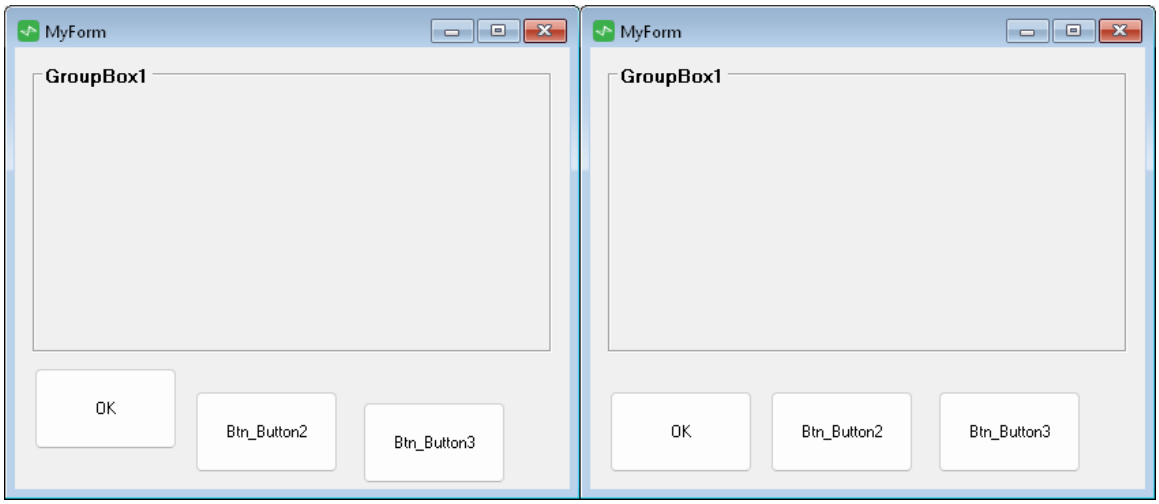
When using the **Vertical Standard Spacing**, **Horizontal Standard Spacing**, **Vertically Adjacent**, and **Horizontally Adjacent** commands or toolbar buttons, the position of the master control (that is, the control that is displayed with sizing handles) remains unchanged and not the control that is in the top or the farthest left position.

Examples of Laying Out Controls

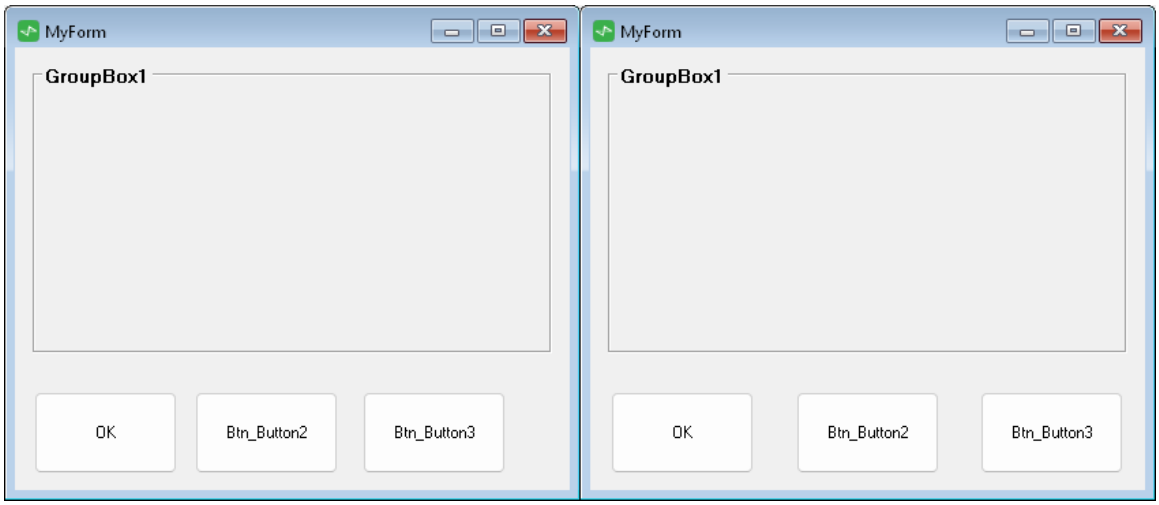
The following example shows the use of the layout commands to arrange the buttons on a form. The buttons were all selected and then made the same size by using the **Size** commands.



With the buttons still selected, use the **Alignment** commands to align your controls, as shown in the example in the following image.



Finally, you can then space the selected controls neatly across the form, by using the **Spacing** commands, as shown in the example in the following image.



In this example, the buttons have been moved relative to the whole form. If they had been part of a container control, they would have been spread evenly relative to the container, not to the form.

Using Grids to Position Controls on Forms

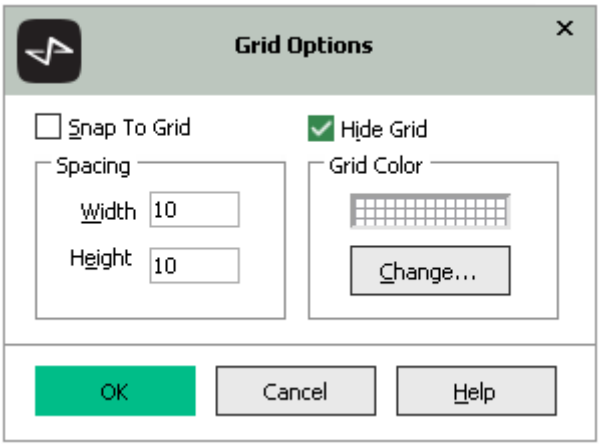
A grid assists you in positioning your controls on a form.

You can toggle the display of a grid or you can change your grid options, as required.

To access the Grid Options dialog

- Select the **Grid** command from the Options menu.
- Click the **Grid** toolbar button (toggles the grid display on or off only).

When you select the **Grid** command from the Options menu, the Grid Options dialog shown in the following image is then displayed.



To specify your grid options

1. Check the **Snap To Grid** check box if you want to control the placement of your controls.

   When this feature is enabled, the grid lines dictate the placement and movement of the controls on your form, and controls snap to the grid even when the grid is not displayed.

   Note For web forms, the **Snap To Grid** check box is checked by default.

2. Uncheck the **Hide Grid** check box if you want the grid displayed in your Painter windows.
3. In the Spacing group box, use the **Width** and **Height** text boxes to change the default horizontal and vertical spacing of your grid from the default values of 10 pixels, if required.
4. In the Grid Color group box, click the **Change** button if you want to change the color of your grid lines. The common Color dialog is then displayed. Select the color that you want for your grid and then click the **OK** button.

   Tip If you are using a 16-color palette, the default grid color may match the background color and you will therefore need to use this feature to make the grid visible. In other cases, changing the grid color is optional.

5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Using Alignment Hairs to Position Controls on Forms

Alignment hairs assist you in positioning your controls on a form, by drawing a window margin on your form and displaying alignment hairs when you move or resize controls by dragging them with the mouse.

Note The window margin is not displayed for a web form.

You can toggle the display of alignment hairs, as required.

To toggle the display of alignment hairs on your form

- Select the **Show Alignment Hairs** command from the Options menu.
- Click the **Alignment Hairs** toolbar button.

When the display of alignment hairs is selected, a check mark is displayed to the left of the command in the Options menu and alignment hairs (lines) are then applied to your form. (To change the option back again and hide the display of alignment hairs on your forms, repeat the action. The check mark is then no longer displayed in the Options menu.)

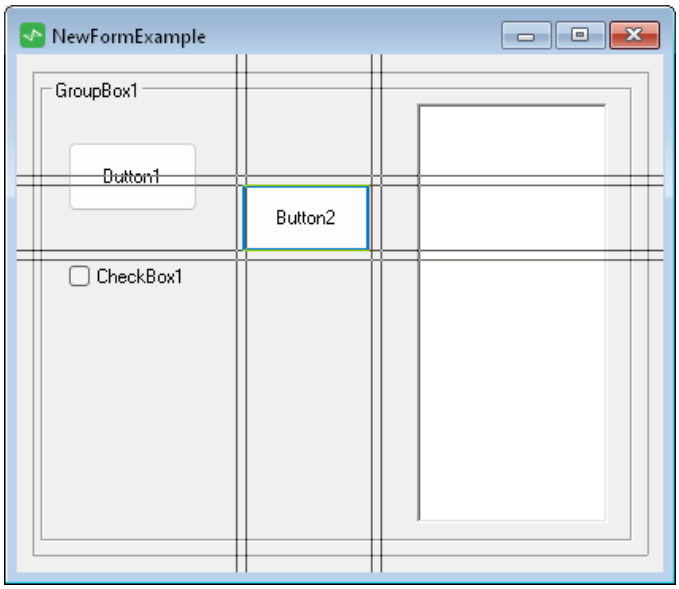
The window margin provided by the alignment hairs contains your controls so that you can maintain a standard margin between the controls and the border of the form.

The margin is drawn in the current grid color, and is for guidance only. It is not drawn on your runtime form.

Note If you have selected the grid display, the alignment border is not displayed but the grid is drawn only up to an imaginary alignment border so that the bounds of the border are apparent from the extent of the grid.

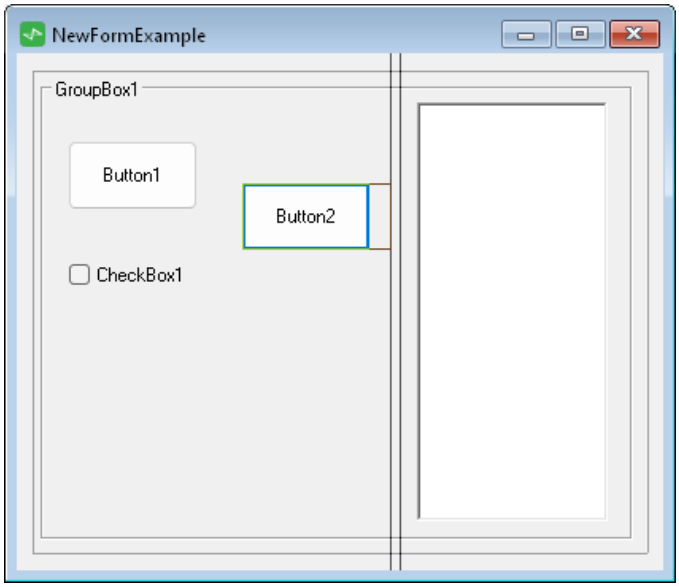
Alignment hairs enable you to quickly and accurately align your controls when you move controls by dragging them with the mouse.

The hairs extend the width and height of the form, and align with the sides of the master control, as shown in the following image.



Another set of hairs is displayed parallel to the first set, but offset by the standard control spacing. This enables you to maintain a standard gap between your controls.

When alignment hairs are displayed and you drag a sizing handle, alignment hairs are drawn only on the side of the control on which the sizing handle is being dragged. An example of alignment hairs positioned beside a dragged sizing handle is shown in the following image.



If you resize a control that can have children, an extra alignment hair is displayed inside that control. This enables you to resize the parent while maintaining the recommended space between a child and its parent border.

What Happens Next

When you have created your first form for the application, it becomes the default start-up form when you run your application.

During development, you can nominate another form as the start-up form, by using the **Change** command from the Application menu in the Application Browser. (For details about using the **Application** sheet of the Define Application dialog, see "Defining Applications", in Chapter 3.)

You can also change the start-up form in your code, by setting the **Application**::**startupForm** property.

Maintaining Properties for Your Form or Control

To maintain the properties associated with your form or control, use the **Show Properties Dialog** command from the Window menu. Alternatively, click the **Properties** toolbar button.

The display of the Properties dialog varies, depending on whether you selected a form or control, or on the type of control. In general, there are five pages, or list types, covering the various properties.

Note Click the **Stay on top of Painter** icon at the top left of the dialog or select the **Properties on Top** command from the Options menu to keep the Properties dialog on top of the Painter, or click the **Help** icon (**?**) to access online help for the dialog.

To save the current form when the Properties dialog has focus, press F2.

Most of the properties have default settings and many property values are set automatically when you manipulate your form or controls in Painter or you use some of the menu commands; for example, the **Layout** or the **Options** command.

In the Properties dialog:

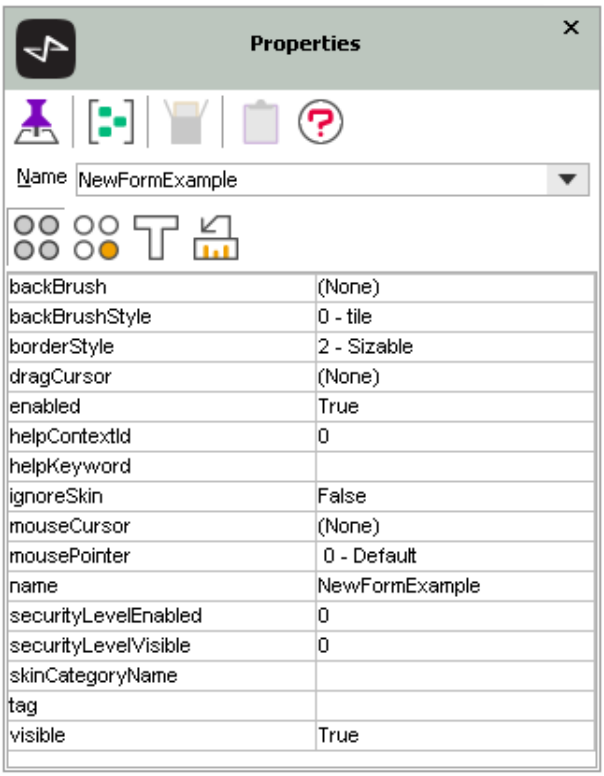
- The value of a property in the right column is displayed with blue text when the selected control is inherited from a superclass, indicating that you cannot change the value.
- When you click a text value in the right column and the selected control is inherited from a superclass, the content of that text box is read-only, so that you can select and copy text.
- Combo boxes in the right column are disabled for inherited control values.

For details about form and control properties, see Chapter 2 of the *Encyclopaedia of Classes*.

» To access the Properties dialog for the current form or control

- Select the **Show Properties Dialog** command from the Window menu
- Click the **Properties** toolbar button
- Press F4

The Properties dialog, similar to that shown in the following image, is then displayed.



The **Common** page is displayed by default when the Properties dialog is first opened.

Tip You can resize the first column of the Properties dialog, by dragging the vertical grid line to the required position. This may be useful, for example, when viewing ActiveX control property names that can be very long.

The Properties dialog is also displayed when you:

- Press F4 when the Hierarchy for Form dialog has focus.

  Conversely, the Hierarchy for Form dialog is displayed when you press F5 when the Properties dialog has focus.
- Double-click a control name in the list box of the Hierarchy for Form dialog. (The Properties dialog is then displayed after the control is selected in the Painter.)

Double-clicking a property in the Properties dialog marks the form as having been modified when:

- You toggle a boolean value.
- The value is part of a list in which case the value of the **listIndex** property is incremented.
- A Property Pages dialog is displayed and the **showPropertyPage** method returns that a value has been changed.
- A common dialog is displayed and is not cancelled.

Double-clicking an entry calls the Property Pages dialog only if the **getPropertyDisplay** method that is called returns **true** and the **hasPropertyPage** method also returns **true**. If the **getPropertyDisplay** method does not return **true**, the Property Pages dialog must be opened by using the **Change** entry in the combo box associated with the property in the column on the right of the **Specific** page of the Properties dialog.

» **To change the properties of your form or control, click one of the icons listed in the following table**

To change the …	Icon	For details, see…
Common properties of your form or control		Maintaining Common Properties
Properties specific to your form or control		Maintaining Specific Control Properties
Size and position of the window of your form or control		Maintaining Size and Position Properties
Color or font of your form or control		Maintaining Font and Color Properties

Each change that you make to a property is applied immediately the focus is changed from that property or when you press Enter.

Using the Name Combo Box

Use the **Name** combo box to select the control or form that you want to edit, by:

- Selecting the control or form in the drop-down list box portion of the combo box
- Partially entering the required entry so that it is automatically located

  As you enter the text, the first entry that has the specified text as a prefix is selected. Pressing Enter then selects that entry in the combo box, displays the properties for the control in the Properties dialog, and selects the control on the form.

The **Name** combo box contains the name of the object for which properties are displayed. You can view properties for any other object, by selecting that object from the list.

Using the Control About Box Icon

If the control that you have selected is an ActiveX control, click the **Control About Box** icon to display details about that control. If the control is a standard Jade Painter control, click the **Control About Box** icon to display details about the version of Jade that you are using.

If the control does not implement its own About box, the default Control About Box is displayed.

Using the Property Page Dialog Icon

Click the **Property Page Dialog** icon to display the Property Pages dialog for your ActiveX control.

The Property Pages dialog displays all the property pages for the ActiveX control in tabbed sheets. Use this dialog to change the property settings for the control.

This button is enabled only when the selected control is an ActiveX control.

Using the Update all selected controls Icon

Click the **Update all selected controls** icon if you want your property changes to be applied to two or more controls.

The current state of the icon is saved with your other Painter settings when you close the Properties dialog, for use when you reopen the dialog.

Maintaining Common Properties

Use the **Common** page of the Properties dialog, accessed by clicking the **Common** icon, to define the common properties for your form or control. The **Common** page is displayed when you first access the Properties dialog. (For an example of the **Common** page, see "Maintaining Properties for Your Form or Control", earlier in this chapter.)

To change the common properties of your form or control

1. Click the property that you want to change. The property is then highlighted and the associated common property list box or text box enables you to make a selection or entry.
2. If a text box is associated with the property, make your change by overwriting the existing value in the text box.
3. If a list box is associated with the property, click on the list box and then make your selection from the menu that is displayed.
4. If you click the non-default selection in the list box for the **dragCursor** property or the **mouseCursor** property, the common File Open dialog is displayed, to enable you to select your image file.
5. When you click on the **skinCategoryName** property, a combo box with a list of skin category names is displayed.

   The list contains only those category names that are assigned to a skin of the same type or superclass type as the window currently selected in the Painter. The list box also contains a blank entry, to enable you to clear the current skin category name. When the required skin has not been loaded into the current development environment, you can also specify a skin category name that is not displayed in the list.

To change the parent of a control

1. Click on the control whose parent you want to change. The property is then highlighted.
2. Select the **parent** property and then select the **Change** item in the drop down list box that is then displayed. This list box enables you to change the direct parent of the control to the form or to another control. The Change Parent dialog is then displayed.

   The control selected in the Jade Painter is displayed in the **Control** text box, for informational purposes only. (If you want to change the parent of another control, you must first close the Change Parent dialog and repeat steps 1 and 2 in this process.)

   The current parent object of the selected control is highlighted in the **Parent** list box.
3. In the **Parent** list box, select the form or control that you want the selected to reference at run time.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

Note The display properties of a control are relative to the parent control. Changing the parent form or container of a control affects the placement and visibility of the child control.

Maintaining Specific Control Properties

Use the **Specific** page of the Properties dialog, accessed by clicking the **Specific** icon, to define properties specific to your form or control.

To change the specific properties of your form or control

1. Click the property that you want to change. The property is then highlighted, enabling you to use the associated list box or text box to make an entry or selection.
2. If a text box is associated with the property, make your change by overwriting the existing value in the text box.
3. If a list box is associated with the property, click the list box and then make your selection from the menu that is displayed.
4. For details about changing the **allowDocking** property value for a form or for the Frame or **JadeDockContainer** control, see "Allowing Control Docking", earlier in this chapter.
5. For the **backBrush** property or the **icon** property, if you click the non-default selection in the list box, the standard File Open dialog is displayed, to enable you to select your image file.

Maintaining Font and Color Properties

Use the **Font/Color** page of the Properties dialog, accessed by clicking the **Font/Color** icon, to change the font or colors of your form or control.

To change the font or colors of your form or control

1. Click the property that you want to change. The property is then highlighted, enabling you to specify the required font attributes in the associated text boxes or combo boxes.
2. Make your change by overwriting the existing value in the text box, or selecting the required value in the list box.

Maintaining Size and Position Properties

Use the **Size/Position** page of the Properties dialog, accessed by clicking the **Size/Position** icon, to change the size and position of your form or control within its window.

To change the Size/Position properties of your form or control

1. Click the property that you want to change. The property is then highlighted, enabling you to specify the required size and position in the associated text boxes.
2. Make your change by overwriting the existing value in the text box, or by selecting the required value in the list box.

Alternatively, you can select the **Standard Size** command from the Size layout submenu when one control or a group of controls is selected.

The size of the selected controls is then adjusted dynamically to the standard size of the dialog units for your application (by default, 14 dialog units).

Designing Menus

The **Menu Design** command from the File menu enables you to easily create and customize a menu bar for your painted screen.

Menu Design provides the following features for developing a menu.

- Menu bar creation
- Up to 20 levels of submenus
- Automatic insertion of standard Help and Window menus

- Description (text) associated with a menu command
- Link to online help for a menu command
- Shortcut keys associated with a menu command
- Ability to make menu commands checked, disabled, or invisible
- Security governing who can view or use a menu
- The following menu accelerator keys options.
  - Ctrl+0 through Ctrl+9
  - Ctrl+Shift+A through Ctrl+Shift+Z
  - Ctrl+Shift+0 through Ctrl+Shift+9

  The Jade Painter Menu Design form displays:
  - A list of accelerators that are available for use for the current selected menu item. This list contains any alphabetic characters in the menu captions that are not used as an accelerator in any other peer menu item.
  - Top-level menu item accelerators of any superclass forms are excluded from the list if the menu item is a top-level item. Subclass forms are not considered.
  - The accelerators used for any menu items in the same submenu list are excluded if the menu item is a submenu.
  - A message, in red, if the currently selected menu item has an accelerator that is duplicated in any peer menu item. This is a warning only. Duplicated accelerators are permitted and handled. The message includes the caption of the menu item that duplicates use of the accelerator key.

For a menu design example, see "Example of Designing a Menu", later in this chapter.

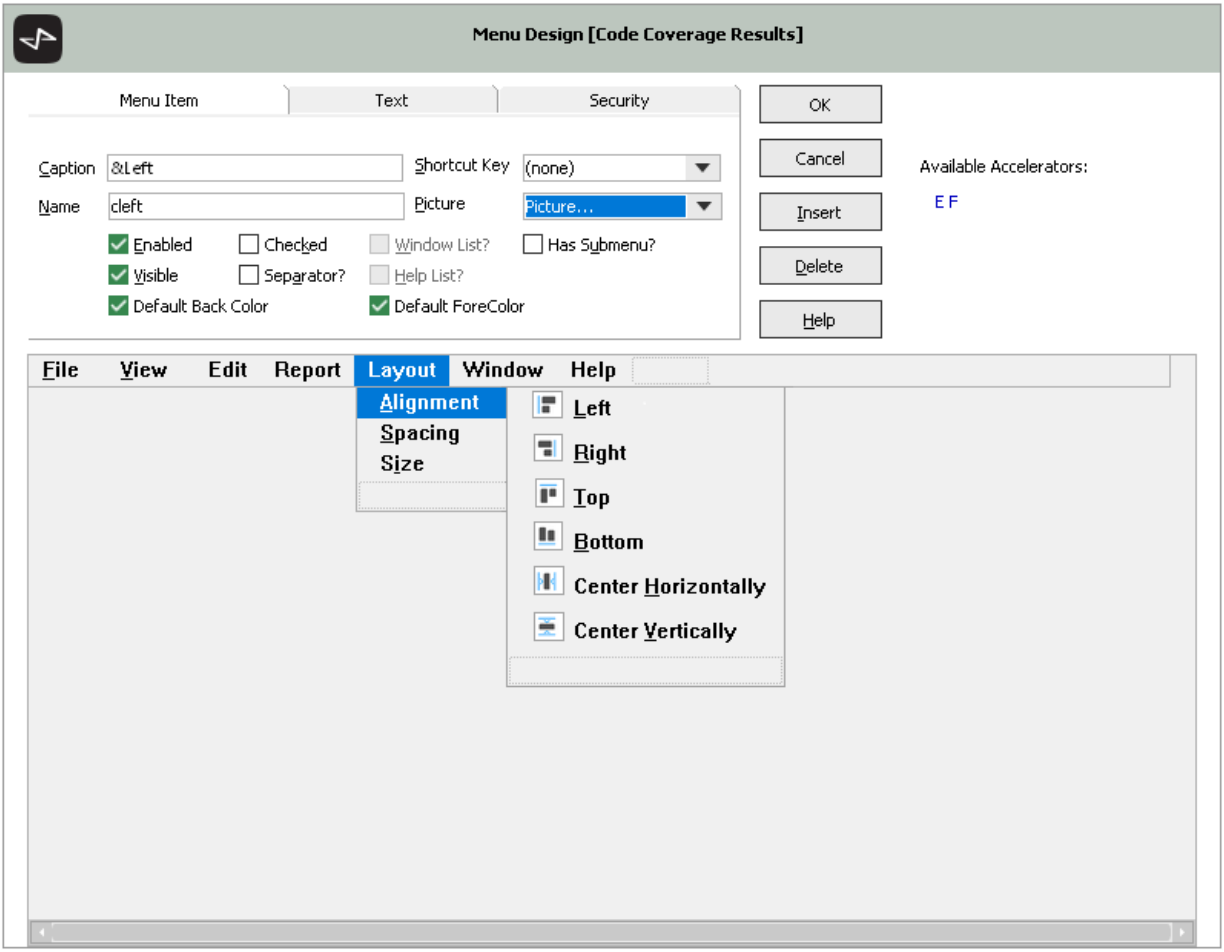
Using the Menu Design Command

Use the **Menu Design** command to create and maintain a menu for a form.

» To access the Menu Design command, perform one of the following actions

- Select the **Menu Design** command from the File menu
- Click the **Menu Design** toolbar button
- If you have already created a menu, double-click the menu bar on your form

The **Menu Design** dialog, shown in the example in the following image, is then displayed. The lower half of the dialog displays the menu that you are building.



To create and maintain your menu using the Menu Design dialog

1. If this is a new menu, define the first menu bar item by using the **Menu Item** sheet of the Menu Design dialog. An empty cell is automatically created, to enable you to define the next menu item.

   To create a menu item, you must specify a value in the **Caption** and **Name** text boxes. The **Name** value is required only if the menu that you are defining does not have a submenu. You can specify any other properties later, if necessary.

2. Click an empty cell to define a new menu item or click an existing menu item to make changes. When you add a menu bar item, an empty cell is created to the right of that item on the menu bar and another empty cell is created below it, in a new submenu.

3. To create a submenu, check the **Has Submenu** check box on the parent menu. A submenu is automatically created and the dialog creates the first empty cell in the new submenu. Menu bar items have the **Has Submenu** check box checked by default.

4. Specify the required properties on the Menu Design dialog sheets for the current menu item.

5. Repeat steps 2 through 4 for each menu item that you want to add or change.

6. Click the **OK** button.

The additions and changes you have made to your menu are accepted, the Menu Design dialog is closed, and the Painter window is then displayed. (If you want to abandon your menu additions and changes, close your form without saving it.)

Maintaining Menu Item Properties

The **Menu Item** sheet is displayed when you first access the Menu Design dialog or when you click the **Menu Item** sheet on the Menu Design dialog.

To create a menu item, you must specify a value in the **Caption** and **Name** text boxes. The **Name** value is required only if the menu that you are defining does not have a submenu or the menu item is defined as a separator. You can specify any other properties later, if necessary. (For details about creating a menu separator dynamically at run time, see the **MenuItem** class **caption** property in Chapter 1 of the *Encyclopaedia of Classes*.)

To maintain the properties on the Menu Item sheet

1. In the **Caption** text box, specify the text to be displayed as the label text for your menu item.

   The **caption** property for a menu is a string with a maximum length of 100 characters.

   Note You can use the **MenuItem** class **caption** property to assign an access key to your menu. In the caption, include an ampersand character (**&**) immediately preceding the character that you want for an access key. The character is underlined. At run time, press Alt followed by your underlined character to move the focus to that control.

   To include an ampersand character in a caption without creating an access key, enter two ampersand characters (**&&**). A single ampersand character is then displayed in the caption and no characters are underlined.

2. In the **Name** text box, specify the internal name for your menu item. The name must start with a letter, and can include numbers and underscore characters, but cannot include punctuation symbols or spaces.

3. If you want quick access to this menu, use the **Shortcut Key** combo box to select a shortcut key. (You cannot apply a shortcut key to menu items that have submenus.)

   The Painter checks that the shortcut key combination is not already assigned to a menu item on the form being painted, any superclass forms, and any subclass forms.

   If the selected shortcut key combination (for example, Ctrl+F2) is already assigned, the *Allow reuse?* warning message is displayed, identifying which schema, form, and menu item already has that shortcut key assigned. You are prompted to click the:

   - **No** button if you do not want the shortcut key or keys reused. The **Shortcut Key** combo box is then reset to the previously selected entry.
   - **Yes** button, to update the menu item with the selected shortcut key or keys.

4. If you want to display a picture on your menu label, select **Picture** in the **Picture** combo box. The common File Open dialog is then displayed, to enable you to select the required graphics file.

5. If you want the menu disabled at run time, uncheck the **Enabled** check box.

6. If you want to hide the menu at run time, uncheck the **Visible** check box.

7. If you do not want to use the default background area of a menu, uncheck the **Default Back Color** check box. The common Color dialog is then displayed, to enable you to select the background color that you require.

   The background color is used only when an inner image is not set or it is not a brush.

8. If you do not want the foreground color of menu text, uncheck the **Default ForeColor** check box. The common Color dialog is then displayed, to enable you to select or define the foreground color that you require for menu text.

9. To check a menu item, check the **Checked** check box. At run time, a checked menu has a check mark displayed to the left of it. (You cannot check menu items with submenus.)

10. To specify a separator line in the current menu, check the **Separator?** check box.

    A separator line logically separates groups of commands.

    The following sequence of events is required to insert a separator in the current menu.

    a. Click the blank cell at the bottom of the menu.

    b. Click the **Separator** check box so that it is checked (selected).

    c. Enter the appropriate caption and name details in the **Menu Item** sheet.

11. When you have created your top-level menu item, you can then select the:

    a. **Window List?** check box, to automatically include the Window menu containing the standard CUA menu commands on your menu bar. (This check box is checked by default.)

       The **Close All MDI Forms** command is displayed in the Window menu list on the **Menu Item** sheet only when the **Window List?** check box is checked. To add the **Close All MDI Forms** command to an existing Window list menu, uncheck the **Window List?** check box and then check it again.

       At run time, if you select this menu item, all visible, enabled forms that do not have the **allowClose** property set to **false** are unloaded as if the user has selected the **Close** button on each MDI child form. The **Form** class **queryUnload** and **unload** event methods are called, as normal. If any form rejects the unload (via the **cancel** parameter in the **queryUnload** event method), the unloading process will cease.

       In addition, if the MDI frame has no visible and enabled MDI child windows displayed at run time, the **Arrange Icons**, **Cascade**, **Close All MDI Forms**, **New Window**, **Tile Horizontal**, **Tile Vertical**, and **Close All Windows** menu items are disabled.

    b. **Help List?** check box, to automatically include the Help menu containing the standard CUA help commands on your menu bar (that is, the **Index** and **About** commands).

       When the **Index** command is selected, help is displayed based on the value of the **helpKeyword** property. If the value of the **helpKeyword** property is null, displayed help is based on the value of the **helpContextId** property. If this is zero (**0**), index help is displayed. The **click** method is never executed.

       The **click** method is never executed when the **About** command is selected.

    c. **Has Submenu?** check box, to automatically create a submenu for the selected menu item. (Menu items that have submenus cannot be checked or have a shortcut key applied.)

    Top-level menus marked as a Window list or a Help list by using these check boxes are always placed in the correct Windows order at run time and in the Jade Painter (that is, the Window menu followed by the Help menu are always the last two menus, regardless of where they are placed in the Menu Designer). The position of these menus is based on the setting of the **Window List?** and **Help List?** check boxes, and not the names specified in the **Name** text box.

    If you check only the **Window List?** check box, the Window menu is the last menu on the menu bar.

Maintaining Menu Text Properties

The **Text** sheet is displayed when you click the **Text** sheet on the Menu Design dialog. The lower half of the dialog displays the menu item that you are building. Entries in the text boxes of the **Text** sheet are optional.

To specify or change the text properties for your menu

1. In the **Disable Reason** text box, optionally specify text that can be used to apply a runtime message that tells the user why the selected menu command has been disabled. (You must provide the code to handle this.)
2. In the **Description** text box, optionally specify your own documentation describing the current menu item. Typically, this description is displayed in the status bar of your application when the menu item is selected, or used as bubble help on web pages when no bubble help is specified.
3. In the **Help Id** text box, optionally specify the identifier (context number) that relates to the Help entry for this menu item. (If you specify a value in the **Help Keyword** text box, that value is used for context-sensitivity in preference to any value that you enter in this **Help Id** text box.)

   This action sets the **helpContextId** property for the menu item. If you have created a Windows environment help file for your application, when a user presses F1 with the focus on a menu item, Jade automatically calls help and requests the topic identified by the current **helpContextId** property. For details, see "Creating Context Links to Your Own Application Help File", in Chapter 2.
4. In the **Help Keyword** text box, optionally specify the identifier (context number) that relates to the Help entry for this menu item. (If you specify a value in the **Help Keyword** text box, that value is used for context-sensitivity in preference to any value that you enter in the **Help Id** text box.)

   This action sets the **helpKeyword** property for the menu item. If you have created a Windows environment help file for your application, when a user presses F1 with the focus on a menu item, Jade automatically calls help and requests the topic identified by the current **helpKeyword** property.

   For details, see "Creating Context Links to Your Own Application Help File", in Chapter 2.

Maintaining Menu Security Properties

The **Security** sheet is displayed when you click the **Security** sheet on the Menu Design dialog. The lower half of the dialog displays the menu that you are building.

To specify or change the security properties for your menu

1. In the **Enabled** text box, specify the number representing the security level for the menu command. At run time, the menu command is enabled only for users who have a security level equal to or greater than the level that you specify.
2. In the **Visible** text box, specify the number representing the security level for the current menu command. At run time, the menu command is visible only to users who have a security level equal to or greater than the level that you specify.

Inserting a Menu Item Using the Insert Button

To insert a menu item

1. Click the menu item that will be displayed after the menu item you want to insert (that is, click the menu item to the right of the insertion if it is on the menu bar, or below the insertion if it is in a submenu).
2. Click the **Insert** button. A blank cell is then inserted in the required position.
3. Define the properties for the menu item that you are inserting, by using the sheets of the Menu Design dialog.
4. Click the **OK** button.

The additions and changes you have made to your menu are accepted, the Menu Design dialog is closed, and the Painter window is then displayed. (If you want to abandon your menu additions and changes, close your form without saving your changes.)

Deleting a Menu Item Using the Delete Button

To delete a menu item

1. Click the menu item that you want to delete.
2. Click the **Delete** button. The selected menu item is then deleted.

To reverse the deletion, click the **Cancel** button to abandon the changes that you have made since you last clicked the **OK** button.

Moving a Menu Item

To move a menu item

1. Click and drag the menu item that you want to move.

   Dragging your menu item across the menu screen causes submenus under the cursor to be displayed, enabling you to select your insertion point.
2. Drop the menu item onto the menu item that will immediately follow it.

If you have moved your menu item to the menu bar, the menu item that you dropped it on is moved one place to the right to make room for your insertion.

If the menu item was moved to a menu below the menu bar, the menu item on which you dropped it is moved one place down the menu to make room for your insertion.

When you move a menu that has submenus, the submenus are also moved.

Example of Designing a Menu

The following subsections provide an example of menu creation using the File menu **Menu Design** command.

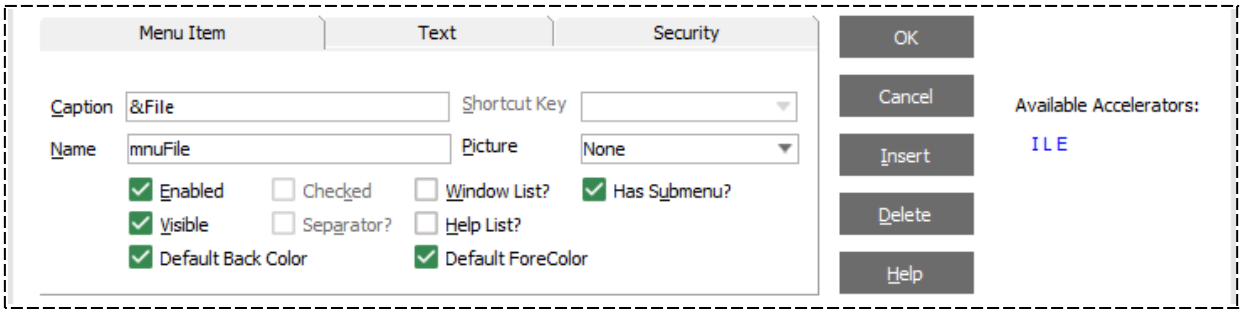
1. Creating the Menu Bar
2. Creating the First Submenu Level
3. Adding Further Submenu Levels
4. Adding Standard Help and Window Menus
5. Adding More Menu Design Features

Creating the Menu Bar

When you access the **Menu Design** command from the File menu, the Menu Design dialog is then displayed. Start designing your menu by defining the menu bar.

You must provide at least a caption for all menu items and you must provide a name for menus defined at the lowest level; that is, menus that have no submenus. (The name is optional at all other menu levels).

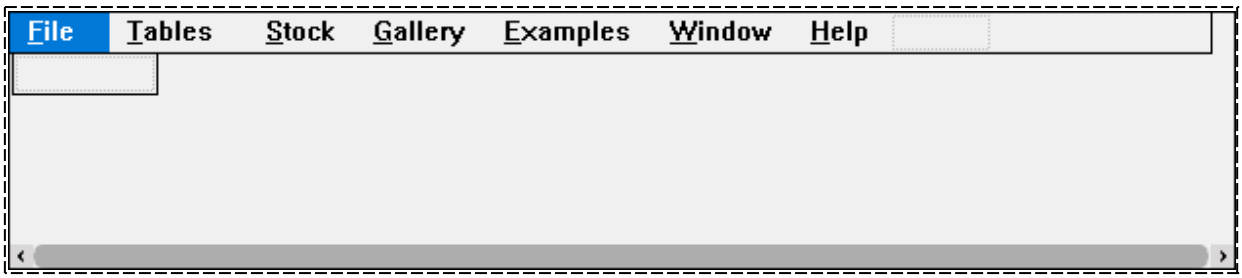
Use the top half of the Menu Design dialog, shown in the following example, to define the properties for each menu item.



The lower half of the dialog displays the menu that you are building.

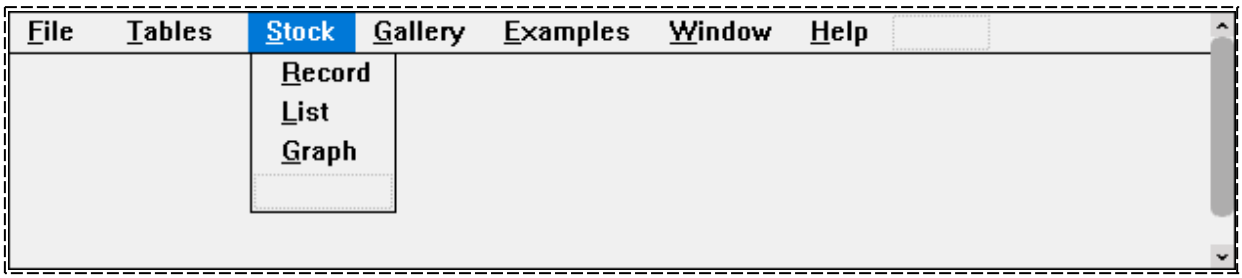
Creating the First Submenu Level

When you click one of your menus on the menu bar, the blank cell that is automatically displayed below it is the position for your first submenu entry. In the following image, the File menu has been clicked.



A blank cell is created for your next entry each time you enter a command on a submenu.

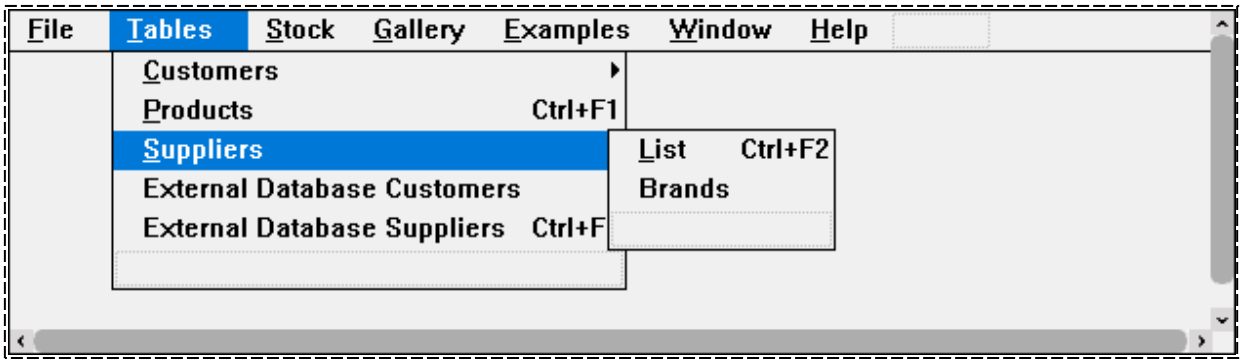
You can create submenus for each of your submenus, simply by clicking the **Has Submenu** check box on the Menu Design dialog. In the following image, three submenu items have been added to the Faults menu.



Adding Further Submenu Levels

When you check the **Has Submenu** check box on the Menu Design dialog, a further level of submenus is created, with the first empty cell marked for your entry.

In the following example, the **Suppliers** command has had a submenu created for it, with two menu items: **List** and **Brands**.

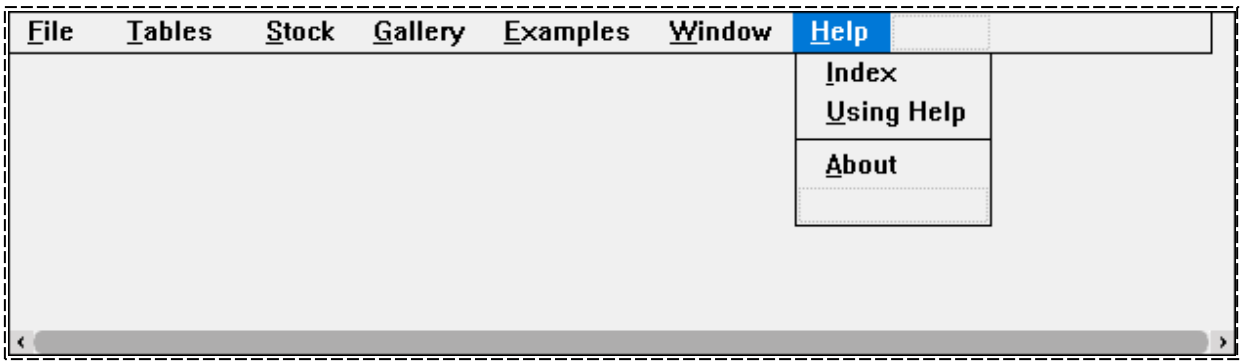


In this example, a submenu has already been defined for the **Customers** command, indicated by the arrow symbol at the right of the command.

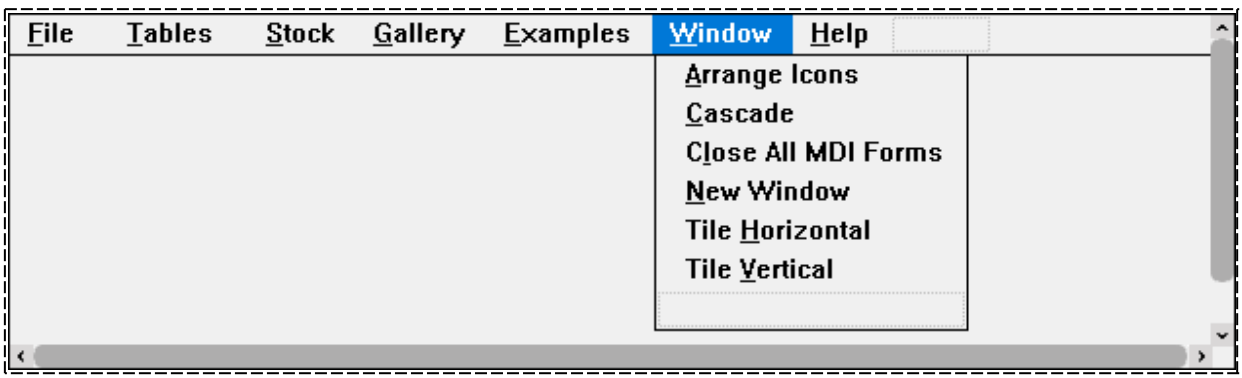
This submenu can be displayed and maintained simply by clicking the **Customers** menu. (For details about assigning a shortcut key to a menu item that does not have a submenu, see "Maintaining Menu Item Properties", earlier in this chapter.)

Adding Standard Help and Window Menus

The following example shows the standard Help menu that is displayed simply by checking the **Help List** check box in the dialog at the upper left of the Menu Design window.

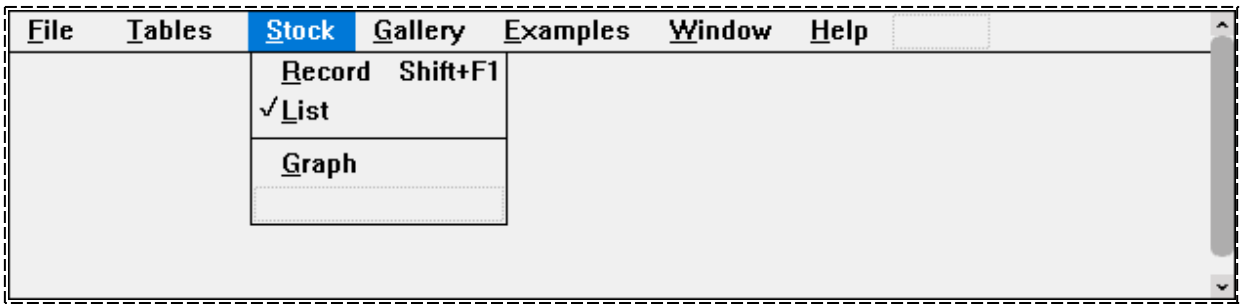


The following example shows the standard Window menu that is displayed simply by checking the **Window List** check box in the dialog at the upper left of the Menu Design window.



Adding More Menu Design Features

Other features of the **Menu Design** command are shown in the following image. Under the Stock menu, a Shift+F1 shortcut key has been selected for the **Record** command. On the same submenu, the **List** command has had the **checked** property applied and a **Separator** has been inserted between the **List** and **Graph** commands.



Defining Methods for Your Form

To define methods to support your forms, controls, and menus, see "Defining and Compiling Jade Methods and Conditions" in Chapter 4 of this document, and "MenuItem Class" in Chapter 1 of the *Encyclopaedia of Classes*.

Testing Your Form

Use the **Run Form** command from the File menu to test a form that you are developing. The **Run Form** command runs any form, whether that form was created by you or generated by using the Form Wizard. The **Run Form** command is disabled:

- When the form is in the latest version of a versioned schema
- Until a form is loaded into the Painter; that is, selected for edit

» To run a test of your form

- Select the **Run Form** command from the File menu.

The form runs as though in runtime mode; that is, you can use it to enter data. When running a form, the following may apply.

- It may fail if there is any dependency on another form or another process being run first.
- If the form is an MDI child form, it is run inside an MDI test frame.

Selecting Skins for Painted Forms

Use the **Select Skin** command from the File menu to select a runtime skin that is used to display any form that you are painting. Conversely, if a runtime skin is currently displayed in Painter forms, you can specify that no skin is displayed in your forms in the Jade Painter.

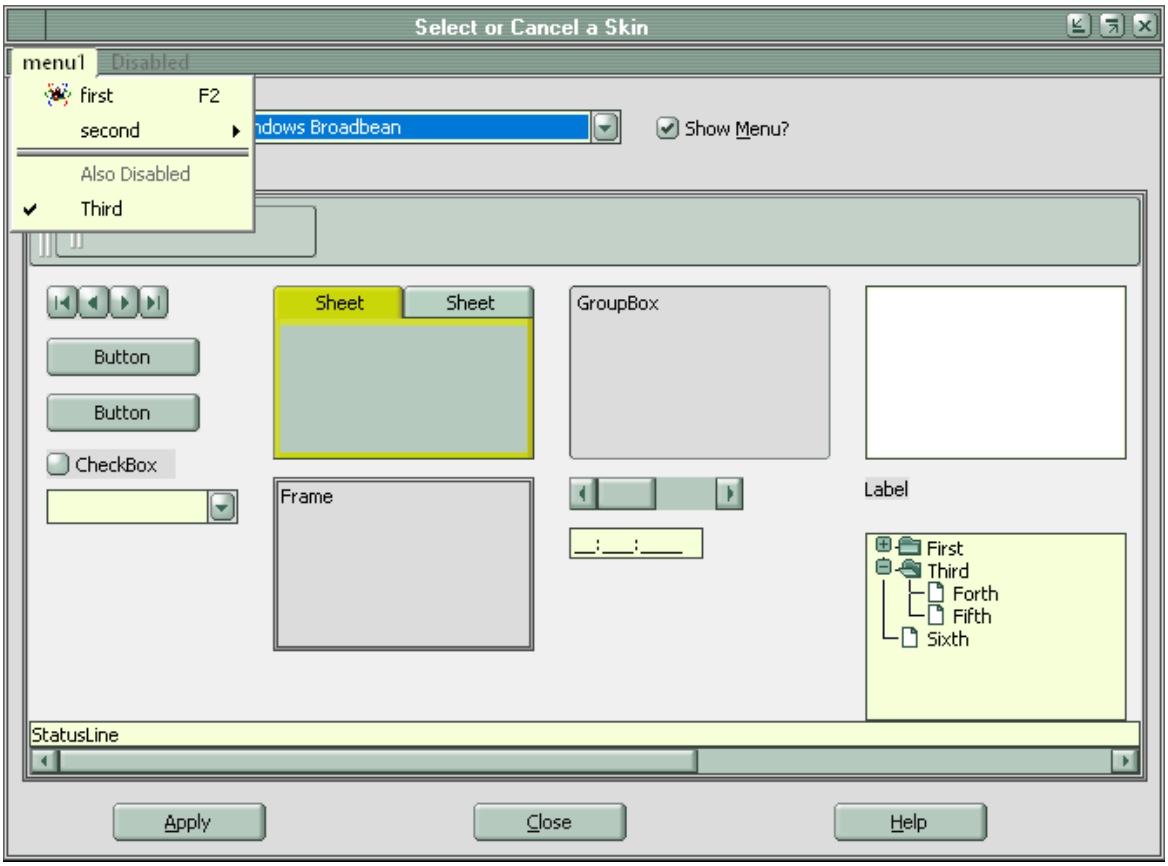
To select or cancel the display of a runtime skin on painted forms

1. Select the **Select Skin** command from the File menu.

   The Select or Cancel a Skin form is then displayed, to enable you to select the runtime skin in the **Choose Skin** combo box. If you have not loaded any runtime skins into your Jade system, the default value of **<None>** is the only value available in this combo box.

   Tip The **examples\skins** subfolder of the Jade install files contains runtime skins that you can load. For details about loading the **SampleSkins.ddb** or **SampleSkins.ddx** file, see the **readme.txt** file in that subfolder.

2. When you select a runtime skin, the **Control Examples** pane on the form displays an example of controls (and menu and menu items, if selected for display) using that skin, as shown in the following example.



3. When you are happy with the controls and menu on the painted form displayed in that skin, click the **Apply** button. That skin is then applied to any forms being painted.

If a skin is selected, the Jade Painter caption reads Jade Painter : *schema-name::form-name* - using skin '*skin-name*' - [*caption-of-form*]; for example:

```
Jade Painter : DemoSchema::Company - using skin 'Windows Broadbean' - [Company]
```

In addition, any subsequent forms opened in the Jade Painter are displayed using the selected runtime skin.

The selected skin is saved in your user preferences when you close the Jade Painter and restored when you re-open the Painter.

Applies to Version: 2016.0.01 and higher

Saving Your Form

To save your form, perform one of the following actions

- Select the **Save Form** command from the File menu.
- Click the **Save Form** toolbar button.
- When the Properties dialog has focus, press F2 to save the current form.
- Select the **Close Form** command, and then click the **Yes** button when asked if you want to save the changes to your form. (To discard your changes since your last save, click the **No** button when closing your form.)

When multiple monitors are running on a workstation and you save a form in the Jade Painter, the values of the **left** and **top** properties are converted to be relative to the top and left of the primary monitor. For details about saving your form as a different name or style (for example, to save a printer form as a web-enabled form), see "Save Form As", later in this chapter.

Editing an Existing Form

To load an existing form so that you can maintain it, perform one of the following actions

- Select the **Edit Form** command from the File menu.
- Select the user-defined form in the Class List of the Class Browser and then select the **Edit Form** command from the Classes menu.
- Click the **Edit form** toolbar button.
- Select the form from the list of most recently used forms at the bottom of the File menu.

The **Load Form** dialog is then displayed, to enable you to specify or select the name of the form that you want to load for editing.

To load an existing form

1. If you do not want to edit a form for the Painter's current schema, in the **Schema** combo box, select the schema that contains the form that you want to edit.

   The list box area of the **Form** combo box then displays the forms defined for that schema.

2. In the text box of the **Form** combo box, specify the name of the form that you want to edit or select the form from the list box. Alternatively, when the **Form** text box has focus, use the up (↑) and down (↓) arrow keys to scroll up or down the list of forms so that the form selected in the list is displayed in the **Form** text box.

   Tip For details about filtering the entries displayed in the **Form** list box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.

3. Check the **Show as Hierarchy** check box if you want to display a hierarchical list of all forms. In hierarchical form, the filtering process is not performed.

   This check box is unchecked by default, so that the list of forms is displayed alphabetically sorted and it allows the filtering process to be performed.

4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Using the Keyboard and Mouse to Edit Your Form

When the selected form has been loaded, use combinations of keyboard and mouse actions to manipulate the controls on the form.

You can also change details of a control or a group of controls by using the Properties dialog. For details, see "Maintaining Properties for Your Form or Control", earlier in this chapter. For details about changing the type of an existing control (for example, changing a **GroupBox** control that does not have children to a **Frame** control), see "Changing a Control Type", later in this chapter.

All controls in the form are made visible and are enabled in the Painter, regardless of the setting of the **visible** and the **enabled** properties of each control.

However, a control inherited from a superclass form is disabled, regardless of the setting of the **enabled** property of that control on the current form. This provides you with a visual indicator in the Painter that you cannot edit the control on the current form, as it belongs to the form of a superclass.

For details, see the following subsections.

Selecting a Control to Edit

To select a control

- Click the control that you want to edit.

A selected control is highlighted by the drawing of a thick border around the control in the Windows highlight color. Eight boxes, or sizing handles, are also drawn around the border of the control, to enable you to resize it.

Changing the Size of a Selected Control

To change the size of the selected control

1. Move the mouse pointer slowly over a sizing handle, so that the cursor changes shape to a double-ended arrow as it passes over a small square.
2. Click the mouse and then hold down the left mouse button while moving the mouse in one of the directions pointed to by the double-ended arrow.

Selecting Multiple Controls

To select multiple controls, perform one of the following actions

- Drag out a selection rectangle on the form.

  Any control that intersects the sizing rectangle is then selected.

- Click on each control to be selected, while holding down the Shift key.

The sizing handles are drawn only on the last control that you selected.

To deselect a control that is already selected

- While holding down the Shift key, click on the control that you do not require in the selection.

You can select a number of controls by dragging out a selection rectangle if you perform the mouse down part of the drag operation over a vacant area of the form. (Starting the drag operation over a control is interpreted as a movement of the control.)

When dragging out a rectangle:

- Pressing the Ctrl+Shift keys and then dragging the rectangle allows selection even if the start of the drag operation is over a control.
- If the control move lock operation is on (that is, you selected the **Lock Control Positions** command from the Options menu or clicked the **Control move lock** padlock toolbar button), the drag selection can start over a control, as the controls cannot be shifted.

Changing the Size of Selected Controls

To change the size of all selected controls

- Drag a sizing handle on the master control.

Moving Controls about Your Form

To move a control or a group of selected controls about your form

- Drag the control or selected controls that you want to move.

A control that you can drag is indicated by the cursor changing shape to a four-headed arrow as it passes over the control. All selected controls move during the drag operation.

You can use the Shift key and the drag operation to move a selected control or controls. Selecting more than one control requires you to hold down the Shift key or the Ctrl key during selection. As it is possible to inadvertently shift or copy controls during a multiple selection operation, turn on the control move lock functionality (by selecting the **Lock Control Positions** command from the Options menu or clicking the **Control move lock** padlock toolbar button) and then use the Shift key during multiple control selection).

Copying a Control

To copy a control or a group of selected controls that has the same parent

- Hold down the Ctrl key and then drag the selected control or controls to the new location on the form.

To copy selected controls that have different parents, use the copy and paste edit functions.

You can use the Ctrl key and the drag operation to create a copy of the selected control or controls at a new location. Selecting more than one control requires you to hold down the Shift key or the Ctrl key during selection.

As it is possible to inadvertently shift or copy controls during a multiple selection operation, turn on the control move lock functionality (by selecting the **Lock Control Positions** command from the Options menu or clicking the **Control move lock** padlock toolbar button) and then use the Shift key during multiple control selection).

Displaying the Properties Dialog for a Control

To display the Properties dialog for a control, perform one of the following actions

- Double-click on the control
- Press F4
- Select the **Show Properties Dialog** command from the Window menu

The Properties dialog initially displays details of the selected control. You can use the Properties dialog to change the control whose details are displayed, if required. That control then becomes the selected control on the form that you are editing.

You can use the Properties dialog to change the same property on all selected control at once, by clicking the **Update all selected controls** icon. For details, see "Maintaining Properties for Your Form or Control", earlier in this chapter.

Positioning a New Control on Your Form

When you add a new control to the form, the cursor is displayed as cross hairs when it moves across the form.

To position your new control, perform one of the following actions

- Click the position on the form at which you want to locate the new control. (The new control is created as the default size.)
- Drag out the rectangle, to change the control size.

Placing a Control on a Container Control

Container controls (for example, a frame or group box) allow you to place other controls on top of them. Place a control on a container control as you would when positioning a control on the form. If you position your control over a control that is not a container in add mode, the cursor changes to the NO cursor. For details about allowing docking in a container control, see "Allowing Control Docking", earlier in this chapter.

A control that is positioned on a container control becomes a child control of the container, or parent, control. The position coordinates of a child control are always relative to its parent control. Child controls remain relative to the parent so that they move if the parent control moves.

To change the parent of a control, perform one of the following actions

- Use the Properties dialog.
- Drag the child control onto a container control.

When you drag a control, the control under the mouse pointer at the time of the mouse up action is made the parent control, unless the NO cursor is displayed, in which case the drag operation is aborted. The parent control is not changed when the mouse up (or drop) is actioned over a selected control.

Removing a Control from Your Form

- Select the control that you want to remove and then press the Delete key.

Making Small Positional Adjustments to a Control

To make small adjustments to the position of the selected control

- Use the arrow keys.

An arrow key changes the position of the control in the direction indicated by the arrow by one pixel each time the key is pressed.

Changing a Control Type

You can change the type of a control on a form, providing that the change does not break any rules regarding the control that can be the parent and the type of child control types that are permitted.

The control name, event method code, and so on, are retained, so that you do not have to first delete the existing control, add the replacement control of the required type, and then create the appropriate event methods.

To change the existing type of a control selected on a form

1. Select the **Change Type** command from the Controls menu.

   If the selected control has existing method references, a message box then displays the name of the control and the number of methods that reference it, advising you that the methods will be marked for recompile. Click the **Yes** button to confirm that you want to change the control type now. When you save the form (by selecting the **Save Form** command from the File menu), all method references to the control are then automatically recompiled during the save operation. (Alternatively, click the **No** button if you want to abandon the changing of the control type.)

   The Change Control Type dialog is then displayed. The **name** property and the existing type of the selected control are displayed in the **Control** and **Current Type** text boxes, respectively. As these are read-only for information purposes, you cannot change these values.

2. In the **New Type** list box, select the new type that you require for the control. The control type that you select must conform to the rules about the permitted parent control and child control types. For example, you can change a **GroupBox** control that contains child controls to a **Frame** but you cannot change it to a **Folder** (as a **Folder** control can have only **Sheet** controls as children).

   Although you cannot change a **GroupBox** control that has children to a **Button** control (as **Button** controls cannot have children), you can change a **GroupBox** control that does not have children to a **Button** control. If the selected control type is invalid, a message box is then displayed. Click the **OK** button to return to the Change Control Type dialog and select a valid control type.

3. When you have selected a valid control type, click the **OK** button in the Change Control Type dialog. Alternatively, click the **Cancel** button to abandon your selection.

Effects of Editing an Existing Form

If the form that you are editing is being used at run time, you are warned that the form cannot be saved until no user is using the form. If you try to save the edited form, the save action is disallowed until the form is not being used.

When you access an existing form in the Jade Painter, you automatically lock the form class. Although locking a form prevents other users from making changes, they can view the unchanged form. When you exit from Painter, the lock is released.

Removing ActiveX Controls from Forms

When an ActiveX control cannot be located and installed, you cannot run or edit the form on which it resides.

If an ActiveX control cannot be loaded when a form is opened for editing, the Jade Painter now displays a message box, prompting you to confirm that the load should continue. The message box displays:

```
The load of the control control-name of type control-type failed with an
exception: error-text error-code
```

Click:

- **Yes** to continue the load action. The control is then marked as deleted and saving the form permanently deletes the control from the form.

  The loading of the form continues and the control is marked as being deleted. The control and any event methods for the control are removed from the form when you save it.

- **No** to abort the load action.

  The form load is aborted and abandoned.

Note The form load cannot be continued if the control being loaded is on a superform or if the control would have had children.

The ability to delete controls from a command file and from the Jade Platform development environment Class Browser will be considered in a future release. The only current work around is to extract the class and manually remove the control from the ***.cls** and ***.ddb** or ***.ddx** files.

Applies to Version: 2020.0.01 and higher

Painter Menus

This section includes information about the menus and commands available in the Jade Painter. The Jade Painter menus are listed in the following table.

Menu	Description
File	Administers a Jade Painter session and displays the forms in all schemas most recently opened for editing
Edit	Editing functions; for example, cutting, copying, or pasting part of a form
Controls	Adds and manipulates the controls on your painted form
Layout	Fine-tunes the sizing and placement of controls on your painted form
Options	Changes the options associated with the Painter screen
Window	Accesses the Properties dialog to enable you to maintain the properties associated with your form or controls, and displays the forms you have currently open
Help	Accesses the standard CUA help options

Menu commands that are not available for selection are disabled. For example, before you select a control, the **Bring to Top** command in the Controls menu is disabled and cannot be selected.

File Menu

The File menu commands are listed in the following table.

Command	Description
New Form	Opens a new form
Edit Form	Opens an existing form for editing
Form Wizard	Runs Form Wizard
Run Form	Runs the form
Select Skin	Opens the Select or Cancel a Skin form
Save Form	Saves the current form
Save Form As	Saves the current form as a new form with the new name
Copy Form Into Locale	Copies a form from one locale into another locale
Close Form	Closes the current form
Delete Form	Deletes the specified form
Delete Current Form	Deletes the currently active form
Menu Design	Provides access to the Menu Design facility
Browser	Switches to the Jade Schema Browser
Exit Painter	Exits from your Painter session

The File menu also displays the forms most recently opened for editing.

New Form Command

Use the **New Form** command from the File menu to create a new form. The New Form dialog is then displayed, to enable you to define your new form. (For details, see "Adding a New Form", earlier in this chapter.)

Use a form to create an interface for your Jade applications. A form is a window on which you paint controls for the running of your application. Controls are objects such as text boxes, list boxes, buttons, check boxes, and so on. You also use a form to define a print layout.

Note The first form that you create for an application becomes the default start-up form when you run your application. During development, you can nominate another form as the start-up form by using the **Change** command from the Application Browser. You can also change the startup form in your code, by setting the **Application** class **startupForm** property.

Edit Form Command

Use the **Edit Form** command from the File menu to maintain an existing form.

To edit an existing form, perform one of the following actions

- Select the **Edit Form** command from the File menu.
- Click the **Edit Form** toolbar button.

The **Edit Form** dialog is then displayed, to enable you to select the form to load. (For details, see "Editing an Existing Form", earlier in this chapter.)

Form Wizard Command

Use the **Form Wizard** command from the File menu to create a form quickly and easily.

The main purpose of Form Wizard is to help you create file maintenance-type forms where the focus is on the instances of one particular class. The form that you create with Form Wizard is complete in its own right, and can be executed immediately.

The painting of controls and the coding of simple methods associated with those controls are automatic. Form Wizard obtains the necessary information to create a form by prompting you for that information. When all the data has been gathered, a screen layout (form) is prepared, along with supporting methods.

Note You cannot use Form Wizard against a previously created form, whether that form was created with Form Wizard or not.

For details, see "Using Form Wizard to Create a Form", later in this chapter.

Run Form Command

Use the **Run Form** command from the File menu to test a form that you are developing.

When testing a form, the following points apply.

- You can use the **Run Form** command to run any form, whether that form was generated by you or generated by using the Form Wizard.
- The form must be selected for edit in the Painter before you can run it.
- The form runs as though in runtime mode; that is, it can be used to enter data. However, note the following restrictions.
  - The results are unpredictable if there is any dependency on another form or another process being run first.
  - An MDI child form is run inside an MDI test frame.

For details, see "Testing Your Form", earlier in this chapter.

Select Skin Command

Use the **Select Skin** command from the File menu to select a runtime skin that is used to display any form that you are painting. Conversely, if a runtime skin is currently displayed in Painter forms, you can specify that no skin is displayed in your forms in the Jade Painter.

For details, see "Selecting a Skin for Painted Forms", earlier in this chapter.

Applies to Version: 2016.0.01 and higher

Save Form Command

Use the **Save Form** command from the File menu to save your current form. (For details, see "Saving Your Form", earlier in this chapter.)

Note To discard your changes since your last save, select the **No** button on the Save dialog when closing your form.

Save Form As Command

Use the **Save Form As** command from the File menu to save your form as a new form with a different name or style (for example, to save a printer form as a web-enabled form). This command accesses the Save Form As dialog, which enables you to create a new copy of your form.

As you cannot save your form under a different superform or schema, the **Sub-Form of** and **Schema** combo boxes are disabled on this dialog. In addition, as you cannot copy a form as an existing form name in the current schema, you cannot select a form in the **Existing Forms** list box. The listed forms are for display purposes only, to enable you to view the names of existing forms.

For details about saving your form, see "Saving Your Form", earlier in this chapter.

Copy Form Into Locale Command

Use the **Copy Form Into Locale** command from the File menu to copy a form from one locale into another locale in the same schema (replacing the form in the target locale), when your schema contains two or more locales.

For details, see "Copying a Form into another Locale", later in this chapter.

Close Form Command

Use the **Close Form** command from the File menu to your close your form.

If you have not made any changes to your form since it was last saved, your form is then closed. If you have made changes since your form was last saved, a message box is displayed, asking you to confirm if you want to save it.

Delete Form Command

Use the **Delete Form** command from the File menu to delete a form. The Delete Form dialog that is then displayed enables you to specify the name of the form that you want to delete.

Notes You cannot delete a form if it has sub-forms.

When you delete a form, the corresponding class and all its methods are deleted. Any methods referencing the deleted form are flagged as *uncompiled*.

Delete Current Form Command

Use the **Delete Current Form** command from the File menu to delete the current form. This enables you to delete the currently active form while allowing you to visually confirm that it is the correct form.

If the form can be deleted (that is, it has no meta data inhibitors such as has property references), the menu command is enabled and the caption is **Delete Current Form:*active-form-name***.

If the form cannot be deleted, the menu command is disabled. Placing the cursor over the disabled command displays in the status line the reason that the form cannot be deleted (for example, the form has subclasses).

Clicking the **Delete Current Form** command:

- Reconfirms that the form can be deleted.
- Checks that the form is not in use elsewhere in the system.
- Confirms that you can perform the delete action (that is, the security library is called).

- Deletes the form and its class if there are no inhibitors.
- Closes the form in Painter if the form is deleted and discards any unsaved changes.

Applies to Version: 2020.0.01 and higher

Menu Design Command

Use the **Menu Design** command from the File menu to easily create and customize a menu bar for your painted screen.

For details, see "Designing Menus", earlier in this chapter.

Browser Command

Use the **Browser** command from the File menu to open a Class Browser for the current Jade form.

To open the Class Browser for the current form, perform one of the following actions

- Select the **Browser** command from the File menu.
- Click the **Browser** toolbar button.

A Class Browser for the current Jade form is then displayed. If a Class Browser already exists with that form class selected, it receives focus. If no Class Browser exists with that form class selected, a new Class Browser is opened.

When you open a Class Browser, your Jade Painter session changes are not lost. You can return to the Jade Painter at any time to continue maintaining Jade forms, by performing one of the following actions.

- Select the **Painter** command from the File menu.
- Press Ctrl+P.
- Press Alt+Tab until the Jade Painter window is displayed.

Exit Painter Command

Use the **Exit Painter** command from the File menu to exit from Painter. When you exit from Painter, you are returned to the Jade Browser.

To exit from Painter

- Select the **Exit Painter** command from the File menu.

If you have not made any changes to your forms since your last save (or your forms were "fast-saved"), your forms are closed, and you exit from Painter and return to the Jade Browser.

If you have made changes since your last save, a dialog is displayed, asking you if you want to save the current form. Take one of the following actions.

- Click the **Yes** button to save your form on closing.
- Click the **No** button to close the form and discard any changes since your last save.
- Click the **Cancel** button to ignore the exit request, and continue your Painter session.

Edit Menu

The Edit menu commands are listed in the following table.

Command	Description
Cut	Copies the selected control to the clipboard and deletes it from its current location on the form
Copy	Copies the selected control to the clipboard
Paste	Pastes the control from the clipboard to the form
Select All	Selects all controls on the form
Find	Locates a specified control on the form
Undo Last Layout	Undoes the last layout command

Edit Menu commands that are not available for selection are disabled. For example, the **Cut** and **Copy** commands are disabled until you select a painted control.

Cut Command

Use the **Cut** command from the Edit menu to delete the selected controls from your painted form and copy them to the clipboard.

The **Cut** command is not enabled until at least one painted control is selected.

Note Deleting a control causes the corresponding property of the class to be removed, which in turn marks as uncompiled any methods using or referencing that property.

If a selected control has children, the child controls are also cut.

To select a control or group of controls, perform one of the following actions

- To select one control, simply click it.
- If you want to select a group of controls, perform one of the following actions.
  - After selecting the first control, hold down the Shift key and then click the other controls that you want to include in the group.
  - Click and drag a box around the controls that you want to select.

To activate the cut operation, perform one of the following actions

- Select the **Cut** command from the Edit menu
- Press Ctrl+X

The selected controls are then removed from your form.

When you use the **Cut** command, the selected controls are copied to the clipboard from which they can be pasted elsewhere on the current form, or into another form. The clipboard is not saved between Jade sessions.

Note You cannot cut a control that is inherited. The **Cut** command is also disabled if there is an inherited control in the group of controls that you have selected.

Copy Command

Use the **Copy** command from the Edit menu to copy the selected control or controls to the clipboard.

The **Copy** command is not enabled until at least one control is selected.

Note If a selected control has children, the child controls are also copied.

To select a control or a group of controls, perform one of the following actions

- Click a control
- If you want to select a group of controls, hold down the Shift key and then click the other controls that are to be included in the group

To activate the copy operation, perform one of the following actions

- Select the **Copy** command from the Edit menu
- Press Ctrl+C

The selected control is then copied to the clipboard. The clipboard is not saved between Jade sessions.

When you use the **Copy** command, the selected control is left intact in its original position, but is copied to the clipboard, from which it can be pasted elsewhere on the current form, or into another form.

To copy a control or group of controls that have the same parent, by dragging

- Hold down the Ctrl key and then drag the selected control or controls to the new location on the form.

To copy selected controls that have different parents, you must use the copy and paste functions.

Paste Command

Use the **Paste** command from the Edit menu to paste a control or controls from the clipboard into your painted form. The pasted controls may have been cut or copied from the current form or from another form.

The **Paste** command is enabled only when you have cut or copied a control or controls to the clipboard. The clipboard is not saved between Jade sessions.

Pasted controls have default names generated for them of the form, in the following format.

```
original-name_nn
```

The *nn* value is an integer string starting at 1.

To paste a control or controls into the current form, perform one of the following actions

- Select the **Paste** command from the Edit menu
- Press Ctrl+V

When you use the **Paste** command, the controls are copied from the clipboard and placed in the top left position of your painted form. You can then arrange them on the form to meet your requirements. For details about the order in which controls are pasted on a form from the Painter clipboard, see "Changing the Runtime Tab Sequence", earlier in this chapter.

Select All Command

The **Select All** command from the Edit menu enables you to select all of the controls on your painted form so that you can reposition all your controls at once.

After selecting this command, position the caret in any of the controls, hold down the left mouse button, and then drag the control so that all of the controls move in formation.

Find Command

Use the **Find** command from the Edit menu to locate a specified control.

The **Find** command also enables you to bring the control to the top of any overlapping sequence of controls when it is located.

To find a control on the current form

1. Select the **Find** command from the Edit menu. The **Find a Control** dialog is then displayed.
2. Select the required control from the **Control to Find** combo box.

   Tip For details about filtering the entries displayed in the list box associated with the **Control to Find** combo box, see "Filtering Entries in Combo Box and List Box Text", in Chapter 2.
3. Check the **Bring to Top** check box if the control that you are searching for may overlap with other controls and you want it displayed on top of the others.
4. Click the **OK** button to confirm your selections.

The specified control is then highlighted in the current form.

Undo Last Layout Command

Use the **Undo Last Layout** command from the Edit menu to undo the last layout command that you actioned.

To undo the last layout command, perform one of the following actions

- Select the **Undo Last Layout** command from the Edit menu.
- Click the **Undo** toolbar button.
- Press Ctrl+Z

The last layout command that you actioned is then undone and the change made by that action is rolled back.

Controls Menu

The Controls menu commands are listed in the following table.

Command	Description
Add Control	Adds a new control to the form
Delete Control	Deletes the selected control from the form
Change Type	Changes the type of an existing control

Command	Description
Select Parent	Selects the parent of the currently selected child control
Select All Children	Deselects all currently selected controls and selects all immediate children of the current form or control
Select All Siblings	Deselects all currently selected controls and selects all immediate children of the parent control
Push to Bottom	Pushes the selected control behind the control that it overlaps
Push all Parents and Control to Bottom	Pushes the control and all of its parents to the bottom of the displayed zOrder
Bring to Top	Makes the selected control sit on top of the overlapping controls
Bring all Parents and Control to Top	Brings the control and all of its parents to the top of the displayed zOrder
Check For Duplicate Control Accelerators	Checks whether there are any duplicated control caption accelerators in the full hierarchy of the currently selected form
Find Available Caption Accelerator Characters	Shows the characters in a control caption and all superclasses and all subclasses of the current form that are available to be used as a unique accelerator
Tab Ordering	Changes the mode to allow input of tab indexes

Add Control Command

Use the **Add Control** command from the Controls menu to add a new control to your current form. The **Add Control Type** combo box is then displayed, to enable you to select the type of control that you want to add. For details, see "Adding Controls to Your Form", earlier in this chapter.

Delete Control Command

Use the **Delete Control** command from the Controls menu to delete controls from your painted form. This command is disabled if the selected control is inherited.

Before using the **Delete Control** command, you must select the control or controls that you want to delete. The selected controls are highlighted.

To delete a selected control or group of controls

1. Click a control to select it.

   Alternatively, if you want to delete a group of controls, hold down the Shift key and click the other controls that you want to include in the group.

2. Select the **Delete Control** command from the Controls menu.

If a selected control is not referenced in any Jade methods, it is unconditionally deleted.

If a selected control is referenced by one or more Jade methods, a message box is displayed. Click the **Yes** button to confirm the deletion of the current control. Alternatively, click the **No** button to cancel the deletion request.

Change Type Command

Use the **Change Type** command from the Controls menu to change the type of an existing control (for example, to change a **GroupBox** control that does not have children to a **Button** control). For details, see "Changing a Control Type", earlier in this chapter.

Select Parent Command

Use the **Select Parent** command from the Controls menu to select the parent of the currently selected control. This command is enabled only when the currently selected control (the one displayed with sizing handles) is a child of another control.

When you select this command, the "current control" status is moved from the child control to the parent; that is, the child becomes unselected.

The selected status of other controls remains unchanged.

Select All Children Command

Use the **Select All Children** command from the Controls menu to deselect all currently selected controls and select all immediate children of the current form or control; for example, to move a block of controls to another parent, reposition a block of controls, or change properties on a group of controls.

This command is enabled only if the currently selected form or control has children.

If no control is selected, all immediate children of the form are selected.

Applies to Version: 2018.0.01 and higher

Select All Siblings Command

Use the **Select All Siblings** command from the Controls menu to deselect all currently selected controls and select all immediate children of the parent control; for example, to move a block of controls to another parent, reposition a block of controls, or change properties on a group of controls.

This command is enabled only if a control is currently selected and the parent of the control has children other than the currently selected control.

Applies to Version: 2018.0.01 and higher

Push to Bottom Command

Use the **Push to Bottom** command from the Controls menu to push the selected control behind the control or controls that it overlaps. (For the reverse of the **Push to Bottom** command, see the "Bring to Top" command.)

» To push the selected control behind controls that it overlaps, perform one of the following actions

- Select the **Push to Bottom** command from the Controls menu.
- Click the **Push to Bottom** palette button.

The selected control is then pushed behind the other overlapping controls. See also "Pushing a Control to the Bottom", earlier in this chapter.

Push all Parents and Control to Bottom Command

Use the **Push all Parents and Control to Bottom** command from the Controls menu to push the selected control and all of its parents to the bottom of the displayed zOrder. (The **Push to Bottom** command affects only the zOrder of the selected control.)

For the reverse of the **Push all Parents and Control to Bottom** command, see the **Bring all Parents and Control to Top** command.

To push the selected control and all of its parents to the bottom of the zOrder

- Select the **Push all Parents and Control to Bottom** command from the Controls menu.

The selected control and all of its parents are then pushed behind the other overlapping controls. See also "Pushing a Control to the Bottom", earlier in this chapter.

Applies to Version: 2020.0.01 and higher

Bring to Top Command

Use the **Bring to Top** command from the Controls menu to bring the selected control to the top of the control or controls that overlap it. (For the reverse of the **Bring to Top** command, see the "Push to Bottom" command.)

To place the selected control on top of controls that overlap it, perform one of the following actions

- Select the **Bring to Top** command from the Controls menu.
- Click the **Bring to top** palette button.

The selected control is then positioned on top of controls that overlapped it. See also "Bringing a Control to the Top", earlier in this chapter.

Bring all Parents and Control to Top Command

Use the **Bring all Parents and Control to Top** command from the Controls menu to bring the selected control and all of its parents to the top of the displayed zOrder. (The **Bring to Top** command affects only the zOrder of the selected control.)

For the reverse of the **Bring all Parents and Control to Top** command, see the **Push all Parents and Control to Bottom** command.

To place the selected control and all of its parent to the top of the zOrder

- Select the **Bring all Parents and Control to Top** command from the Controls menu.

The selected control and all of its parents are then positioned on top of controls that overlapped it. See also "Bringing a Control to the Top", earlier in this chapter.

Applies to Version: 2020.0.01 and higher

Check For Duplicate Control Accelerators Command

Use the **Check For Duplicate Control Accelerators** command from the Controls menu to check whether there are any duplicated control caption accelerators in the full hierarchy of the currently selected form. For details, see "Finding the Available Control Caption Accelerator Characters", earlier in this chapter.

Applies to Version: 2020.0.01 and higher

Find Available Caption Accelerator Characters Command

Use the **Find Available Caption Accelerator Characters** command from the Controls menu to show the characters in a control caption and all superclasses and all subclasses of the current form that are available to be used as a unique accelerator. For details, see "Finding the Available Control Caption Accelerator Characters", earlier in this chapter.

Applies to Version: 2020.0.01 and higher

Tab Ordering Command

Use the **Tab Ordering** command from the Controls menu to change the tab sequence that the user experiences at run time. For details, see "Changing the Runtime Tab Sequence", earlier in this chapter.

Layout Menu

Use the commands in the Layout menu to fine-tune the sizing and placement of controls on your painted form. For example, if you have painted a row of buttons, you may want to space them neatly on your form and make them all the same size.

The layout commands normally operate only on controls that are in the same container. The exception to this is that you can apply layout commands across print frames.

The Layout menu commands are listed in the following table.

Command	Description
Alignment	Displays the submenu of alignment options
Spacing	Displays the Spacing submenu of options
Size	Displays the Sizing submenu of options
Suspend Parent Alignments	The **alignContainer** and **alignChildren** properties of controls are treated as though the property values are zero (**0**) so that no automatic alignment occurs

See also "Defining the Layout of Your Form" and "Examples of Laying Out Controls", earlier in this chapter.

Alignment Command

Use the **Alignment** command from the Layout menu to access a submenu of commands that enable you to align selected groups of controls. For details, see "Alignment Controls", earlier in this chapter.

For details about aligning docking controls, see "Allowing Control Docking", earlier in this chapter and the JadeDockBar and JadeDockBase classes in Chapter 2 of the *Encyclopaedia of Classes*.

For details about positioning docking controls to meet your requirements when the **alignContainer** and **alignChildren** properties are set so that no automatic alignment occurs, see "Suspend Parent Alignments Command", later in this chapter.

To display the alignment submenu

- Select the **Alignment** command from the Layout menu.

The **Alignment** command submenu, providing the commands listed in the following table, is then displayed.

Command	Description
Left	Aligns to the left edge of the master control
Right	Aligns to the right edge of the master control
Top	Aligns to the top edge of the master control
Bottom	Aligns to the bottom edge of the master control
Centre Horizontally	Centers the selected controls horizontally
Centre Vertically	Centers the selected controls vertically

See also "Alignment Controls" under "Defining the Layout of Your Form", and "Examples of Laying Out Controls", earlier in this chapter.

When laying out your form, use the Size subcommands before using the **Alignment** and Spacing layout commands, as the **Size** subcommands can alter the spacing of your controls.

Left Command

Use the **Left** command from the Alignment menu to align a group of controls so that the left edges of the selected controls are aligned.

» To select a group of controls and assign a master control

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that are to be in your group.

The last control that you select becomes the master control.

» To align the left edges of your selected group of controls, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Left** command from the Alignment menu.
- Click the **Align left** toolbar button.

Right Command

Use the **Right** command from the Alignment menu to align a group of controls so that the right edges of the selected controls are aligned.

» To select a group of controls and assign a master control

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that are to be in your group

The last control that you select becomes the master control.

» To align the right edges of the selected controls, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Right** command from the Alignment menu.
- Click the **Align right** toolbar button.

Top Command

Use the **Top** command from the Alignment menu to align a group of controls so that the top edges of the selected controls are aligned.

To select a group of controls and assign a master control

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that are to be in your group

The last control that you select becomes the master control.

To align the top edges of the selected controls, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Top** command from the Alignment menu.
- Click the **Align top** toolbar button.

Bottom Command

Use the **Bottom** command from the Alignment menu to align a group of controls so that the bottom edges of the selected controls are aligned.

To select a group of controls and assign a master control

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that are to be in your group.

The last control that you select becomes the master control.

To align the bottom edges of the selected controls, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Bottom** command from the Alignment menu.
- Click the **Align bottom** toolbar button.

Centre Horizontally Command

Use the **Centre Horizontally** command from the Alignment menu to center a control or group of controls relative to the horizontal axis of your form.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that are to be in your group

To center the selected controls horizontally, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Centre Horizontally** command from the Alignment menu.
- Click the **Centre Horizontally** toolbar button.

The selected controls do not move relative to each other. If you select one control only, it centers only that control.

Centre Vertically Command

Use the **Centre Vertically** command from the Alignment menu to center a control or group of controls relative to the vertical axis of your form.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that are to be in your group

To center the selected controls vertically, perform one of the following actions

- Select the **Alignment** command from the Layout menu and then select the **Centre Vertically** command from the Alignment menu.
- Click the **Centre Vertically** toolbar button.

The selected controls do not move relative to each other. If you select one control only, it centers only that control.

Spacing Command

Use the **Spacing** command from the Layout menu to access a submenu of commands that enable you to neatly and logically space your controls across or down the form or container control. (For details, see "Spacing Your Controls" and "Examples of Laying Out Controls", earlier in this chapter.)

To display the Spacing submenu

- Select the **Spacing** command from the Layout menu.

The **Spacing** command submenu, providing the commands listed in the following table, is then displayed.

Command	Description
Space Evenly Down	Spaces the selected controls evenly down the container
Space Evenly Across	Spaces the selected controls evenly across the container
Spread Down Container	Spreads the selected controls evenly down the entire container
Spread Across Container	Spreads the selected controls evenly across the entire container
Vertically Adjacent	Makes the selected controls vertically adjacent
Horizontally Adjacent	Makes the selected controls horizontally adjacent
Vertical Standard Spacing	Automatically spaces controls vertically
Horizontal Standard Spacing	Automatically spaces controls horizontally
Top/Right Command Buttons	Positions command buttons at top right
Bottom/Right Command Buttons	Positions command buttons at bottom right

See also "Spacing Your Controls" under "Defining the Layout of Your Form", and "Examples of Laying Out Controls", elsewhere in this chapter.

When laying out your form, use the **Size** subcommands before using the **Alignment** and **Spacing** layout commands, as the **Size** subcommands can alter the spacing of your controls.

Space Evenly Down Command

Use the **Space Evenly Down** command from the Spacing menu to space a group of controls so that the vertical distance between each control is the same.

The **Space Evenly Down** command differs from the **Spread Down Container** command in that it spreads the controls evenly between the highest and lowest of the controls you have selected only. The **Spread Down Container** command spreads the controls evenly down the whole form or container control.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

Although you must select at least three controls to use this command, the order of selection is not important.

To space the controls evenly down the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Space Evenly Down** command from the Spacing menu.
- Click the **Space Evenly Down** toolbar button.

Space Evenly Across Command

Use the **Space Evenly Across** command from the Spacing menu to space a group of controls so that the horizontal distance between each control is the same.

The **Space Evenly Across** command differs from the **Spread Across Container** command in that it spreads the controls evenly between the farthest left and farthest right of the controls you have selected only. The **Spread Across Container** command spaces evenly across the whole form or container control.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

Although you must select at least three controls to use this command, the order of selection is not important.

To space the controls evenly across the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Space Evenly Across** command from the Spacing menu.
- Click the **Space Evenly Across** toolbar button.

Spread Down Container Command

Use the **Spread Down Container** command from the Spacing menu to space a group of controls down a form or container control so that each control has an equal vertical space adjacent to it.

The **Spread Down Container** command differs from the **Space Evenly Down** command in that it spreads the controls evenly down the whole form or container control. The **Space Evenly Down** command simply spaces evenly between the highest and lowest of the controls that you have selected.

You can apply the **Spread Down Container** command to one control but its action is the same as the **Centre Vertically** command on the Alignment submenu.

To select a group of controls

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

All of the controls that you select in this way are highlighted and form your group.

To spread the controls evenly down the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Spread Down Container** command from the Spacing menu.
- Click the **Spread down container** toolbar button.

Spread Across Container Command

Use the **Spread Across Container** command from the Spacing menu to space a group of controls across a form or container control so that the each control has an equal horizontal space adjacent to it.

The **Spread Across Container** command differs from the **Space Evenly Across** command in that it spreads the controls evenly across the whole form or container control. The **Space Evenly Across** command simply spaces evenly between the farthest left and farthest right of the controls that you have selected.

You can apply the **Spread Across Container** command to one control but its action is the same as the **Centre Horizontally** command on the Alignment submenu.

To select a group of controls

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

All of the controls that you select in this way are highlighted and form your group.

To spread the controls evenly across the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Spread Across Container** command from the Spacing menu.
- Click the **Spread across container** toolbar button.

Vertically Adjacent Command

Use the **Vertically Adjacent** command from the Spacing menu to attach controls vertically. The highest control that you select does not move when the **Vertically Adjacent** command is actioned. All of the selected controls but the highest control are moved upward.

Controls that are in different horizontal parts of the form are moved and they become adjacent along a vertical axis.

To select this command from the **Spacing** menu, you must first select a group of two or more controls.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All of the controls that you select in this way are highlighted and form your group.

To make you selected controls vertically adjacent, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Vertically Adjacent** command from the Spacing submenu.
- Click the **Vertically adjacent** toolbar button.

Horizontally Adjacent Command

Use the **Horizontally Adjacent** command from the Spacing menu to attach controls horizontally. The farthest left control selected does not move when the **Horizontally Adjacent** command is actioned. All of the selected controls but the one that is farthest left are moved to the left.

Controls that are in different vertical parts of the form are moved and they become adjacent along a horizontal axis.

To select this command from the **Spacing** menu, you must first select a group of two or more controls.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All of the controls that you select in this way are highlighted and form your group.

To action the Horizontally Adjacent command, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Horizontally Adjacent** command from the Spacing menu.
- Click the **Horizontally adjacent** toolbar button.

Vertical Standard Spacing Command

Use the **Vertical Standard Spacing** command from the Spacing menu to align selected controls vertically with standard spacing between each control.

If you have a requirement to standardize the spacing of your controls, especially button controls, use this command to standardize the spacing between controls that you want positioned vertically down your form. The default standard spacing is four dialog units.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All of the controls that you select in this way are highlighted and form your group. The first control selected becomes the master control.

To align the controls vertically with standard spacing, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Vertical Standard Spacing** command from the Spacing submenu.
- Click the **Vertical standard spacing** toolbar button.

Your selected group of controls is then vertically aligned with the master control and with standard space between each control.

Horizontal Standard Spacing Command

Use the **Horizontal Standard Spacing** command from the Spacing menu to align selected controls horizontally with standard spacing between each control.

If you have a requirement to standardize the spacing of your controls, especially button controls, use this command to standardize the spacing between controls that you want positioned horizontally across your form. The default standard spacing is four dialog units.

To select a group of controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All of the controls that you select in this way are highlighted and form your group. The first control selected becomes the master control.

To align the controls horizontally with standard spacing, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Horizontal Standard Spacing** command from the Spacing submenu.
- Click the **Horizontal standard spacing** toolbar button.

Your selected group of controls is then aligned horizontally with the master control and with standard space between each control.

Top/Right Command Buttons Command

Use the **Top/Right Command Buttons** command from the Spacing menu to standardize the positioning of your command controls in the top right of your form or container.

The standard command buttons are **OK**, **Cancel**, and **Help**. As you would normally group these together, the **Top/Right Command Buttons** command enables you to standardize the position of these controls down the upper right of your form or container.

To select a group of command controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All of the controls that you select in this way are highlighted and form your group.

To place the controls at the top right of the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Top/Right Command Buttons** command from the Spacing submenu.
- Click the **Top/Right Command Buttons** toolbar button.

Your selected controls are then stacked vertically in the upper right of the form or container. The controls are aligned with the margins and are separated from each other by the standard spacing (that is, by four dialog units).

Bottom/Right Command Buttons Command

Use the **Bottom/Right Command Buttons** command from the Spacing menu to standardize the positioning of your command controls in the bottom right of your form or container.

The standard command buttons are **OK**, **Cancel**, and **Help**. As you would normally group these together, the **Bottom/Right Command Buttons** command enables you to standardize the position of these controls across the lower right of your form or container.

To select a group of command controls

1. Click the first control of your proposed group
2. While holding down the Shift key, click the other controls that you want in your group

All the controls selected in this way are highlighted, and form your group.

To place the controls at the bottom right of the form or container, perform one of the following actions

- Select the **Spacing** command from the Layout menu and then select the **Bottom Right Command Buttons** command from the Spacing submenu.
- Click the **Bottom Right Command Buttons** toolbar button.

Your selected controls are then arranged horizontally in the lower right of the form or container. The controls are aligned with the margins and are separated from each other by the standard spacing (that is, by four dialog units).

Size Command

Use the **Size** command from the Layout menu to access a submenu of commands that enable you to standardize the size of your controls. (For details, see "Sizing Your Controls" and "Examples of Laying Out Controls", earlier in this chapter.)

Before aligning and spacing the controls on your form, you may want to make some controls identical in size; for example, button controls. You may also want to standardize the width or the height of other controls.

The **Size** command also enables you to scale a control so that it fits the current contents of the control and enables you to standardize the size of controls.

To display the Size submenu

- Select the **Size** command from the layout menu.

The **Size** command submenu, providing the commands listed in the following table, is then displayed.

Command	Description
Same Width	Makes all selected controls the same width
Same Height	Makes all selected controls the same height
Same Height and Width	Makes all selected controls the same height and width
Standard Size	Resizes all selected controls to their default sizes

See also "Sizing Your Controls" under "Defining the Layout of Your Form", and "Examples of Laying Out Controls", earlier in this chapter.

The **Same Width**, **Same Height**, and **Same Height and Width** subcommands are disabled until you select a group of two or more controls.

When laying out your form, use the **Size** subcommands before using the Alignment and Spacing layout commands, as the **Size** subcommands can alter the spacing of your controls.

Same Width Command

Use the **Same Width** command from the Size menu to standardize the width of two or more selected controls. The controls that you select are assigned a width equal to the selected master control. Width truncation or extension is performed from the right.

To select a group of controls and assign a master control

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

The last control that you select becomes the master control.

To standardize the width of the selected controls, perform one of the following actions

- Select the **Size** command from the Layout menu and then select the **Same Width** command from the Size submenu.
- Click the **Same width** toolbar button.

Same Height Command

Use the **Same Height** command from the Size menu to standardize the height of two or more selected controls.

The controls that you select are assigned a height equal to the selected master control. Height truncation or extension is from the bottom.

To select a group of controls and assign a master control

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

The last control that you select becomes the master control.

To standardize the height of the selected controls, perform one of the following actions

- Select the **Size** command from the Layout menu and then select the **Same Height** command from the Size submenu.
- Click the **Same height** toolbar button.

Same Height and Width Command

Use the **Same Height and Width** command from the Size menu to standardize the size of two or more selected controls. The controls that you select are assigned a height and width equal to the selected master control. Height truncation or extension is from the bottom; width truncation or extension is from the right.

To select a group of controls and assign a master control

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

The last control that you select becomes the master control.

To standardize the height and width of the controls, perform one of the following actions

- Select the **Size** command from the Layout menu and then select the **Same Height and Width** command from the Size submenu.
- Click the **Same height and width** toolbar button.

Standard Size Command

Use the **Standard Size** command from the Size menu to perform the following actions.

- Size controls so that they fit the current contents
- Standardize the size of controls

Note Web form controls that are single-line controls are resized to fit on one line.

The controls that are affected by the **Standard Size** command are listed in the following table.

Control	Height	Width
Command button	14 dialog units	Accommodates the caption
Option button	14 dialog units	Accommodates the caption
Check box	14 dialog units	Accommodates the caption
Combo box	14 dialog units	Unchanged
Label	14 dialog units	Accommodates the caption
Text box (single line)	14 dialog units	Accommodates the caption

The height of the controls listed in this table is the default value. You can change this standard height for your application by using the Application menu **Change** command from in the Application Browser.

To apply the **Standard Size** command, you can select one control or a group of controls.

To select a group of controls

1. Click the first control of your proposed group.
2. While holding down the Shift key, click the other controls that you want in your group.

All of the controls that you select in this way are highlighted and form your group.

To resize all selected controls to their default values, perform one of the following actions

- Select the **Size** command from the Layout menu and then select the **Standard Size** command from the Size submenu.
- Click the **Standard Size** toolbar button.

The control or group of controls is then resized accordingly.

Suspend Parent Alignments Command

If it is difficult positioning docking controls to meet your requirements when the **alignContainer** and **alignChildren** properties are set, use the **Suspend Parent Alignments** command from the Layout menu. The **alignContainer** and **alignChildren** properties of controls are then treated as though the property values are zero (**0**) so that no automatic alignment occurs.

When this command is unchecked (the default), the **alignContainer** and **alignChildren** properties of controls behave as normal.

To suspend the alignment of parents to position controls to meet your requirements

- Select the **Suspend Parent Alignments** command from the Layout menu.

The alignment of parent controls is then suspended and a check mark symbol is displayed at the left of the **Suspend Parent Alignments** command.

Notes If you attempt to save the form when the command is checked, a message box is displayed asking if you want to continue. If you indicate that the save process is to continue, the form is saved using the current control positions that could differ from those that apply when normal alignment operations are in effect. If you do not want to continue, the save process is cancelled.

Suspending alignment applies only to the **alignContainer** and **alignChildren** properties and it does not affect the behavior of the **parentAspect**property or the **StatusLine** control.

To resume normal behavior

- Select the **Suspend Parent Alignments** command from the Layout menu.

The check mark symbol is then removed from the left of the command and the required alignments are applied.

Applies to Version: 2020.0.02 and higher

Options Menu

Use the Options menu from the Jade Painter window to change the options associated with the Painter screen.

The Options menu commands are listed in the following table.

Command	Description
Grid	Provides options to display and use a grid
Edit Control Palette	Edits the buttons on the **Control** palette
Show Alignment Hairs	Displays fine alignment lines
Lock Control Positions	Locks all current control positions so that controls cannot be repositioned
Control Hierarchy on Top	Positions the Hierarchy for Form dialog always on top
Properties on Top	Positions control properties dialog always on top
Border all Controls	Borders all controls for visibility during development
Hide Control Palette	Toggles to hide or display the **Control** palette
Hide Alignment/Size Palette	Toggles to hide or display the Alignment/Size palette
Hide Tools Palette	Toggles to hide or display the Tools toolbar
Hide Status Bar	Toggles to hide or display the status bar

Grid Command

Use the **Grid** command from the Options menu to control the display of a grid in your Painter window. A grid helps you to position your controls on a form. You can also define the scale of your grid spacing.

To access the Grid Options dialog, perform one of the following actions

- Select the **Grid** command from the Options menu
- Click the **Grid** toolbar button (toggles the grid display on or off only)

When you select the **Grid** command from the Options menu (or you press Alt+O, G), the Grid Options dialog is then displayed. For details about specifying your grid options, see "Using Grids to Position Controls on Forms", earlier in this chapter.

Edit Control Palette Command

Use the **Edit Control Palette** command from the Options menu to reorganize the buttons on your **Control** palette. You can add, remove, or relocate your control buttons.

To edit the Control palette

- Select the **Edit Control Palette** command from the Options menu.

The Control Palette Edit dialog is then displayed. For details, see "Editing the Control Palette", later in this chapter.

Show Alignment Hairs Command

Use the **Show Alignment Hairs** command from the Options menu to control the display of fine alignment lines on your form. When alignment hairs are displayed, a window margin is drawn on your form and alignment lines are displayed when you move or resize controls by dragging them with the mouse.

Note The window margin is not displayed for a web form.

To toggle the display of alignment hairs on your form, perform one of the following actions

- Select the **Show Alignment Hairs** command from the Options menu
- Click the **Alignment Hairs** toolbar button

For details and an example, see "Using Alignment Hairs to Position Controls on Forms", earlier in this chapter.

Lock Control Positions Command

Select the **Lock Control Positions** command from the Options menu to lock all the controls on your form so that you cannot accidentally move them by dragging.

To lock all control positions, perform one of the following actions

- Select the **Lock Control Positions** command from the Options menu
- Click the **Control move lock** toolbar button

Control Hierarchy on Top Command

Select the **Control Hierarchy on Top** command from the Options menu to cause the Hierarchy for Form dialog to always appear on top of (overlay) your Painter window.

Note If you do not select this command, the Painter window may be positioned over the Hierarchy for Form dialog when you click on your Painter window.

To cause the Hierarchy for Form dialog to always overlay the Painter window

- Select the **Control Hierarchy on Top** command from the Options menu. (Alternatively, click the **Stay on top of Painter** icon at the top left of the dialog.)

For details about the Hierarchy for Form dialog, see "Displaying a Hierarchical List of Controls on a Form", earlier in this chapter.

Applies to Version: 2016.0.01 and higher

Properties on Top Command

Select the **Properties on Top** command from the Options menu to cause the Properties dialog to always appear on top of (overlay) your Painter window.

Note If you do not select this command, the Painter window may be positioned over the Properties dialog when you click on your Painter window.

To cause the Properties dialog to always overlay the Painter window

- Select the **Properties on Top** command from the Options menu. (Alternatively, click the **Stay on top of Painter** icon at the top left of the dialog.)

For details about the Properties dialog, see "Maintaining Properties for Your Form or Control", earlier in this chapter.

Border all Controls Command

Select the **Border all Controls** command from the Options menu to cause a border always to be displayed around your controls in Painter.

If you define any control property with a border style of **none** and those controls have the same background color as the form, they are invisible when they are not selected. Selecting the **Border all Controls** command identifies the position of your controls on forms.

This command does not affect the status of your control borders at run time.

To cause the display of a border around all controls

- Select the **Border all Controls** command from the Options menu

Hide Control Palette Command

Select the **Hide Control Palette** command from the Options menu to remove the display of the toolbar that contains the control palette buttons from the Painter window.

As the **Control** palette toolbar is displayed by default when you have an open form, you can use this command to hide the display of the toolbar.

To hide the Control palette toolbar

- Select the **Hide Control Palette** command from the Options menu.

Hide Alignment/Size Palette Command

Select the **Hide Alignment/Size Palette** command from the Options menu to remove the display of the toolbar that contains the alignment and size palette buttons from the Painter window.

As the **Alignment and Size** palette toolbar is displayed by default when you have an open form, you can use this command to hide the display of the palette.

To hide the Alignment/Size palette

- Select the **Hide Alignment/Size Palette** command from the Options menu.

Hide Tools Palette Command

Select the **Hide Tools Palette** command from the Options menu to remove the display of the **Tools** toolbar from the Painter window.

As the **Tools** toolbar is displayed by default when you have an open form, you can use this command to hide the display of the toolbar.

To hide the Tools toolbar

- Select the **Hide Tools Palette** command from the Options menu.

Hide Status Bar Command

Select the **Hide Status Bar** command from the Options menu to remove the display of the status bar from the bottom of the Painter window.

To hide the status bar

- Select the **Hide Status Bar** command from the Options menu.

Window Menu

The Window menu provides the commands listed in the following table that enables you to access the Properties dialog that enables you to maintain the properties associated with your forms or controls and to translate control properties on the current form when the forms management style of the form is **FormsMngmt_Single_Multi (2)**.

The Window menu also displays the forms that are currently open for editing, enabling you to switch between them.

Command	Toolbar Button	Access the …
Show Control Hierarchy Dialog		Hierarchy for Form dialog
Show Properties Dialog		Properties dialog
Translatable Properties		Translatable Property Browser

For details about translating control properties, see "Using the Translatable Property Browser", earlier in this chapter.

Show Control Hierarchy Dialog Command

Use the **Show Control Hierarchy Dialog** command from the Window menu to access the Hierarchy for Form dialog that enables you to display a hierarchical list of all controls painted on the currently active form.

A list box displays a hierarchical list of parent-child relationships of each control painted on the form. Each control is listed alphabetically by control name under its control parent entry. Each entry is displayed as *control-name* (*control-class-name*) '*caption*'; for example:

```
btnEdit (Button) 'Edit'
```

To access the Hierarchy for Form dialog

- Select **Show Control Hierarchy Dialog** from the Window menu.

For details, see "Displaying a Hierarchical List of Controls on a Form", earlier in this chapter.

Show Properties Dialog Command

Use the **Show Properties Dialog** command from the Window menu to access the Properties dialog that enables you to maintain the properties associated with your forms and controls.

The dialog that is displayed depends on whether you selected a form or a control, or on the type of control. There are generally five sheets that for form and control properties.

Most of the properties have default settings and many property values are set automatically when you manipulate your form or controls in Painter or when you use some of the menu commands; for example, commands in the Layout menu and the Options menu.

To access the Properties dialog, perform one of the following actions

- Select **Show Properties Dialog** from the Window menu.
- Click the **Properties** toolbar button.
- Double-click the form that you are editing.
- Press F4.

The Properties dialog is then displayed. For details, see "Maintaining Properties for Your Form or Control", earlier in this chapter.

Help Menu

Use the command in the Help menu from Painter windows to access the standard CUA help option.

The Help menu provides standard options, or commands, throughout the Jade Platform development environment.

The Help menu command is listed in the following table.

Command	Description
Index	Opens the contents page for online help

Index Command

Use the **Index** command from the Help menu to access the Jade online help.

To access the Jade online help, perform one of the following actions

- Select the **Index** command from the Help menu
- Press F1

The first page of the Jade online help directory document is then displayed, which provides a summary of and hyperlinks to all documents in the Jade Platform product information library.

Note Parts of the Jade Platform development environment contains "hotspots" for direct entry to Jade online help; that is, pressing F1 can pass you directly to the relevant documentation for the item that has the focus. When you access the Jade online help, the topic relating to your help request is then displayed.

The Jade Platform Product Information Library document contains the titles, file names, and contents summary of the Jade Platform documentation.

To navigate to the first page of any document to which you have access

- Click the hyperlink in the first column of the table on the page that is first displayed.

For a summary of the information contained in a document

- Click the hyperlink in the second column of the table on the page that is first displayed.

Note Documentation files are not part of the installation process, and must be downloaded and installed from the Jade website or the release medium separately, if required, into the **documentation** folder of your Jade installation directory.

For details about using Jade help files, see "Jade Online Help", in Chapter 2.

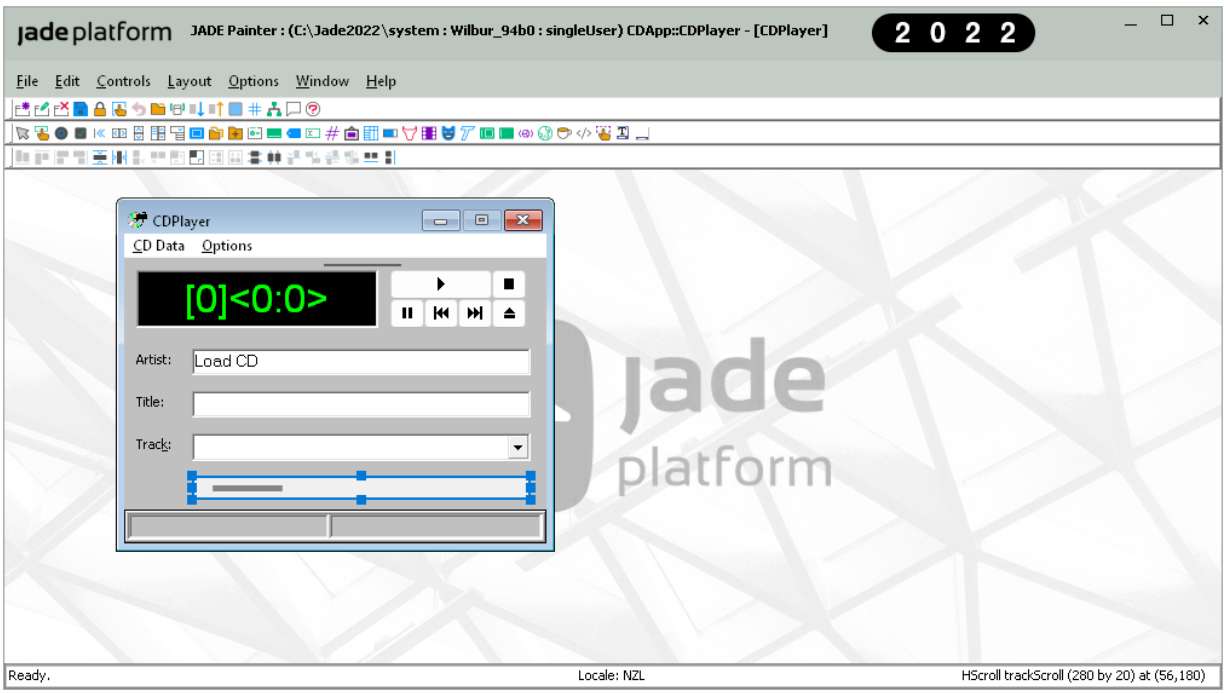
Painter Toolbars

Three rows of buttons can be displayed on the Painter screen. The **Tools** toolbar is displayed by default, to enable you to perform administrative functions.

Tips You can reposition a toolbar vertically or horizontally, by clicking on the grip bar and dragging the toolbar to the required position on the background form or causing it to float in an independent window.

You can change the size of displayed toolbar icons to or from small (shown in the following image), medium, or large. For details, see "Changing the Size of Toolbar Icons", in Chapter 2.

The **Control** and **Alignment and Size** palettes, represented as toolbars, are always hidden when you start a Painter session and they are displayed when you open your first form for edit, as shown in the following image.



These palettes enable you to position, align, size, and otherwise manipulate your controls. When all open forms are closed, the display of the **Control** and **Alignment and Size** palettes is then hidden.

Use the **Hide Tools Palette**, **Hide Control Palette**, **Hide Alignment/Size Palette**, and **Hide Status Bar** commands from the Options menu to toggle the display of toolbars and the status bar.

A check mark displayed to the left of one of these commands indicates that the palette or bar is hidden.

When all toolbars are displayed, the rows of buttons represent (from top to bottom), as follows.

- **Tools** toolbar

  The **Tools** toolbar buttons (the top row of buttons when all three rows are displayed) provide a point-and-click alternative to other tools on the Painter menu; for example, the File menu commands or Options menu commands.

- **Control** palette

  The **Control** palette buttons (the middle row of buttons when all three rows are displayed) enable you to select and place your controls on your form.

- **Alignment and Size** palette

  The **Alignment and Size** palette buttons (the bottom row of buttons when all three rows are displayed) enable you to arrange and size your controls on the form.

Tools Toolbar Button Functions

Use the **Tools** toolbar buttons (the top row of buttons when all three rows are displayed) as a point-and-click alternative to other tools on the Painter menu; for example, the File menu commands or Options menu commands.

The functions of Painter **Tools** toolbar buttons are listed in the following table, with a reference to the sections earlier in this chapter that provide more details.

Button	Action	For details, see…
	Opens a new form	New Form Command
	Opens an existing form for editing	Edit Form Command
	Closes the current form	Close Form Command
	Saves the current form	Save Form Command
	Locks all current control positions	Lock Control Positions Command
	Accesses the Properties dialog	Show Properties Dialog Command
	Undoes the last layout command	Undo Last Layout Command
	Changes the mode to allow input of tab indexes	Tab Ordering Command
	Accesses the Menu Design facility	Menu Design Command
	Pushes the selected control behind the control that it overlaps	Push to Bottom Command
	Makes the selected control sit on top of the overlapping controls	Bring to Top Command
	Toggles (displays or hides) the grid	Grid Command
	Toggles (displays or hides) alignment hairs	Show Alignment Hairs Command
	Displays the Class Browse with the class for the current form selected	Browser Command
	Accesses context-sensitive online help for a specific element	Help Menu
	Opens the contents page for Painter online help	Index Command

Control Palette Button Functions

The middle row of buttons when all three palette rows are displayed provides the **Control** palette buttons listed in the following table, to enable you to select and place your controls on your form.

Button	Description	For details, see…
	Standard paint mode, instead of insert mode	Adding Controls to Your Form
	Inserts the button control	Button Class

Button	Description	For details, see…
	Inserts the option button control	OptionButton Class
	Inserts the check box control	CheckBox Class
	Inserts the browse button control	BrowseButtons Class
	Inserts the horizontal scrollbar control	ScrollBar Class
	Inserts the vertical scrollbar control	ScrollBar Class
	Inserts the list box control	ListBox Class
	Inserts the combo box control	ComboBox Class
	Inserts the frame control	Frame Class
	Inserts the folder control	Folder Class
	Inserts the sheet control	Sheet Class
	Inserts the group box control	GroupBox Class
	Inserts the status line control	StatusLine Class
	Inserts the label control	Label Class
	Inserts the text box control	TextBox Class
	Inserts the edit mask control	JadeEditMask Class
	Inserts the picture control	Picture Class
	Inserts the table control	Table Class
	Inserts the progress bar control	ProgressBar Class
	Inserts the OLE control	OleControl Class
	Inserts the multimedia control	MultiMedia Class
	Inserts the mask control	JadeMask Class
	Inserts the rich text format control	JadeRichText Class
	Inserts the dock container control	JadeDockContainer Class
	Inserts the dock bar control	JadeDockBar Class

Button	Description	For details, see…
	Inserts the Jade XAML control	JadeXamlControl Class
	Inserts the web insert control	WebInsert Class
	Inserts the web Java applet control	WebJavaApplet Class
	Inserts the web HTML control	WebHTML Class
	Inserts the web hotspot control	WebHotSpot Class
	Inserts the Jade text edit control	JadeTextEdit Class
	Inserts the text editor control	JadeEditor Class

The **Mode** button, listed first in this table, is documented under "Adding Controls to Your Form", earlier in this chapter. For details about all controls listed in this table, see the appropriate control class in Chapter 2 of the *Encyclopaedia of Classes*.

As each schema in your database can contain different controls, the form that currently has focus determines the control palette that is displayed in the Jade Painter.

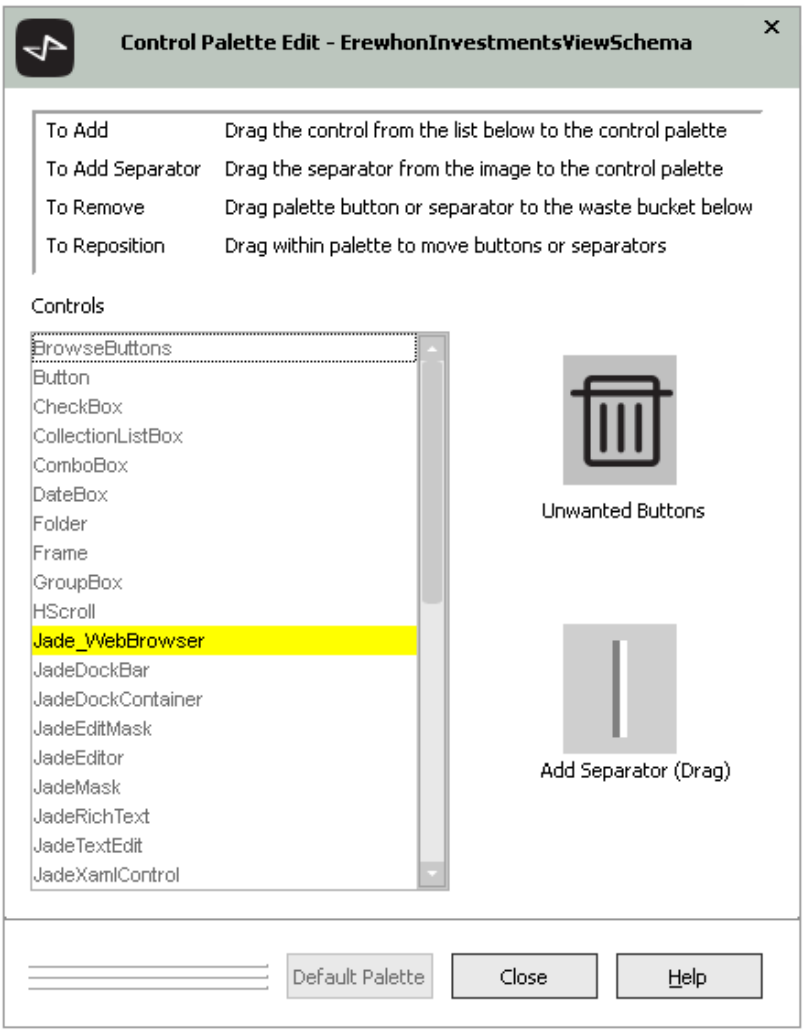
If you have a form for three different schemas currently open in the Painter, the **Control** palette may be displayed differently when each form has focus.

Editing the Control Palette

To edit the Control palette

- Select the **Edit Control Palette** command from the Options menu.

The Control Palette Edit dialog is then displayed, as shown in the following image.



As each schema in the database can contain different controls, the name of the schema that contains the form that currently has focus is displayed in the title bar of the dialog, to enable you to edit the controls that are defined for that schema.

Controls displayed in the **Controls** list box that already positioned on the **Control** palette are disabled.

Any controls not yet added to the **Control** palette that are displayed in the **Controls** list box are enabled and have a background color of yellow.

To add a control to the Control palette

- Drag the required control from the **Controls** list box to the **Control** palette.

  You can drag only entries that are not already added to the palette.

An icon for the control is then displayed on the toolbar at the position to which you dragged it. The entry in the **Controls** list box is disabled and the yellow background is removed.

To add a separator to the control palette

1. Click the separator image on the Control Palette Edit dialog.
2. Drag it to the required position on the **Control** palette.

To remove a separator or control icon from the Control palette

- Drag the palette item to the **Unwanted Buttons** image on the Control Palette Edit dialog.

A control dragged from the **Control** palette in this way is then enabled in the **Controls** list box and displayed with a yellow background.

Bubble help is disabled when dragging the toolbar icon to the Control Palette Edit dialog.

To reposition the icons or separators within the Control palette

- Drag a control icon or a separator to a new location on the **Control** palette.

Any changes that you make to the palette, including adding separators, are stored in your user profile for the selected schema when you click the **Close** button on the Control Palette Edit dialog.

To return the **Control** palette to its default state, click the **Default Palette** button.

Alignment and Size Palette Button Functions

Use the **Alignment and Size** palette buttons (the bottom row of buttons when all three rows are displayed) to arrange your controls on the form. These buttons enable you to align and size your controls. The Painter **Alignment and Size** palette buttons are listed in the following table, with a reference to the sections earlier in this chapter that provide more details.

Button	Description	For details, see…
	Aligns selected controls to the bottom edge of master control	Bottom Command
	Aligns selected controls to the top edge of master control	Top Command
	Aligns selected controls to the left edge of master control	Left Command
	Aligns selected controls to the right edge of master control	Right Command
	Centers the selected controls vertically	Centre Vertically Command
	Centers the selected controls horizontally	Centre Horizontally Command
	Makes all selected controls the same width	Same Width Command
	Makes all selected controls the same height	Same Height Command
	Makes all selected controls the same height and width	Same Height and Width Command
	Resizes all selected controls to the dialog units of the application	Standard Size Command

Button	Description	For details, see…
	Spaces selected controls evenly down the container	Space Evenly Down Command
	Spaces selected controls evenly across the container	Space Evenly Across Command
	Spreads selected controls evenly down the entire container	Spread Down Container Command
	Spreads selected controls evenly across the entire container	Spread Across Container Command
	Makes selected controls vertically adjacent	Vertically Adjacent Command
	Makes selected controls horizontally adjacent	Horizontally Adjacent Command
	Automatically spaces controls vertically	Vertical Standard Spacing Command
	Automatically spaces controls horizontally	Horizontal Standard Spacing Command
	Positions command buttons at bottom right	Bottom/Right Command Buttons Command
	Positions command buttons at top right	Top/Right Command Buttons Command

Most of the alignment and size buttons action a group of controls. In some cases, alignment or sizing is relative to a master control (which has sizing handles).

Using Form Wizard to Create a Form

Form Wizard enables you to create a form quickly and easily. The Form Wizard helps you to create file maintenance-type forms, where the focus is on the instances of one specific class. A form that you create with Form Wizard is complete in its own right, and can be executed immediately.

The painting of controls and the coding of simple methods associated with those controls are produced automatically. Form Wizard obtains the necessary information to create a form by prompting you for that information. When all required data has been gathered, a screen layout (form) is generated, along with supporting methods.

The methods created by Form Wizard are simple and well-documented, to enable you to easily customize the methods or to expand the form to be able to access multiple classes.

Note You cannot use Form Wizard to maintain a previously created form, whether that form was created with Form Wizard or not.

What Form Wizard Produces

The form that is automatically generated by Form Wizard varies, depending on the input that you supply to a series of question-and-answer dialogs provided by Form Wizard.

For details, see the following subsections.

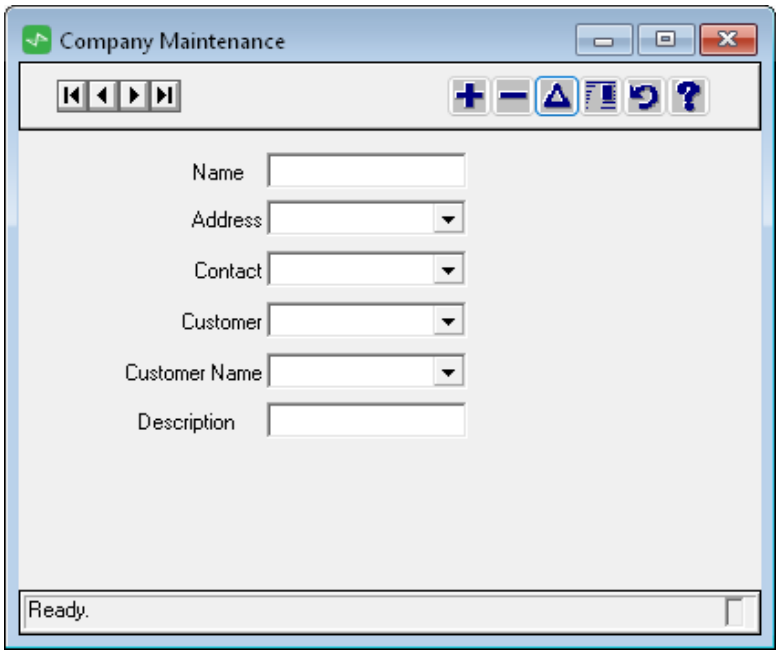
Form Functions

The functions (capabilities) that you expect of a form (that is, browsing, searching, or updating) are optional features that you specify during the Form Wizard question-and-answer phase. The optional functions that you can specify for your form are listed in the following table.

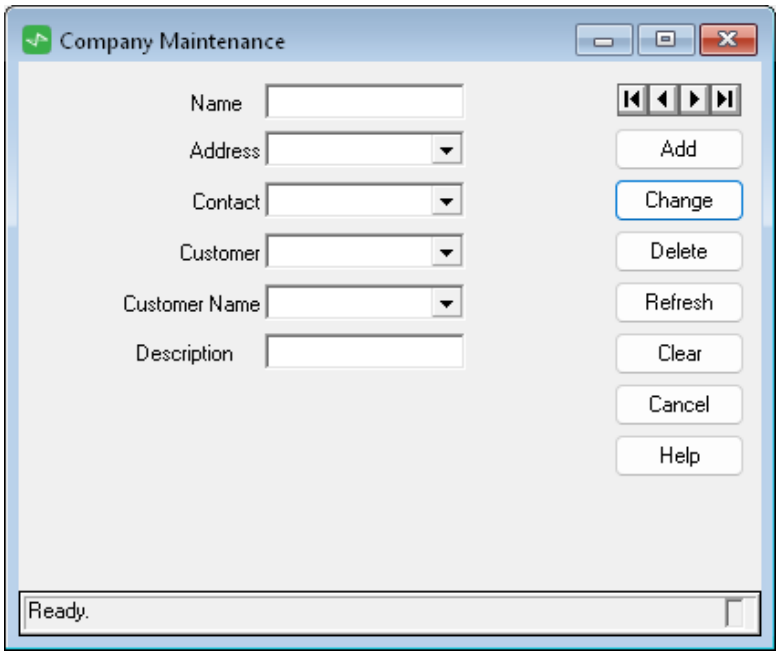
Function	Description
Update	Enables you to read, create, delete, or change instances using the form.
Browse	Enables you to browse instances both forwards and backwards. A set of browse buttons is automatically ascribed to the form if you select browse capability.
Find	Provides your form with search capability.

Form Layout

You can choose to have your function buttons on a toolbar, as shown in the following image.



If you do not want a toolbar on your form, any function buttons are displayed in the body of the form, as shown in the following image.



The display of the menu bar is optional.

If you choose to display multiple instances on one screen, your form is laid out in table form. (A toolbar is always produced on a table-type form.)

Using the Generated Form

The form generated by Form Wizard contains buttons that enable you to control the operations of the form at run time.

If you have not specified that the form is to contain a toolbar, these function buttons are contained in the body of the form.

If your function buttons are on the toolbar, the icons listed in the following table represent them.

Button	Function	Description
	Find	Finds an instance
	Add	Adds an instance
	Delete	Deletes an instance
	Change	Changes an instance
	Clear	Clears all form fields (unless the format is **show instances in a table**)

Button	Function	Description
	Refresh	Refreshes the instance on the form
	Help	Displays online help

For more details, see the following subsections.

Find Button

The **Find** button, which is included if you specify the **Find** function, is displayed only when browsing instances of a dictionary. The values in the controls that correspond to the keys of the dictionary are used to position within the dictionary.

If you specify that searches operate on an exact match basis, only an instance whose attributes match the values in the corresponding controls is retrieved. If not, an error is displayed.

If you specified partial-match searches, a "greater than or equal to" test is applied in finding an instance to display.

Add Button

Click the **Add** button, which is included if you specify the **Update** function, to create a new instance whose attributes equal the values of the corresponding controls. The new instance is added to the instances collection and the input fields are cleared ready for the next instance.

When you have added an instance, you can use the **Next** or **Previous** browse button to display the instance that follows or precedes the instance that has been added.

If you specify multiple instance display, a blank row is inserted in the instances table on the form and the new instance is specified in the blank row.

Delete Button

Click the **Delete** button, which is included if you specify the **Update** function, to delete the instance (and remove it from the collection).

After a deletion, the next instance is automatically displayed. If you delete the last instance, the previous instance is displayed. When there are no more instances to delete, the empty form is displayed.

Change Button

Click the **Change** button, which is included if you specify the **Update** function, to update the current instance with any changes made. The transaction is committed at this point and the lock held on the instance is released.

Clear Button

Click the **Clear** button to clear all form fields; for example, before starting to specify a new instance. If an uncommitted change exists, a message is displayed, enabling you to indicate whether those changes should be saved or discarded.

Refresh Button

Click the **Refresh** button to refresh (update) the display of all fields on the form.

Help Button

Click the **Help** button to display any help documentation that you have set up.

Note The Form Wizard does not generate help text.

Browse Buttons

If you have requested the Form Wizard Browse function, a set of browse buttons is automatically set up by Form Wizard, to enable you to browse through the collection of instances. (For more details, see "Using the Generated Form", earlier in this chapter.)

When the form is first loaded, the first instance of the collection is displayed. You can locate other instances by using the browse buttons. If a status line has been included, it displays the relative order of the instance displayed.

If you use the browse buttons after a change has been made to an instance but before a transaction is committed, a warning message is displayed, and you are asked if the changes should be saved or discarded. If you specified the display of multiple instances on a form, browse buttons are not displayed and instances are displayed in a table that automatically includes scroll bars.

Types of Controls

The types of controls that are displayed depend on the features that you specify during the Form Wizard question-and-answer phase. The types of controls are listed in the following table.

Feature Type	Generated Control
Boolean	Check box
Binary	Picture
Class	Combo box
Collection-class	List box
Other primitive types	Text box

The following points apply to the creation and positioning of controls on your form.

- Controls are created in the center of the form, starting at the top and then stacked down the form with a small gap between each control.
- If the number of created controls does not fit into the visible area of the form, a vertical scroll bar is added to the form.
- If you specify a large number of features to be displayed on your form, you may have to use the Painter to relocate the controls for a more aesthetic layout.
- You can specify any number of controls on your form. The limit is controlled by available memory, but is far more than any practical limit.
- Controls are created as read-only when the associated feature is:
  - A method
  - A read-only or protected property

- Type **Binary**, or a collection
- A property at the Auto end of an explicit inverse reference

Methods

The methods created by the Form Wizard support multiuser access to the class instances. The locking strategy in a multiuser environment works on the following basis.

1. An instance is not locked until an update is started.
2. At this point, the edition on the locked instance is compared to the edition of the instance at its initial read (that is, when the form was loaded for display).
3. If the editions are the same, the update continues. If the editions differ, implying that some other user has already updated this instance, the update is rejected.
4. If an update is rejected, a message is displayed informing you of the situation. The display is then refreshed with the updated instance, which you can then update in the normal manner.

Accessing the Form Wizard

To access Form Wizard from Painter

- Select the **Form Wizard** command from the Painter File menu.

Jade Form Wizard is then displayed.

To access Form Wizard from a browser

- Select the **Form Wizard** command from a browser File menu.

A Painter session is started automatically and the Form Wizard is then displayed. Each set of options is displayed on successive sheets of the Form Wizard dialog. Use the **Next >** and **< Back** buttons to navigate forwards and backwards through the sheets.

When you initiate Form Wizard, you access a question-and-answer phase. Questions are displayed on various sheets of the Form Wizard dialog and your answers provide the characteristics of your form. The subjects presented by the question-and-answer sheets enable you to:

- Specify your form name and title
- Specify the target class
- Specify the visible instances
- Select the presentation style for the instances
- Select the functionality of your form
- Select browse functionality for your form
- Select find functionality for your form
- Select the features displayed on your form
- Select caption and control names

- Incorporate a toolbar, menu, and status line
- Initiate the form build

Note Not all sheets may be displayed, depending on the options that you specify.

For details, see the following subsections.

Specifying Your Form Name and Title

When you access the Form Wizard dialog, the first sheet enables you to specify a name and title for your form.

» To specify your form name and title

1. Specify the name for your form in the **Form Name** text box.

   You must specify a name for your form. Forms cannot have the same name as another form or global object (for example, the application name or a predefined Jade class).

   The form name must start with a character and have a maximum length of 60 characters. It can include numbers and underscore characters but it cannot include punctuation or spaces.

   Forms are defined in the Jade database as classes. If you specify the first character of the name as a lowercase character, it is converted to an uppercase character.

2. Specify the title for your form in the **Title Name** text box, if required.

   If you do not specify a title for your form, a title is automatically generated, based on your specified form name.

   The form title defaults to the form name, with a space inserted before each capital letter. For example, if you specify a form name of **CustomerMaintenance** and do not specify a title, it is generated with a form title of **Customer Maintenance**.

3. Click the **Next >** button to affect your selection and display the next sheet.

Specifying the Target Class

The second sheet of the Form Wizard dialog enables you to select the target class; that is, the class whose instances are updated by using this form.

» To select the target class for your form

1. Select the target class from the **Class** combo box. The classes displayed alphabetically in the list box portion of the **Class** combo box represent all the classes defined in your schema.

   Alternatively, you can specify the name of the target class in the text box portion of the **Class** combo box.

2. Click the **Next >** button to affect your selection and display the next sheet.

The target class that you select for your form is the class from which you select features to be displayed on your form for all instances or for instances of a specified collection.

Specifying the Visible Instances

The third sheet of the Form Wizard dialog enables you to specify the instances over which your form operates.

To define the order and range of instances over which your form operates

- If you want your form to operate over all instances, click the **All Instances** option button.
- If you want your form to operate only over the instances of a specific collection, click the **Instances of the collection** option button and specify the required collection in the text box, as shown in the following example.

  ```
  Company.firstInstance.customerList
  ```

  You must specify a valid Jade language expression that specifies the instance of the required collection. The Jade compiler validates this input.

  The members of the collection must be instances of the target class that you specified in the previous sheet.

Click the **Next >** button to affect your selection and display the next sheet.

Caution If you select the **All Instances** option, the **load** method reads all the instances of the selected class and loads them into an object array when you run the form. For a large number of instances, this may take some time.

Selecting the Presentation Style for the Instances

The fourth sheet of the Form Wizard dialog enables you to select how you want the instances presented; that is, one at a time, or multiple instances in a table.

By default, each property of an instance is displayed in its own control.

To specify that your form is to display a number of instances in a table

1. Select the **Show instances in a table** option button.

   Each row of the table represents an instance of the class. Each column represents a property of the instances.

2. Click the **Next >** button to affect your selection and display the next sheet.

Selecting the Functionality of Your Form

The fifth sheet of the Form Wizard dialog enables you to specify if your form is to have read-only functionality or full update functionality.

By default, forms have read-only functionality over the instances of your selected class.

To select update functionality for your form

1. Click the **Update** option button.

   Update functionality provides you with a form that has read, create, delete, and change capability for the instances of your selected class.

2. Click the **Next >** button to affect your selection and display the next sheet.

Selecting Browse Functionality for Your Form

The sixth sheet of the Form Wizard dialog is displayed only if you have selected the display of one instance at a time in the fourth sheet. If you chose to display multiple instances in a table, that table is automatically provided with scroll bars.

This sheet enables you to include a browse facility on your form. If you select the browse facility (the default), your form is provided with a set of browse buttons for browsing the class instances.

To specify that no browse facility is to be provided for your form

1. Uncheck the **Allow Browse** check box. (The **Allow Browse** check box is checked by default.)

   This check box applies only to single instance forms.

2. Click the **Next >** button to affect your selection and display the next sheet.

Selecting Find Functionality for Your Form

The seventh sheet of the Form Wizard dialog is displayed only if the following conditions are true.

1. You specified that you wanted instances of a specific collection.
2. The collection you specified is a member key dictionary.
3. You selected the display of one instance at a time.

This sheet enables you to include a random-access find facility on your form. You can also specify that you want partial key searching. By default, partial-key searches are performed. With a partial key search, the instance that is equal to or greater than the specified entry is returned when the user enters a partial key.

To enable find capability

1. Ensure that the **Allow Find** check box is checked. (This check box is checked by default.)
2. If you do not want random-key searches performed on a partial key entry, select the **Exact match** option button.

   The **Exact match** option requires that the user specify the complete key (an exact match) when using the search facility to find a specific instance.

3. Click the **Next >** button to affect your selection and display the next sheet.

Selecting the Features Displayed on Your Form

The eighth sheet of the Form Wizard dialog enables you to select the features that are to be displayed on your form. The list of displayed features that you can select includes the following.

- All properties, both public and read-only.
- Accessor methods of the target class and its superclasses. (Accessor methods are non-updating methods that return a value and take no parameters.)

The **Feature List** list box contains all of the properties and accessor methods that are associated with your class selection. From this list, select the features that are to be displayed on your form.

The **Selected features** list box contains the features that you have selected to be displayed on your form. The sequence of the features in this list box becomes the sequence in which the selected features are displayed on your form.

To select the features that are displayed on your form

- Double-click a feature in the **Feature List** list box.
- Select one or more of the features listed in the **Feature List** list box, and then click the single right-arrow (**>**)

button.

- To select the display of all features listed in the **Feature List** list box, click the double right-arrow (**>>**) button.

The selected features are then automatically copied to the **Selected features** list box.

You can select a feature once only. An attempt to select a feature that is already selected has no effect.

Features are added to the **Selected features** list box in the order in which you select them. The exception to this is when dealing with instances in a member key dictionary, when the properties that are keys are always positioned at the top of the list in the **Selected Features** list box.

Your selections are not finalized until you click the **Next >** button. Before you click the **Next >** button, you can reverse any of your selections by removing them from the **Selected features** list box.

You can also change the sequence of features in the **Selected features** list box, as this represents the sequence in which they are displayed on your completed form.

Removing Features Selected for Display

» **To remove features from the Selected features list box, perform one of the following actions**

- Double-click a feature in the **Selected Features** list box.
- Select one or more of the features in the **Selected features** list box and then click the single left-arrow (**<**) button.
- To select all features in the **Selected features** list box, click the double left-arrow (**<<**) button.

The selected features are then automatically moved from the **Selected features** list box to the **Features List** list box.

Click the **Next >** button to affect your selection and display the next sheet.

Changing the Sequence of Displayed Features

» **To change the sequence of features in the Selected features list box, perform one of the following actions**

- Select a feature, and then click the up or down arrow button to move the feature up or down the **Selected features** list box.
- Drag a selected feature to the required position in the **Selected features** list box.

The order of the features in the **Selected Features** list box becomes the sequence in which the features are displayed on your completed form.

Click the **Next >** button to affect your selection and display the next sheet.

Selecting Captions and Control Names

The ninth sheet of the Form Wizard dialog enables you to specify captions and names for the controls on your form.

The Form Wizard automatically provides a caption and a control name for each of the features that you selected on the previous sheet. You can accept the automatic names that are provided or you can overwrite them with the captions or names that you require.

The caption is built in the following way.

- The caption is based on the feature name.
- It is assumed that the feature name is lowercase, except at the beginning of new words within the name; for example, **customerAddressLine1**.
- Capitalizing the first letter of the first word and inserting a space between the words then creates the default caption. For example, the caption for **customerAddressLine1** becomes:

```
Customer Address Line1
```

The control name is built in the following way.

- The feature name is prefixed by **ctl_**
- The name is truncated to 19 characters. For example, **customerAddressLine1** becomes:

```
ctl_customerAddress
```

- If the control name causes a conflict with another control, property, or method name, a number preceded by an underscore character (_) is appended to the name in the form. This number is increased by one until a valid unique name is found. For example, the truncation of **customerAddressLine1** would conflict with **customerAddressLine2**, and so on. This is resolved automatically, so that the control name for **customerAddressLine1** becomes:

```
ctl_customerAddress_1
```

- Click the **Next >** button to affect your selection and display the next sheet.

Note The control name is not displayed on this sheet if you have selected the display of instances in a table, as the table itself becomes the only control on your form.

For details about control name prefixes, see "Defining Control Name Prefixes" under "Creating a Form", earlier in this chapter.

Incorporating a Toolbar, Menu, and Status Line

The tenth sheet of the Form Wizard dialog enables you to include one or more of the toolbar, menu, and status line Windows features on your form.

» To display a toolbar, status line, and menu on your form

1. Ensure that the **Show Toolbar** check box is checked. (The **Show Toolbar** check box is checked by default.) When a toolbar is displayed on your form, the buttons that are included in the toolbar depend on the selections that you made in earlier Form Wizard sheets. For details, see "Using the Generated Form" under "What Form Wizard Produces", earlier in this chapter.
2. Ensure that the **Show StatusLine** check box is checked. (The **Show StatusLine** check box is checked by default.)

   The status line is divided into two parts. The first part displays the status, which is most often **Ready**. The second part displays the current instance number of the instance displayed, or when using a dictionary, it displays the number of instances in the dictionary.
3. Check the **Show Menu** check box. (By default, a menu is not displayed on your form.)

When a menu is displayed on your form, standard menus for File, Edit, and Help are displayed on the form. The Edit menu contains submenus for **Cut**, **Copy**, and **Paste** functions, and the Help menu contains the **Index** and **About** submenus.

4. Uncheck the **Generate Code** check box if you do not want code (logic) to support your form at run time generated. (By default, code for supporting a form at run time is generated.)

   Uncheck the **Generate Code** check box if you do not want to generate standard Jade code; for example, you want only the graphical user interface (the form image) and perhaps a few custom-defined methods to support a page on the web. (Forms generated as web pages do not need all of the support logic generated by the standard Form Wizard.)

5. Click the **Next >** button to affect your selection and display the next sheet.

Initiating the Form Build

The eleventh, and final, sheet of the Form Wizard dialog question-and-answer sequence enables you to initiate the building of your form.

Note You can use the **< Back** button to review any of your form selections at any time before you initiate the building of your form.

To start your form build

- Click the **Start Build** button.

The building of your form and the generation of supporting methods is then started.

Monitoring Your Form Build

A progress dialog is displayed while the Form Wizard builds your form.

The stages of the form build process are listed in the following table.

Stage	Description
Build form	A new form is created as if you had used the **New Form** command from the File menu of the Painter and a class is created with the same name as the form.
Create controls	The types of controls created depend upon the types of features that you have selected.
Save form	The new form is saved in the same way as if you had used the **Save Form** command from the File menu of the Painter. The form becomes the *current* form in the Painter and is added to the **recent opens** list of the File menu of Painter.
Generate methods	From 15 through 20 methods are created for the form and one for each selected attribute.
Compile methods	The created methods are compiled.

For details about using the form, see "Using the Generated Form", earlier in this chapter.

Copying a Form into another Locale

When your schema contains two or more locales, you can copy a form from one locale into another locale in the same schema (replacing the form in the target locale). Alternatively, you can use a command file or the Jade Platform development environment to load an extracted form into another locale.

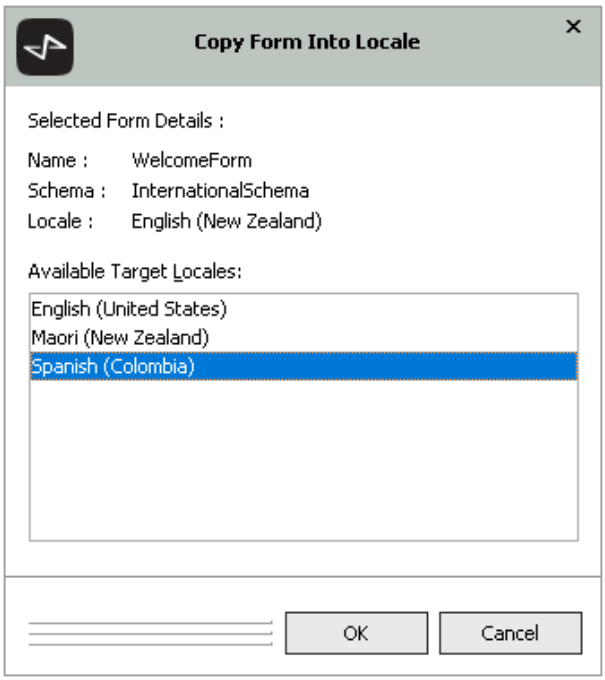
For details about locales, see "Maintaining Locales for Your Schema", in Chapter 11.

To copy a form from one locale and replace a form in another locale in the same schema

1. Select the **Copy Form Into Locale** command from the Painter File menu.

   This command is enabled only when the form has been saved, it is not versioned, and there are one or more other locales in the current schema.

   The Copy Form Into Locale dialog, shown in the following image, is then displayed.



   The locales that are displayed are those in the **Base Locale** combo box on the **Miscellaneous** sheet of the Preferences dialog.

2. In the **Available Target Locales** list box, select the locale into which you want to copy the form that is currently displayed in the Jade Painter.

3. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Chapter 6 Using the Object Inspector

This chapter covers the following topics.

- Using the Quick Inspect Toolbar
- Inspector Security
- Inspecting Object Instances
  - Inspecting Instances of the Current Class
  - Inspecting All Object Instances
  - Inspecting Transient Instances
- Using the Inspector Form
  - Using the File Menu
  - Using the Options Menu
  - Using the History Menu
  - Using the Help Menu
- Inspecting Variable Instances in the Debugger

Using the Quick Inspect Toolbar

The Jade Platform development environment provides the quick inspector toolbar, which:

- Removes the need to write code to inspect an object
- Adds the ability to search and view multiple instances of an entity

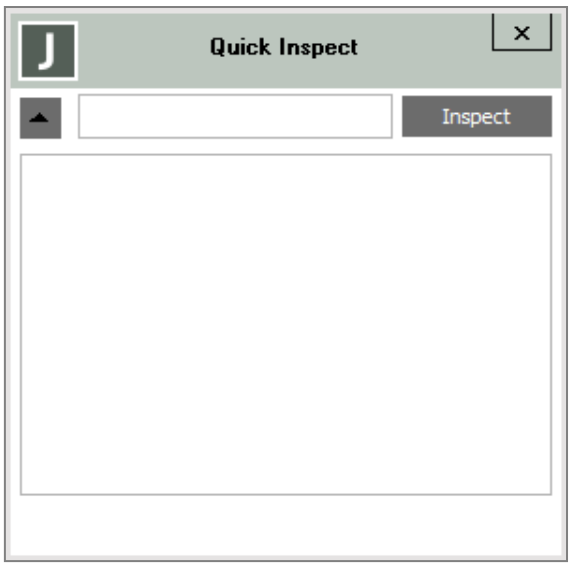
  The valid entries are an unqualified class (where only the current schema will be searched), an existing OID, a class number, or a qualified class name (that is, *schema-name::class-name*, *package-name::class-name*, or *schema-name::package-name::class-name*).
- Can be docked in the development environment main toolbar, and contains:
  - A button (indicated by the down arrow icon) that floats the control.
  - A combo box in which to enter the search entity or select a recently inspected entity. When no text is entered, the combo box displays the current schema.
  - The **Inspect** button that opens the Inspector form with the entity specified in the combo box.

  You can position the docked quick inspect toolbar anywhere in the main toolbar that does not overlap existing control icons (which are always left-aligned).

When the main toolbar is left-aligned or right-aligned, the quick inspect dock bar is hidden and a single toolbar button is displayed. When clicked, it displays the floating Quick Inspect form.

- Can be floated by clicking the float button (indicated by the down arrow icon ∨) or by dragging the toolbar off the main toolbar to display the resizable Quick Inspect form, containing:
  - A button (indicated by the up arrow icon) that docks the control.
  - A text box in which to enter the search entity. When no text is entered, the combo box display the current schema.
  - The **Inspect** button that opens the Inspector with the entity specified in the text box or selected in the list box.
  - A list box that displays search results or previously inspected items for your current session.

  The floating position and size of the Quick Inspect form is retained for the duration of your work session.



When the number of entries in the list box exceeds 1,000, **Next**, **Previous**, **First**, and **Last** buttons are displayed to provide easy navigation over the list.

When the text box is empty, the list box displays previously inspected items. (Historical items are preserved for the current session only.)

If you specify a valid class number, the class name and instances of the class are displayed. If you specify a valid class name, the class number and instances of the class are displayed.

The Jade Platform development environment main toolbar provides a context (popup) menu that contains the **Show Quick Inspect Toolbar** command, which is checked by default. This command is also provided in the development environment View menu. It toggles (hides or shows) the display of the quick inspect toolbar and updates the value of the **Show Quick Inspect Toolbar** check box on the **Window** sheet of the Preferences dialog, which contains the:

- **Show Quick Inspect Toolbar** check box (which is checked by default), to show or hide the display of the quick inspect toolbar. (If the quick inspect toolbar is docked in the toolbar of the main development environment window, hiding the main development environment window toolbar also hides the quick inspect toolbar.)

- **Quick Inspect Width** text box, which enables you to change the width of the quick inspect toolbar combo box from the default value of 300 pixels to the number of pixels you require, with a minimum value of 200 pixels and a maximum of 1,000 pixels.

Tip Use the Ctrl+Alt+I shortcut keys to set focus to the search combo box when the quick inspect toolbar is docked or to the search text box on the Quick Inspect form when the quick inspect toolbar is floating.

If text is selected in more than one editor pane, the latest selected text is pasted into the corresponding search control. If multiple lines are selected, it is cut off at the first **CrLf** end-of-line sequence.

The **Object** class and **Collection** class **inspect** and **inspectModal** methods or the **allowedToInspect** security filter method called on a class enable you to define security to specify users who can inspect class instances in a deployed (run time) database.

See also "Development Function Security Hook", in Chapter 2 of the *Object Manager Guide*.

The development environment Schema Inspector dialog:

- Displays the **Inspect** search text box and button
- The pane at the left of the dialog or form contains a folder containing two sheets: the **History** sheet containing the history hierarchy list box and the **Search** sheet

Note The above list applies only to the development environment Schema Inspector dialog; *not* to the production Schema Collection Inspector form (that is, the Schema Inspector dialog started from the **JadeSchemaInspector** application in the **JadeMonitorSchema**).

The dock or float status and the floating size and position of the Quick Inspect form are saved if you have set your **Save Windows** user preference to **true** (that is, you have checked the **Save Windows** check box on the **Exit** sheet of the Preferences dialog). When you next initiate the Jade Platform development environment, the quick inspect toolbar status is restored.

Applies to Version: 2020.0.01 and higher

Inspector Security

If the Inspector is initiated from a user application, the Inspector is not subject to development environment Inspector security, and no search toolbar is displayed (that is, the Inspector honors development environment Inspector security only if it is initiated from the Jade Platform development environment or the **JadeSchemaInspector** application in the **JadeMonitorSchema**).

In earlier releases, security covering the use of the Inspector form allowed unrestricted access to all instances. To increase inspection security, use one of the following **jadeDevelopmentFunctionSelected** function security hooks.

- The **inspectInstances** security hook task is used by the quick inspect toolbar in both the Jade Platform development environment and the Schema Inspector dialog to control access to the inspection of instances. The **entityName** input parameter enables you to control access to a specific class (*schema-name::class-name*).
- The **inspectMethod** security hook task toggles access to the **Object** class and **Collection** class **inspect** and **inspectModal** methods. The **entityName** input parameter enables you to control access to a specific class (*schema-name::class-name*).

- The **enableInspectToolbar** security hook task is used by the Jade Platform development environment quick inspect toolbar and the Schema Inspector dialog to control access to the quick inspect toolbar itself. The **entityName** input parameter specifies the schema (*schema-name*) and application (*application-name*) that the quick inspect functionality can access. If an **enableInspectToolbar** security hook task fails in the:
  - Jade Platform development environment, the quick inspect toolbar is disabled. If the quick inspect toolbar is already floating when this security hook fails, the floating quick inspect toolbar will be disabled but will still be able to be docked.
  - Schema Inspector, the quick inspect toolbar and the corresponding Option menu **Show Inspect Toolbar** command will not be visible.
- The **searchInstances** security hook task enables you to control the search behavior of the quick inspect toolbar both in the Jade Platform development environment and the Schema Inspector dialog. The **entityName** input parameter specifies the schema (*schema-name*) to which to control access.

When a user fails the security check in the:

- Schema Inspector dialog, the quick inspect toolbar is still displayed, but accepts only valid OIDs.
- Jade Platform development environment, the quick inspect toolbar is still displayed but accepts only valid OIDs and the floating Quick Inspect form does not display a list box.

See also "Development Function Security Hook", in Chapter 2 of the *Object Manager Guide*.

Applies to Version: 2020.0.01 and higher

Inspecting Object Instances

The Inspector form enables you to navigate through your schema, inspecting the database objects; that is, the persistent object instances.

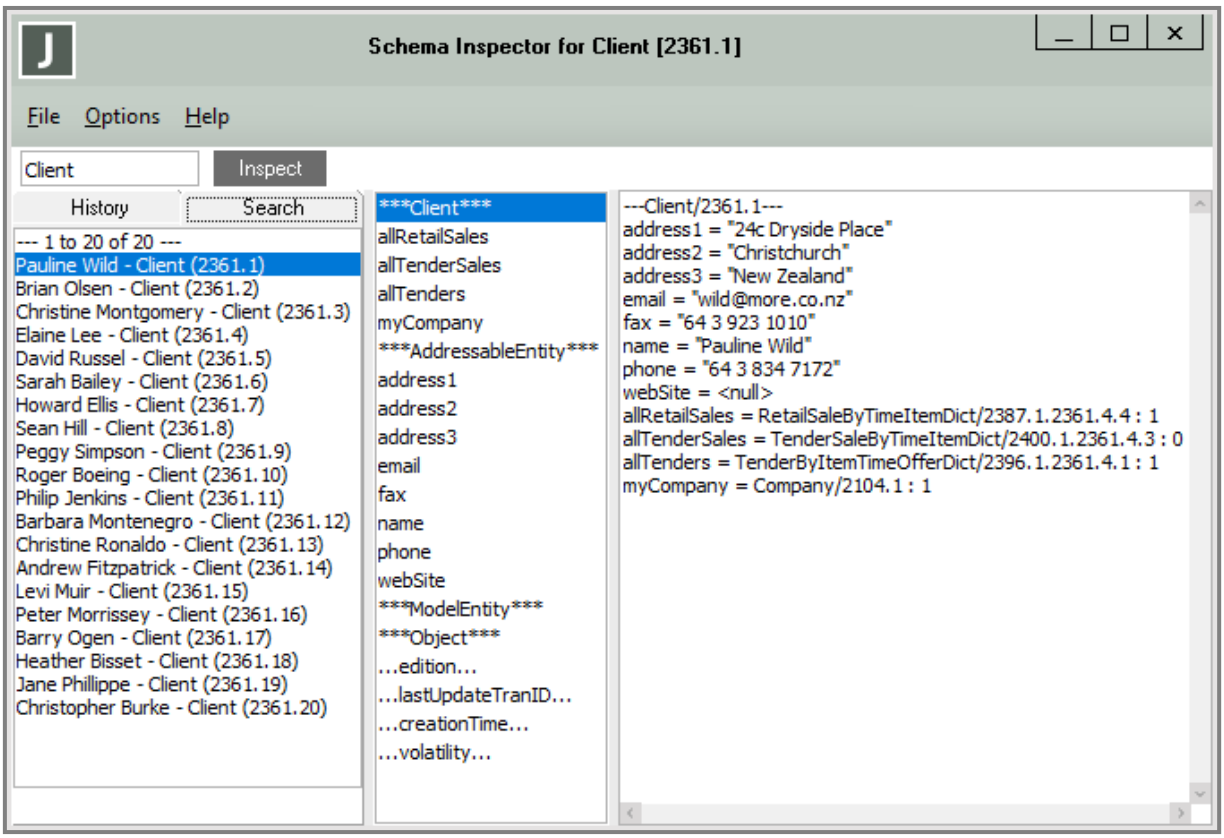
For details about inspecting classes in your deployment databases, see "Inspecting a Deployed Database", in Chapter 1 of the *Schema Inspector Utility Guide*.

Inspecting Instances of the Current Class

» **To display all instances of the selected class, perform one of the following actions**

- Select the **Inspect Instances** command from the Classes menu
- Press Ctrl+I

If the current class contains instances, an Inspector form is opened for the selected class, as shown in the following image.



The title bar of the form for a collection specifies the number of instances of the selected class. The left-hand list contains an entry for each object instance in the selected class.

The Schema Inspector dialog:

- Displays the quick inspect toolbar within the Schema Inspector dialog only if you have checked the **Show Inspect Toolbar** command in the Options menu. (For details, see "Using the Quick Inspect Toolbar", earlier in this chapter.)

  The quick inspect toolbar is hidden if you uncheck the associated command in the Options menu or you do not have security access for this feature.

- Pane at the left of the dialog consists of a folder that has two sheets: the **History** sheet containing the history hierarchy list box and the **Search** sheet.

  If the number of entries in the search list box exceeds 1,000, the results are paginated. When the results are paginated, you can navigate to the next page of results using the right arrow button at the bottom of the list box, or to the last page of results using the arrow button that is farthest right. To navigate to a previous page, click the left arrow, or the arrow button farthest left to return to the first page.

Inspecting All Object Instances

Use the **Inspect All Instances** command from the Classes menu to display all instances of the currently selected class and its subclasses; for example, examine the objects in the database for a class and its subclasses as a result of Jade runtime activities.

» To display all instances of the currently selected class and its subclasses

- Select the **Inspect All Instances** command from the Classes menu

If the selected class or subclasses contain instances, an Inspector form is then opened. The left-hand list contains an entry for each object instance for the selected class and its subclasses.

Inspecting Transient Instances

The Inspector also enables you to inspect shared transient object instances.

From the Class Browser, the Inspector displays all shared transient instances of the currently selected class.

» To display all shared transient instances of the current class, perform one of the following actions

- Select the **Inspect Shared Transients** command from the Classes menu
- Press Ctrl+ J

If the current class contains shared transient instances, an Inspector form is then opened for that class. The left-hand list contains an entry for each shared transient instance for the selected class.

Note You can also inspect shared transient objects from the Schema Browser, by selecting the **Inspect Shared Transients** command from the Schema menu or by pressing Ctrl+ J. The Inspector form then displays all shared transient instances of all classes defined in the current schema.

Using the Inspector Form

» To examine each object for the class

- Select an object in the left-hand list.

The details for the selected object are then displayed at the right of the Inspector form.

» To display the properties of an object

- Double-click on an object in the Object List at the left of the Inspector form.

The specified object and any properties defined for that object are then displayed at the left of the Inspector form.

Notes You can display object properties only if the property is an object reference.

If a dynamic property is a reference, double-clicking the property name opens an inspector for that reference. In addition, dynamic property names are displayed in blue text.

The Schema Collection Inspector title bar specifies the number of instances of the selected class. The list box in the middle pane contains an entry for each instance of the selected class or classes. In addition, if the class is a collection class, a list box at the upper center displays the properties of the collection. Clicking on a:

- Property name displays the property value.
- Collection entry displays the collection entry details.

If the collection class has any properties, both list boxes are automatically made visible. If the collection class does not have any properties, only the collection entries list box display is initially made visible.

When both the collection properties and collection entries list boxes are visible, a resize bar is displayed between them. To change the size of each of the list boxes, drag this resize bar vertically.

Note When a collection is displayed in the middle pane of the Jade Inspector form, clicking on another collection entry preserves the vertical scroll position of the text box that displays the property values on the right, if that position exists.

The Schema Collection Inspector form displays the type, name, and dynamic attribute names and values of **JadeDynamicObject** instances. If a dynamic property is a reference, double-clicking the property name opens an inspector for that reference. In addition, dynamic property names are displayed in dark blue text.

To toggle the display of collection instance properties

- Select the **Show Collection Properties** command in the Options menu.

To examine each object

- Select an object in the middle (object) list.

To inspect a specific object (drill-down)

- Double-click on an object in the Object List in the middle pane of the Schema Inspector form.

The specified object and any properties defined for that object are then displayed in a new Schema Collection Inspector form.

When you select the **Use Same Window** command in the Options menu, each double-click of an object in an Inspector form re-uses the same form to display the selected object, replacing the previously displayed object. A pane at the left of the form contains a hierarchical list box displaying all of the objects that have previously been inspected. The hierarchy indicates the history of how the objects were inspected. The entries display the value of the **name** property if it exists in the object, followed by the class name and the Jade object identifier (oid). Clicking on an entry in the hierarchical history list at the left of the form displays the selected object again.

To view the values for the current property in the object

- Click the required property.

The values for the selected property are then displayed in the pane at the right of the form when the same form is re-used.

Notes If the current property is a reference or a collection and the same form is *not* being re-used, you can double-click on it to open a new Inspector form for that property. This enables you to "drill-down" through your references and collections.

The Call Stack menu is displayed only if call stack information exists for the object.

To inspect an object, click on an object listed in the lower part of the Object List in the center pane, to display details of that object in the pane at the right of the form.

You can inspect the edition, volatility, and time an object was updated or created, by clicking **edition**, **volatility**, **lastUpdateTranId**, or **creationTime**, displayed beneath ***Object*** in the Object List in the center pane, to display that value in the pane on the right of the Inspector Form.

The Schema Inspector form menu bar contains the following menus.

- File
- Options
- Call Stack
- History
- Help

Using the File Menu

Use the File menu in the Schema Inspector form to exit from the deployment inspector. (You can also exit from the Inspector form by clicking the close icon at the top right corner of the form or by selecting the **Exit** command from the Control-Menu.)

To close all open non-modal Inspector forms

- Select the **Close All** command on the File menu.

  Note This command is displayed only when the Schema Inspector form was not shown modally.

All open Schema Inspector forms are then closed.

To open a new Inspector form for the currently displayed object

- Select the **New Window** command on the File menu.

A new Inspector form for the currently displayed object is then displayed (regardless of the setting of the **Use Same Window** command). This enables you to display an object in a separate Schema Inspector form that can be retained regardless of the objects that are inspected in the original form or forms.

To exit from the Inspector form

- Select the **Exit** command on the File menu.

The Schema Inspector form is then closed.

Using the Options Menu

Note If more than one user uses the same Jade initialization file on the client, they will share the same values for parameters in the [JadeInspector] section.

To replace the saved position and size with the default position and size

- Select the **Clear Saved Position/Size** command from the Options menu if you want to use the default position and size for Inspector forms.

When the next Inspector form is opened, the default position and size will be used.

To save the current position and size of the form

- Select the **Save Position/Size** command from the Options menu if you want to save the current position and size of the form.

This information, which is used to position and size any subsequent Inspector forms that are displayed, is stored in the **WindowPos** parameter in the [JadeInspector] section of the Jade initialization file.

To re-use the same form to display the selected object (applies to version 2016.0.01 and higher)

- Select the **Use Same Window** command from the Options menu.

By default, the same Inspector form is not used (that is, this command in unchecked, or unselected), so that each double-click of an object in an Inspector form opens a new Inspector form.

When the **Use Same Window** command is checked, each double-click of an object in an Inspector form re-uses the same form to display the selected object, replacing the previously displayed object. A pane at the left of the form contains a hierarchical list box displaying all of the objects that have previously been inspected. The hierarchy indicates the history of how the objects were inspected. The entries display the value of the **name** property if it exists in the object, followed by the class name and the Jade object identifier (oid). Clicking on an entry in the hierarchical history list at the right of the form displays the selected object again.

The value of this setting (that is, **true** or **false**) is information is stored in the **UseSameWindow** parameter in the [JadeInspector] section of the Jade initialization file.

To specify the font used to display text in the Inspector form

- Select the **Font** command from the Options menu if you want to change the default font from Tahoma, regular, 8.25 points.

The common Font dialog is then displayed, to enable you to make your font selections. When you have selected your required font options, focus is then returned to the Inspector form.

To toggle the display of the collection properties list box

- Select the **Show Collection Properties** command from the Options menu.

The collection properties list box is then displayed or removed from the form and the check mark on the menu item displayed or removed.

To toggle the display of the Inspect toolbar

- Select the **Show Inspect Toolbar** command from the Options menu.

Using the Call Stack Menu

If call stack information exists for a selected object, the Call Stack menu is displayed.

Note To automatically save object create or delete call stack information, set the **DefaultProcessSaveCreateCallStack** or the **DefaultProcessSaveDeleteCallStack** parameter to **true** in the appropriate [JadeClient] or [JadeServer] section of the Jade initialization file.

To view the call stack of an object when it is created

- Select the **Creation** command from the Call Stack menu.

The Call Stack Browser for the selected object is then displayed.

To view call stack of an object when it is deleted

- Select the **Deletion** command from the Call Stack menu.

The Call Stack Browser for the selected object is then displayed.

Applies to Version: 2022.0.03 and higher

Using the History Menu

The History menu displays up to the last 25 objects inspected in the Inspector form, in the reverse order that they were double-clicked.

To redisplay an object instance

- Click an entry in the history list.

The History menu displays a check mark at the left of the object currently displayed in the Inspector form (if it is in the menu item list).

Press the Alt+↑ (up arrow) or Alt+↓ (down arrow) accelerator keys to step to the previous or next entry in the history list, respectively, in the order in which they were originally displayed.

Note The History menu list is not affected by clicking an item in the list or by clicking an item in the hierarchy history list box at the left of the form.

The History menu is displayed only when two or more objects are displayed in the hierarchy list box on the **History** sheet.

Applies to Version: 2016.0.01 and higher

Using the Help Menu

Use the commands in the Inspector form Help menu to access the standard CUA help options. These commands are described in the following subsections.

Index Command

Use the Inspector form Help menu **Index** command to access Jade online help.

To access the online help, perform one of the following actions

- Select the **Index** command from the Help menu
- Press F1

The first page of the Jade online help directory document is then displayed, which provides a summary of and hyperlinks to all documents in the Jade Platform product information library. The Jade Platform Product Information Library document contains the titles, file names, and contents summary of the Jade Platform documentation.

To navigate to the first page of any document to which you have access

- Click the hyperlink in the first column of the table on the page that is first displayed.

For a summary of the information contained in a document

- Click the hyperlink in the second column of the table on the page that is first displayed

Note PDF documentation files are not part of the installation process, and must be downloaded and installed from the Jade website or the release medium separately, if required, into the **documentation** folder of your Jade installation directory.

For details about using Jade help files, see "Jade HTML5 Online Help" or "Jade Platform Product Information Library in Portable Document Format", in Chapter 2.

About Command

Use the Inspector form Help menu **About** command to access information about the Jade Inspector. This information includes the Jade release version and copyright information.

To access the Jade Inspector information

- Select the **About** command from the Help menu

The About Schema Inspector dialog is then displayed. This dialog is for display purposes only.

Inspecting Variable Instances in the Debugger

Use the **Inspect** command from the Debugger Variables menu to display or modify the value of a primitive item or to inspect an item containing an object reference.

» **To display or modify the value of a primitive item or to inspect an item containing an object reference, perform one of the following actions**

- Select the **Inspect** command from the Variables menu
- Click on the **Inspect** toolbar button
- Press Ctrl+I

For details, see "Inspecting and Modifying Values in the Debugger", in Chapter 7. See also "Using the Inspector Form".

Double-click on a method entry in the Jade Debugger Call Stack window to display the Inspector form for the receiver of that method.

Chapter 7 Using the Jade Debugger

This chapter covers the following topics.

- Starting an Application in Debug Mode
  - Displaying Code in the Method Source Window
  - Restrictions
- Using Debugger Menu Commands
- Executing Code in the Debugger
  - Continuing Code Execution
  - Setting the Animate Speed and Toggling the Bubble Help Display
  - Stopping Code Execution in Animate Mode
  - Executing the Current Line of Code
  - Controlling Debugger Execution
- Using Breakpoints in the Debugger
  - Setting or Unsetting Breakpoints
  - Displaying All Breakpoints in the Current Schema
  - Using Conditional Breakpoints and Pass Counts
  - Displaying Source Code for a Breakpoint
  - Removing and Clearing Breakpoints
- Using Watches in the Debugger
  - Selecting a Method in the Call Stack Window
- Displaying the Current Exception Handler Stack
- Viewing Local Variables in the Debugger
- Inspecting and Modifying Values in the Debugger
  - Inspecting an Object Reference
- Terminating a Debugger Session
- Ending a Debugger Session

Starting an Application in Debug Mode

The Jade Platform development environment includes a debugger that enables you to analyze how your application behaves. Debugger features include the following.

- Stepping through code
- Setting breakpoints in code
- Monitoring and changing the values of variables used in the code
- Displaying call stack information

The debugger can be activated when you initiate your Jade application.

Notes To debug all or part of a Workspace, the current method in the **JadeScript** class, or unit test independently of your application, press Shift+F9. (Running a **JadeScript** method always starts a new application copy.) Alternatively, select the Jade menu **Debug** command. (The **Unit Test Debug** command replaces the **Debug** command in the Jade menu when the Jade unit test framework is active.)

To debug a non-GUI application (for example, a server application initiated when the application server is started in Jade thin client mode), you must first change the application type to **GUI, No Forms**, by using the **Application Type** combo box in the **Application** sheet of the Define Application dialog.

Transient methods cannot be debugged, as transient objects created by one process cannot be accessed by another process. If a transient method is executed by an application that is being debugged, the debugger will not display any details of this method; that is, it is not displayed in the call stack, breakpoints cannot be set in a transient method, and variables cannot be inspected.

Inspection of properties from within the Jade debugger (either directly or via the bubble help inspector) will invoke user mapping methods, if defined. However, you should be aware that debugging may fail if the mapping method performs actions not suitable for situations where the execution of the user application is suspended. For example, the creation and display of a form, showing a form modally, display of a message box, deletion of an object in use by the current call stack, and so on.

For details about initiating a Jade unit test in debug mode, see "Running Unit Tests", in Chapter 17 of the *Developer's Reference*.

When debugging a method created from a Workspace, the debug execution initially stops on the first line of execution. Use Jade debugger to set debug breakpoints in the method. You can then step through the logic or debug and run it to meet your requirements.

If you want to add additional debugging code at run time when an application is being debugged, you can call the **Application** class **isBeingDebugged** method to determine whether the application is currently being debugged.

To start an application in debug mode, call the **debugApplication** method, specifying the name of the schema and the application in the respective **schemaName** and **applicationName** parameters. An exception is raised if the Jade Platform development environment is not already running in the same session (using the same **jade exe** environment). Alternatively, to initiate an application in debug mode, passing a specified object to the **initialize** method defined in the application, call the **Application** class **debugApplicationWithParameter** method.

To activate the debugger

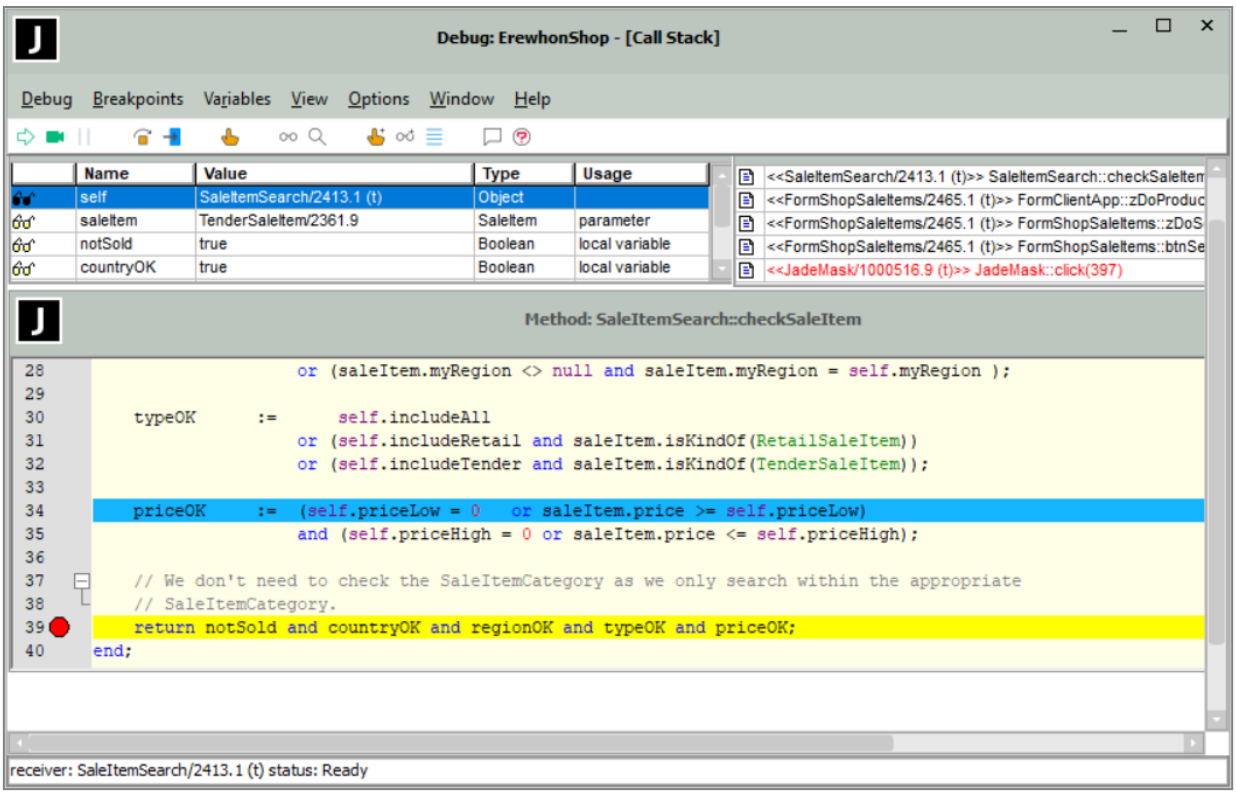
1. Select the **Run Application** toolbar button. (Alternatively, select the Application menu **Run** command from the Application Browser for your selected application.)

   The Run Application dialog is then displayed.

2. Check the **Activate Debugger** check box in the Run Application dialog.

   Alternatively, you can check the **Run in Debug Ready Mode** check box in the Run Application dialog if you want to run the application in debug mode but without the debugger being initiated when this is checked. Attaching to that application then allows debugging of any method already on the execution stack. If the application is idle and no modal dialogs are displayed, attaching the debugger is equivalent to running in debug-ready mode. (For details, see "Attaching the Debugger to a Running Application", in Chapter 1 of the *Runtime Application Guide*.)

The Debugger window, shown in the following image, is then displayed.



Notes By default, the line of the source code that is currently executing is highlighted in blue and the lines on which breakpoints are set are highlighted in yellow. You can change these display colors in the **Editor** sheet from the Preferences dialog (accessed from the Options menu **Preferences** command), by using the **Debug Step Line** or **Debug Line** color option, respectively. In addition, when an enabled breakpoint is set on a line, a red circle is displayed in the breakpoint margin at the left of the method source window and when a disabled breakpoint is set on a line, a white circle is displayed in the breakpoint margin.

The **Save settings on exit** check box in the Debugger Options dialog (accessed by selecting the **Preferences** command from the Options menu) enables you to save the size and position of windows that you resize or reposition.

If you have repositioned or resized windows within the main window and you want to return to the default layout, select the **Restore Default Layout** command in the View menu. This command is disabled if the windows are already in the default layout positions. In the default layout mode, resizing the Jade Debugger window retains the relative default layouts of the three windows. The current left, top, width, and height of the main Jade Debugger window are retained.

The Debugger window contains:

- A menu bar
- A row of toolbar (command) buttons
- A method source window, displaying the source code for the method that is currently executing
- A Local Variables window
- A Call Stack window

To enable you to debug packages and peer schemas, watches and breakpoints are visible across schemas.

Displaying Code in the Method Source Window

In the Method Source window, the next line of code to be executed is highlighted in blue (or the selected color of your choice). Lines of code are displayed as follows.

- An enabled breakpoint set on a line is indicated by a red circle displayed in the breakpoint margin at the left of the method source window. A disabled breakpoint on a line is indicated by a white circle displayed in the breakpoint margin.
- Only the first line is highlighted if an instruction is spread over more than one line.
- Instructions within a loop are highlighted for each iteration of the loop but loop instructions are highlighted once only.
- When you step through code, the highlighted instruction is kept visible by the lines of code scrolling as necessary.
- If the selected line of the method has no compiled byte code, the breakpoint is set on the next source line that has byte code. If there is no source line following with byte code, the breakpoint is set at the beginning of the method.
- Bubble help displays the current value of the variable or object over which the mouse is positioned and you are running the application in standard (fat) client mode.

  The debugger displays the value of a path such as **sc.startDate** in the following cases.

  - If you hover the mouse over an identifier that is part of a path (for example, **startDate** in **sc.startDate**), the bubble help display is **sc.startDate = date**. The value is also displayed if the full path is selected.
  - You can inspect a path in the debugger. If you place the cursor over a variable and then perform an inspect action, the Variable Inspector dialog is displayed with the full path name displayed.

    Notes If a valid path is selected in the editor text, the inspector value display is presented without the Variable Inspector dialog appearing.

    A path that contains a method call cannot be inspected.

  You can turn the display of bubble help on or off, if required. For details, see "Setting the Animate Speed and Toggling the Bubble Help Display", later in this chapter.

Restrictions

When using the Jade debugger, the following restrictions apply.

- If you invoke the debugger during the processing of menu logic, Windows discards any subsequent menu actions. For example, if you break on the **click** event of a popup menu, the menu is not displayed.
- Methods that have no source code are not debugged.
- In multiuser mode, you cannot debug server execution methods.
- Inspection of properties from within the Jade debugger may fail if a user mapping method performs actions not suitable for situations where the execution of the user application is suspended; for example, creation and display of a form, showing a form modally, display of a message box, deletion of an object in use by the current call stack, and so on.

Using Debugger Menu Commands

The debugger menu bar contains the following menus.

- Debug
- Breakpoints
- Variables
- View
- Options
- Window
- Help

Notes Some commands in the debugger menus can also be actioned by selecting the corresponding toolbar button or function key or by right-clicking in a window and selecting the appropriate command from the popup (context) menu that is then displayed.

Visual Studio uses the F9 key to set breakpoints, by default, and uses the F5 key to run and continue debug execution. Jade uses the F5 key to set breakpoints and the F9 key to run and continue debug execution. Switching between the two development environments can lead to confusion and frustration when using the F5 and F9 keys. You can swap the F9 and F5 accelerator key bindings, by checking the **Swap F5 (Toggle Breakpoint) and F9 (Execute/Continue) accelerators** check box on the **Short Cut Keys** sheet on the Preferences dialog. This check box is unchecked by default (that is, false). For details, see "Maintaining Shortcut Keys", in Chapter 2.

Debug Menu

Use the Debug menu to control execution of your code. The Debug menu commands and their resulting actions are listed in the following table.

Command	Function Key	Result
Continue	F9	Continues code execution without interaction with the debugger. The debugger is activated if a breakpoint is encountered.

Command	Function Key	Result
Animate		Continues code execution, displaying and highlighting each line of code as it executes.
Halt Execution		Halts execution of code when in **Animate** mode only.
Step Into	F7	Executes the current line of code. If the next instruction in the current line of code is a method call, the debugger displays the method and enables you to step through that method.
Step Over	F8	Executes the current line of code.
Run To Caret	Ctrl+F8	Executes code up to the line containing the caret.
Run To Return	Shift+F8	Executes code until the current function returns to its caller.
Terminate		Terminates the debugger and your Jade application.
End		Terminates only the debugger.

Breakpoints Menu

Use the Breakpoints menu to maintain breakpoints. The Breakpoints menu commands and their resulting actions are listed in the following table.

Command	Function Key	Result
Toggle	F5	Sets or unsets a breakpoint on the line of code where the caret is positioned
Show All BreakPoints		Displays the Debugger Breakpoints window
Clear All BreakPoints		Resets (clears) all breakpoints in the current schema

You can set breakpoints in any Jade method.

Variables Menu

Use the Variables menu to view and modify values for variables or modify properties for objects. The Variables menu commands and their resulting actions are listed in the following table.

Command	Function Key	Result
Add Watch		Displays the Watches dialog to enable you to specify the name of an item to watch.
Remove Watch		Removes the watch selected in the Watches window.
Clear All Watches		Removes all watches.
Capture Object Call Stacks		Enables or disables the saving of object call stacks.
Inspect	Ctrl+I	Displays the Inspect Variable dialog.

For details about inspecting a variable, see "Inspecting Variable Instances in the Debugger", in Chapter 6.

View Menu

Use the View menu to bring to the front or display the Local Variables, Watches, Call Stack, or Exception Handler Stack window. The View menu commands and their resulting actions are listed in the following table.

Command	Result
Local Variables	Displays the Local Variables window within the Debug window
Watches	Displays the Watches window within the Debug window
Call Stack	Displays the Call Stack window within the Debug window
Restore Default Layout	Returns to the default layout if you have repositioned or resized windows within the main window, and disabled if the windows are already in the default layout positions and sizes
Show Exception Handler Stack	Displays the Exception Handler Stack message box

Options Menu

Use the Options menu to set your animate speed and save the size of position of windows that you resize or reposition. The Options menu command and its resulting action are listed in the following table.

Command	Result
Preferences	Displays the Debugger Options dialog.

For details, see "Setting the Animate Speed and Toggling the Bubble Help Display", elsewhere in this chapter.

Window Menu

Use the Window menu to arrange and manipulate the MDI child windows in the Debugger MDI frame. The Window menu commands and their resulting actions are listed in the following table.

Command	Result
Arrange Icons	Rearranges minimized window icons
Cascade	Cascades windows (the default)
Tile Horizontal	Tiles windows horizontally
Tile Vertical	Tiles windows vertically

The lower portion of the Window menu lists all currently open windows (after these commands) in the order in which they were opened. The current, or active, window is indicated by a check mark to the left of the window number and name. For more details, see "Using Window Menu Commands", in Chapter 2.

Help Menu

Use the commands in the Help menu to access the standard CUA help options. The Help menu commands and their resulting actions are listed in the following table.

Command	Result
Index	Opens the contents of the Jade online help
About Jade Debugger	Displays details about the current version of Jade

For more details, see "Jade Online Help", in Chapter 2.

Executing Code in the Debugger

The modes of code execution, described in the following subsections, are as follows.

- Continuing Code Execution
- Setting the Animate Speed and Toggling the Bubble Help Display
- Stopping Code Execution in Animate Mode
- Executing the Current Line of Code
- Controlling Debugger Execution

Continuing Code Execution

» **To execute code without interaction with the debugger, perform one of the following actions**

- Select the **Continue** command from the Debug menu
- Click the **Continue execution** toolbar button
- Press F9

The debugger is then reactivated if a breakpoint is encountered.

» **To execute code in Animate mode, perform one of the following actions**

- Select the **Animate** command from the Debug menu
- Click the **Execute with animation** toolbar button

The debugger then displays and highlights each line of code in the Method Source window as it is executed.

Setting the Animate Speed and Toggling the Bubble Help Display

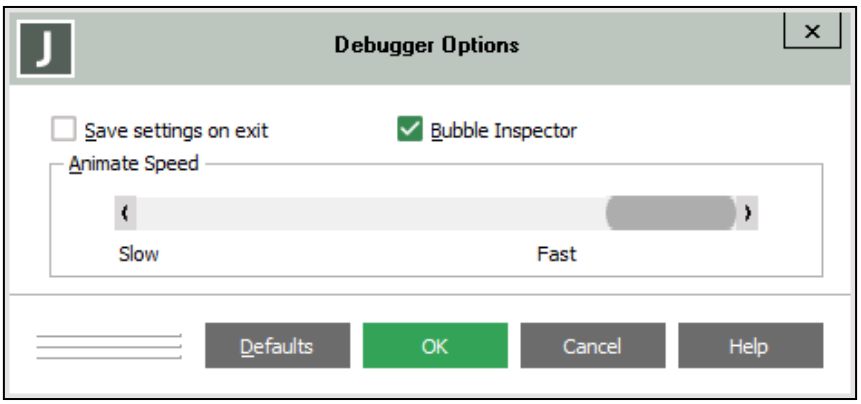
By default, the speed of animate code execution is fast.

You can set the speed that you require for animated code execution.

To set the speed for the Animate mode

1. Select the **Preferences** command from the Options menu.

   The Debugger Options dialog, shown in the following image, is then displayed.



2. If you want your code executed in future debugger sessions at the animate speed that you select, check the **Save settings on exit** check box. By default, the animation speed that you select is not saved when you exit from the debugger.

   The **Save settings on exit** check box also enables you to save the watches that have been set and the size and position of windows that you resize or reposition.

3. Check the **Bubble Inspector** check box if you do not want the value of the variable or object over which the mouse is positioned in the Method Source window displayed in bubble help. This bubble help is displayed by default.

   Tip To optimize performance when stepping through code in thin client mode, it may be desirable to disable bubble help if the connection speed is low.

4. Click the **Defaults** button if you want to restore the settings to the Jade Debugger default values (that is, the animate speed is fast and settings are not saved on exit) and the default debugger window settings. The default values are restored only when you click this button and then the **OK** button.

   If you click the **Cancel** button after you have clicked the **Default** button, the debugger default values are not restored.

5. In the Animate Speed group box, drag the scroll box towards **Slow** or **Fast**, to increase or decrease the animate speed. Alternatively, use the right arrow or left arrow key.

6. Click the **OK** button to set the selected animate speed or to restore the default values.

   Alternatively, click the **Cancel** button to abandon your selection.

Stopping Code Execution in Animate Mode

To halt code execution in Animate mode, perform one of the following actions

- Select the **Halt Execution** command from the Debug menu
- Click the **Halt execution** toolbar button

The code execution is then halted.

Note The **Halt Execution** command is enabled only when code is executing in **Animate** mode.

Executing the Current Line of Code

» **To execute the current line of code, perform one of the following actions**

- Select the **Step Over** command from the Debug menu
- Click the **Step over next statement** toolbar button
- Press F8

The current line of code is then executed.

» **To display and step into a method called during execution of the current line of code, perform one of the following actions**

- Select the **Step Into** command from the Debug menu
- Click the **Step into next statement** toolbar button
- Press F7

If the next instruction in the current line of code is a method call, the debugger displays that method, to enable you to view execution of the method.

Controlling Debugger Execution

The **Run to Caret** and **Run to Return** commands in the Debug menu enable you to specify where and when you want the execution of your code to pause.

» **To execute your code up to the line containing the caret, perform one of the following actions**

- Select **Run to Caret** command from the Debug menu
- Press Ctrl+F8

If the next expression in the current line of code is a method call, the debugger displays that method, to enable you to view execution of the method.

If the caret is not positioned on an expression in a line of source code, execution runs to the first expression following the caret position.

» **To run execution until the current function returns to its caller method, perform one of the following actions**

- Select **Run to Return** command from the Debug menu
- Press Shift+F8

The debugger executes to the line in the calling method in which the method was called, unless a breakpoint is encountered earlier.

Tip Use the **Run To Return** command when you have accidentally single-stepped into an expression that you do not need to debug or when you have determined that the current procedure works to your satisfaction and you no longer want to step slowly through the rest of it.

Using Breakpoints in the Debugger

Breakpoints interrupt execution of your code. The debugger stops before executing a line of code that contains a breakpoint.

Include breakpoints at important points in your code so that you can observe the flow of code or determine the values of variables at a specific point in your code.

Breakpoints are invalid for blank lines, lines that contain only comments, and **end** instruction lines.

Notes You can set breakpoints, including conditional breakpoints, from the Debugger window or from the Jade Platform development environment.

For details about enabling or disabling breakpoints, see "Using Conditional Breakpoints and Pass Counts", later in this chapter.

You can set breakpoints in any Jade method.

After re-compiling a method while using the Jade debugger, the debugger continues to display the old method source under the following circumstances.

- The application is displaying a modal form.
- The debugger next stops on a breakpoint in the same method that was compiled and that method was the last displayed by the debugger.

When the debugger is at a breakpoint:

- If nothing is selected and you hover the mouse over a variable, the value of the variable is displayed when possible (including path expressions). No methods are executed.
- If an expression is selected, hovering over the selection attempts to execute the expression. The expression must be compilable within the context of the method and must return a value.

  If the compile and execution succeed, the returned value is displayed.

  Note It is your responsibility for the effects that the expression may have on the database and execution of the method logic.

In the Breakpoints window:

- When an enabled breakpoint is set on a line, a red circle is displayed in the breakpoint margin.
- When a disabled breakpoint is set on a line, a white circle is displayed in the breakpoint margin.
- Clicking on an empty breakpoint margin for a line sets a breakpoint.
- Clicking on a disabled breakpoint (indicated by a white circle) enables the breakpoint.
- Clicking on an enabled breakpoint (indicated by a red circle) removes the breakpoint.

- Right-clicking in the breakpoint margin displays a breakpoint menu that has the following commands.
  - **Clear All Breakpoints From Method**.
  - **Breakpoint Options**, which initiates the Breakpoint Options dialog that allows you to disable or enable a breakpoint, set a pass count, and set a conditional expression for the breakpoint.
  - **Disable** or **Enable**, which toggles the disabled or enabled status of the current breakpoint.
  - **Remove Breakpoint**.

  The **Disable Breakpoint** or **Enable Breakpoint** and the **Breakpoint Options** commands are disabled if the method has been changed and not compiled or if there is no breakpoint on that line.
- The Breakpoint Options dialog provides a large **Condition** text box that allows for multiple-line input. The dialog can also be resized.

  The **Condition** text box provides AutoComplete functionality.

When the debugger stops on a breakpoint and the Jade Debugger Breakpoints window is displayed, the breakpoint is selected in the Debugger Breakpoints window.

Any selection in the Jade Debugger Breakpoints window is cleared if the halt was for any reason; for example, a step into or step over action.

A breakpoint always shows an icon in the breakpoint column of the user method in the Jade Debugger method source window.

When the application has stopped on a breakpoint and the **Continue** action is performed, the debugger clears the debugger displays, including the last method shown when the execution stack becomes empty.

If a modal form is displayed, the execution stack does not become empty and the debugger displays are not cleared.

If the next breakpoint encountered is in the same method, the debugger retains the currently displayed method.

When the application becomes idle and a modal form is displayed, the debugger clears the debugger displays and the new method source is displayed when a breakpoint is next encountered in that method.

Notes If the method is changed and re-compiled while the debugger has stopped on a breakpoint in that method (or in another method while that method is on the call stack), the interpreter will not reload that method logic until all copies of the method execution have exited from that method.

The debugger continues to display the old source while breakpoints are only encountered in that method until the application execution again becomes idle.

After a breakpoint is encountered in any other method, any breakpoint subsequently encountered in the changed method displays the new source, regardless of whether the old or new version of the method is being executed.

For more details, see the following subsections.

Setting or Unsetting Breakpoints

» To set or unset a breakpoint while in the debugger

- Position the caret on the appropriate line of code in the method source window and then perform one of the following actions.

- Select the **Toggle** command from the Breakpoints menu
- Click the **Toggle Breakpoint** toolbar button
- Press F5
- Press Ctrl+Alt+B

The selected line of code is then highlighted in yellow when a breakpoint is set (or the selected color of your choice). In addition, a red circle displayed in the breakpoint margin at the left of the method source window and the Breakpoints window indicates that a breakpoint is enabled for that line of code, while a white circle indicates that the breakpoint is disabled.

You can also toggle a breakpoint in a line of logic selected in the editor pane of the Class Browser or Primitive Types Browser, by performing one of the following actions.

- Clicking in the breakpoint margin at the left of the line of code in the editor pane adds a breakpoint, indicated by a red circle in the breakpoint margin of the editor pane and the method source window in the Jade Debugger.

  Alternatively, clicking on the icon of an enabled breakpoint removes the breakpoint in the development environment. Clicking on a disabled breakpoint white circle enables the breakpoint.

- Clicking on the icon of a disabled breakpoint enables the breakpoint in the development environment, which is then indicated by a red circle in the breakpoint margin and the first column of the table in the Jade Debugger Breakpoints window.

  Alternatively, clicking on the icon of a disabled breakpoint removes the breakpoint in the development environment. Clicking on a disabled breakpoint white circle enables the breakpoint.

- Clicking the Browser toolbar **Toggle Breakpoint** button.
- Selecting the **Toggle Breakpoint** command from the Methods menu.
- Pressing F5.

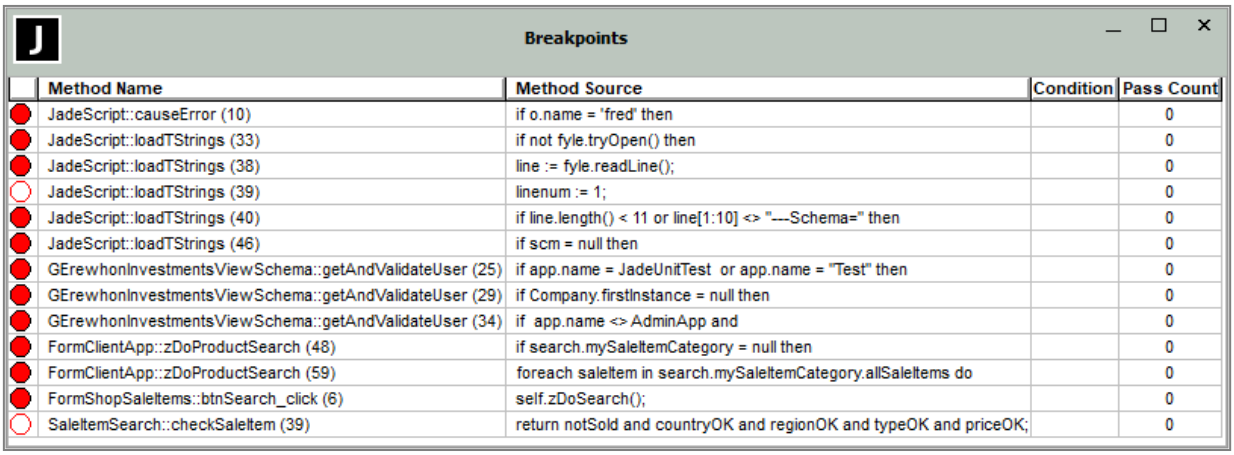
Displaying All Breakpoints in the Current Schema

The Breakpoints window lists all breakpoints that are set in the current schema.

» To show all breakpoints in the current schema, perform one of the following actions

- Select the **Show All BreakPoints** command from the Breakpoints menu
- Click the **Show all breakpoints** toolbar button

The Breakpoints window, shown in the following image, is then displayed.



	Method Name	Method Source	Condition	Pass Count
●	JadeScript::causeError (10)	if o.name = 'fred' then		0
●	JadeScript::loadTStrings (33)	if not fyle.tryOpen() then		0
●	JadeScript::loadTStrings (38)	line := fyle.readLine();		0
○	JadeScript::loadTStrings (39)	linenum := 1;		0
●	JadeScript::loadTStrings (40)	if line.length() < 11 or line[1:10] <> "---Schema=" then		0
●	JadeScript::loadTStrings (46)	if scm = null then		0
●	GErewhonInvestmentsViewSchema::getAndValidateUser (25)	if app.name = JadeUnitTest or app.name = "Test" then		0
●	GErewhonInvestmentsViewSchema::getAndValidateUser (29)	if Company.firstInstance = null then		0
●	GErewhonInvestmentsViewSchema::getAndValidateUser (34)	if app.name <> AdminApp and		0
●	FormClientApp::zDoProductSearch (48)	if search.mySaleItemCategory = null then		0
●	FormClientApp::zDoProductSearch (59)	foreach saleItem in search.mySaleItemCategory.allSaleItems do		0
●	FormShopSaleItems::btnSearch_click (6)	self.zDoSearch();		0
○	SaleItemSearch::checkSaleItem (39)	return notSold and countryOK and regionOK and typeOK and priceOK;		0

The columns in the Breakpoints window are automatically sized. The Breakpoints window is restored to its previous position and size if the **Save settings on exit** check box on the Debugger Options dialog is checked, but any previous column widths are no longer restored.

The first column indicates whether the breakpoint is enabled (a red circle) or disabled (with a white circle). Clicking on a red circle removes the breakpoint; clicking on a white circle enables a disabled breakpoint.

When the debugger is at a breakpoint, if nothing is selected and you hover the mouse over a variable, the value of the variable is displayed when possible (including path expressions). No methods are executed.

If an expression is selected, hovering over the selection attempts to execute the expression. The expression must be compilable within the context of the method and must return a value.

If the compile and execution succeed, the returned value is displayed.

Note It is your responsibility for the effects that the expression may have on the database and execution of the method logic.

Using Conditional Breakpoints and Pass Counts

The Breakpoints window in the Debugger includes:

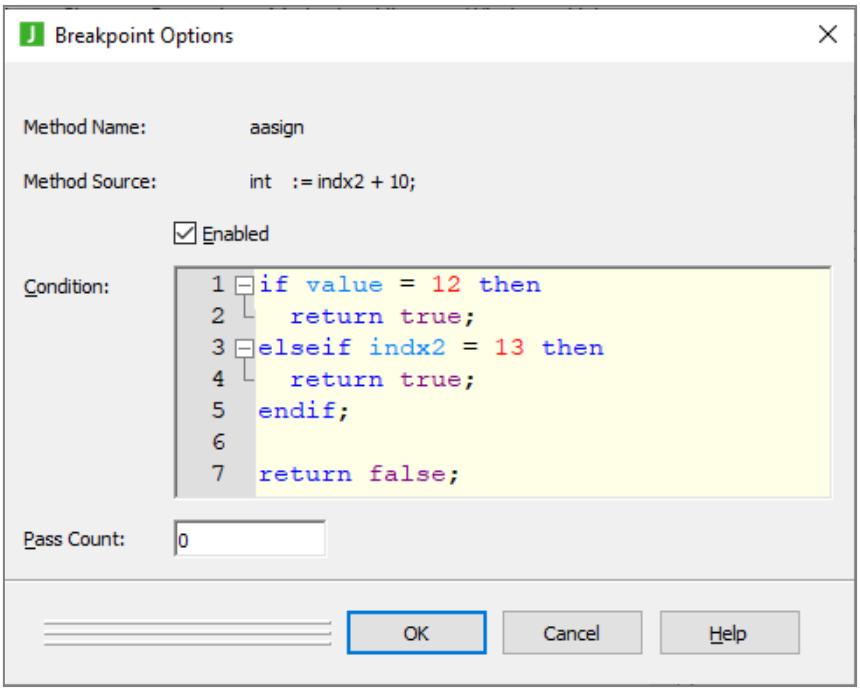
- An indicator that shows whether the breakpoint is enabled.
- The method name and line number for the breakpoint.
- The source of any associated condition.
- A pass count of the number of times a specific statement is executed.

The dialog can be resized.

» To specify options for the selected breakpoint

1. Right-click on a selected breakpoint in the Breakpoints window.
2. From the popup menu that is then displayed, select the **Options** command.

The Breakpoint Options dialog, shown in the following image, is then displayed.



3. Check the **Enabled** check box if you want to disable the current breakpoint. (Conversely, if the selected breakpoint is currently disabled, check the **Enabled** check box to enable that breakpoint.)

Note The Jade Debugger never halts execution at a disabled breakpoint.

Enabled breakpoints are displayed in the first column of the Breakpoints window with a red circle and disabled breakpoints with a white circle. Clicking on a red circle removes the breakpoint; clicking on a white circle enables a disabled breakpoint.

4. In the **Condition** control, specify a condition that you want evaluated, if required.

   A condition can be:

   - Any **Boolean** expression; for example, **"a > b"**, where **"a"** and **"b"** are defined in the scope of the receiving method.
   - One or more **if** instructions that specify a return value of **true** or **false**; for example:

```
if indx = 13 then
    return true;
elseif indx = 1 and cust.number = 345 then
    return true;
else
    return false;
endif;
```

     This condition must begin with the conditional **if** instruction; the final semicolon (**;**) of an expression is optional.

The **Condition** control allows for multiple-line input. In addition, the **Condition** control box also provides AutoComplete functionality. If AutoComplete functionality is enabled, you are provided with the usual prompts when the text is entered. The local parameters, local variables, and local constants for the method in which the breakpoint is defined are pre-loaded in AutoComplete for the entry of the condition. The condition logic expects a **Boolean** return value.

Jade constructs a transient method that is essentially as follows.

```
m1(): Boolean;
vars
    local-variable-and-parameters
begin
    load-locals-entities-with-values-from-the-actual-method-being-debugged
    return expression;  // for case 1 in the above list
    expression;         // for case 2 in the above list
end;
```

The condition source is validated when it is entered. If the enclosing method is subsequently recompiled, the breakpoint is marked for revalidation the next time it is encountered. If it is found to be invalid, the Debugger stops execution at the breakpoint and disables it.

5. In the **Pass count** text box, specify the number of times that the breakpoint is passed (that is, the number of times a specific statement is executed) before execution halts, if required.

   The pass count is decremented each time the breakpoint is reached and the Debugger stops when the count reaches zero (**0**).

6. Click the **OK** button. Your breakpoint options are saved if the **Save Breakpoints** check box is enabled in the **Exit** sheet of the Preferences dialog (accessed from the Options menu **Preferences** command). Alternatively, click the **Cancel** button to abandon your selections.

Breakpoint options (for example, disabling a breakpoint or specifying and changing conditions) can be specified or changed by using the Jade Debugger or the editor pane of the Jade Platform development environment.

Displaying Source Code for a Breakpoint

» To display the source code for a breakpoint, perform one of the following actions

- Double-click the breakpoint in the Breakpoints window.
- Right-click, and then select the **Show Source** command from the popup menu that is displayed.

The source code for the method associated with the breakpoint selected in the Breakpoints window is then displayed.

Removing and Clearing Breakpoints

» To remove a breakpoint selected in the Breakpoints window

- Right-click, and then select the **Remove** command from the popup menu that is displayed.

The selected breakpoint is then removed from the method source; that is, it is unset.

» To clear all breakpoints from the current schema, perform one of the following actions

- Right-click, and then select the **Clear All Breakpoints** command from the submenu that is displayed.
- Select the **Clear All BreakPoints** command from the Breakpoints menu.

All breakpoints in the current schema are then removed; that is, they are unset.

Note When a Jade method is changed and compiled, the location of any breakpoint in that method can change, depending on the nature of the modifications to the method source. This includes changes made by another developer (unless the method is checked out).

Using Watches in the Debugger

Watches are useful when you need to monitor or detect changes to parameters and local variables used in your methods. The Watches window displays the values of items that are being watched.

A path can be used as a debugger watch expression, but a path that contains a method call cannot be inspected.

Although the debugger accepts expressions as watches, they are limited to 100 characters (stored in an identifier array where the key is a maximum of 100 characters). The debugger accepts only the initial entry of a new watch expression if it successfully compiles, executes, and returns a value. To enter a new watch, therefore, the debugger must be at a break position where the expression is valid and the objects involved are not null.

When the watch is active, if it is not valid in another context, the value is displayed as **[Out of scope]**. If the expression cannot be executed because a variable is null, ****not Initialised**** is displayed.

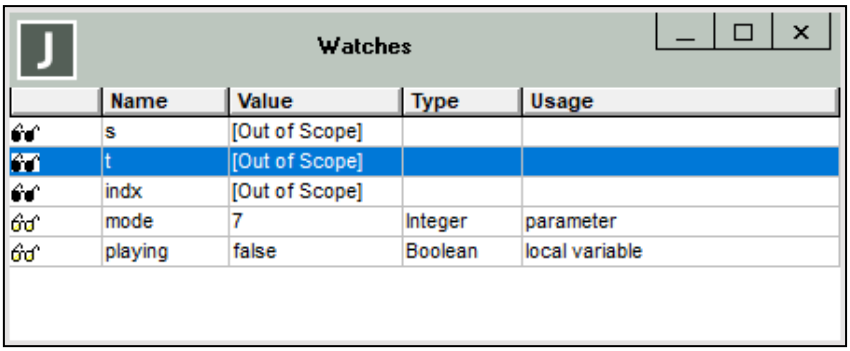
Note It is your responsibility for the effects that the expression may have on the database and execution of the method logic.

As your code executes, the values of the items change, and the Watches window is updated.

» To display the Watches window if it is not already displayed

- Select the View menu **Watches** command.

The following image shows an example of the Watches window.



For each item being watched, the name and value of the item are displayed. The value that is displayed is:

- The current value of the item, if the item is a primitive type
- An oid, if the item contains an object reference
- <null>, if the variable has no value assigned to it
- *[Out of scope]*, if the item is not within the scope of the method currently selected in the Call Stack window or it is

declared but not used in the method currently selected in the Call Stack window

- **not initialised**, if the expression cannot be executed because a variable is null

By default, the variables in the Watches window are displayed in the order in which they are added to the window. You can sort the table of displayed watches by clicking on the fixed column cell of the table. Clicking in the fixed part or a column causes the entries to be displayed alphabetically sorted (case-insensitive) by that column for the rest of the debugger session. For example, clicking on an identifier in the **Name** column sorts the list by identifier name. Clicking the sorted fixed column cell again inverts the sorted list.

If the table is sorted, a numeric sorting indicator is displayed in the fixed cell of the column. If the column selected for sorting is *not* the identifier column, the identifier column is the second sort key.

Notes Values being watched are highlighted when they change.

Position the cursor over an object or variable in a line of code in the Method Source window to display the current value of that item in bubble help. (For details about the circumstances under which bubble help is displayed by default, see "Displaying Code in the Method Source Window", earlier in this chapter.)

Selecting a Method in the Call Stack Window

To select a method whose execution you want to watch, view the Call Stack window and then select one of the methods in the list.

To display the Call Stack window if it is not currently displayed

- Select the View menu **Call Stack** command.

As the methods are selected in the Call Stack window, the Watches list is updated to reflect the values that are within the scope of the selected method. If no method is selected in the Call Stack window, the values are displayed for the method that is currently executing.

Items that can be watched are:

- Method parameters
- Local variables

Tip Double-click on an entry in the Call Stack window to display the Inspector form for the receiver of that method. (For details, see "Inspecting Object Instances", in Chapter 6.)

To copy the current call stack to the clipboard as a string

1. Right-click on the call stack results.
2. Select the **Copy Call Stack to Clipboard** command from the popup menu that is then displayed.

Call stack entries copied to the clipboard are delimited by carriage return / line feed characters (**CrLf**).

Adding a New Watch

The **Add Watch** command from the Variables menu enables you to specify an item to watch. The item that is being watched must be "in scope" at the time it is added to the Watches list.

To add a new watch when the item has focus in the method source window

1. Click on the item to be watched in the method source window.
2. Select the **Add Watch** command from the Variables menu.

   Alternatively, you can click the **Add Watch** toolbar button.

The item on which the caret was positioned is then displayed in the Watches window.

To add a new watch when no item has focus in the method source window

1. Select the **Add Watch** command from the Variables menu. Alternatively, you can click the **Add Watch** toolbar button.

   The Watch Variable dialog is then displayed if the caret was not positioned on a valid object, to enable you to specify the variable that is to be watched.

   Note The Watch Variable dialog is not displayed when the caret is positioned on a valid item in the method source window.
2. In the **Variable Name** text box, specify the variable that is to be watched.
3. Click the **OK** button.

The specified variable is then displayed in the Watches window or the Inspector form.

Removing a Watch

To remove a watch

1. In the Watches window, select the watch item that is to be removed.
2. Select the **Remove Watch** command from the Variables menu.

The watch is then removed from the Watches window.

To remove all watches

- Select the **Clear All Watches** command from the Variables menu.

All watches are then removed from the Watches window.

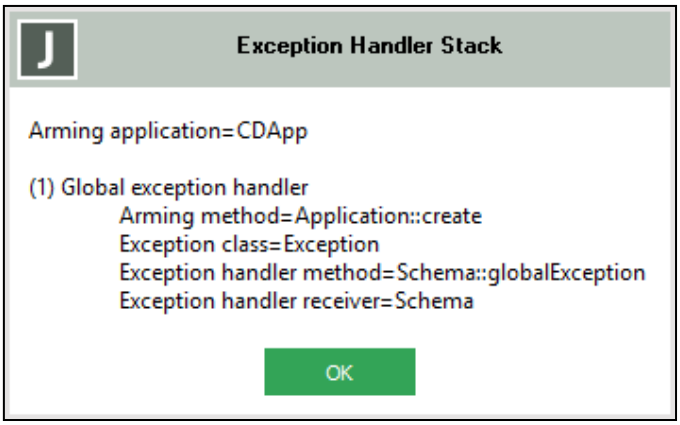
Displaying the Current Exception Handler Stack

The Exception Handler Stack message box enables you to view all exception handlers armed by the receiving process in the current node.

To display the current exception handler stack

- Select the View menu **Show Exception Handler Stack** command.

The Exception Handler Stack message box, shown in the following image, is then displayed.



The exception handler stack displays:

- The name of the application that is arming the exceptions.
- For each armed exception handler, the following information is displayed.
  - The stack depth (the handlers at the top of the stack are invoked first)
  - Whether the exception handler is a global or a local exception handler
  - The name of the class and method that armed the exception (this applies to local exception handlers only)
  - The name of the exception class
  - The name of the class and method of the exception handler method
  - The name of the class of the exception handler receiver

Viewing Local Variables in the Debugger

The Local Variables window enables you to monitor or detect changes to parameters and local variables used in your methods.

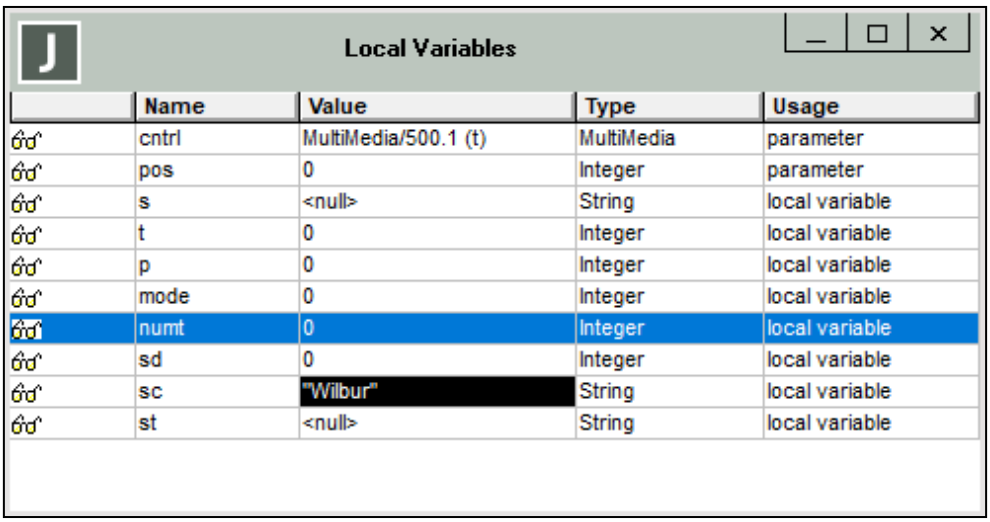
If call stack capturing is enabled, you can view the creation or deletion call stack of an object. For details, see "Viewing Creation or Deletion Call Stack Information", later in this chapter.

» To display the Local Variables window if it is not already displayed

- Select the View menu **Local Variables** command.

Tip As the Local Variables window automatically monitors *all* parameters and local variables in your methods, you do not have to access the Watch Variable dialog to specify each variable that you want to watch in the Watches window.

The Local Variables window, shown in the following image, displays the values of all local variables in your methods. As your code executes, the values of the items change and the Local Variables window is updated. Local variables being watched are highlighted when values change.



	Name	Value	Type	Usage
	cntrl	MultiMedia/500.1 (t)	MultiMedia	parameter
	pos	0	Integer	parameter
	s	<null>	String	local variable
	t	0	Integer	local variable
	p	0	Integer	local variable
	mode	0	Integer	local variable
	numt	0	Integer	local variable
	sd	0	Integer	local variable
	sc	"Wilbur"	String	local variable
	st	<null>	String	local variable

To better display longer and multiple line values, you can resize the Local Variables window.

For each local variable being watched, the name and value of the item are displayed. The value that is displayed is:

- The current value of the variable, if the item is a primitive type
- An oid, if the variable contains an object reference
- <null>, if the variable has no value assigned to it
- *[Out of scope]*, if the variable is not within the scope of the method currently selected in the Call Stack window, or it is declared but not used in the method currently selected in the Call Stack window
- self, if the method references the current instance (method receiver) in any expression

Notes Local variable values being watched are highlighted when they change.

Position the cursor over a local variable in a line of code in the Method Source window to display the current value of that item in bubble help. (For details about the circumstances under which bubble help is displayed by default, see "Displaying Code in the Method Source Window", earlier in this chapter.)

By default, the variables in the Local Variables window are displayed in the order in which they appear in the method. You can sort the table of displayed variables by clicking on the fixed column cell of the table. Clicking in the fixed part or a column causes the entries to be displayed alphabetically sorted (case-insensitive) by that column for the rest of the debugger session. For example, clicking on an identifier in the **Name** column sorts the list by identifier name. Clicking the sorted fixed column cell again inverts the sorted list.

If the table is sorted, a numeric sorting indicator is displayed in the fixed cell of the column. If the column selected for sorting is *not* the identifier column, the identifier column is the second sort key.

To select a method, view the Call Stack window (by selecting the **Call Stack** command from the View menu), and then select one of the methods in the list.

As the methods are selected, the Local Variables list is updated to reflect the values that are within the scope of the selected method. If no method is selected, the values are displayed for the method that is currently executing.

You can watch:

- Method parameters
- Local variables

You can inspect or modify variables selected in the Local Variables window. For details, see "Inspecting and Modifying Values", later in this chapter.

Viewing Creation or Deletion Call Stack Information

If call stack capturing is enabled, you can view the creation or deletion call stack of an object. For details, see "Enabling or Disabling the Capture of Object Call Stacks", later in this chapter.

» To view the creation or deletion call stack of an object

- In the Local Variables window, right-click on that item and then select the **Creation** or **Deletion** option from the **Call Stack** command that is displayed in the popup menu.

The call stack details are then displayed in the Call Stack Browser.

Note The **Call Stack** command is displayed only if call stack information exists for the object.

Applies to Version: 2022.0.03

Enabling or Disabling the Capture of Object Call Stacks

To view the creation or deletion call stack of an object from the Local Variables window, you must enable the capture of call stack information.

» To toggle the capture of create and delete object call stacks

- To enable the capture of object call stacks, select the **Capture Object Call Stacks** command in the Variables menu. A check mark symbol is displayed at the left of the command when the capture of object call stacks is enabled.

  Conversely, to disable the capture of object call stacks, select the **Capture Object Call Stacks** command in the Variables menu so that it no longer displays a check mark symbol.

Note To automatically save object create or delete call stack information, set the **DefaultProcessSaveCreateCallStack** or the **DefaultProcessSaveDeleteCallStack** parameter to **true** in the appropriate [JadeClient] or [JadeServer] section of the Jade initialization file.

Applies to Version: 2022.0.03

Inspecting and Modifying Values in the Debugger

Use the **Inspect** command from the Variables menu to display or modify the value of a primitive item or to inspect an item containing an object reference.

To inspect the value of an item

1. Specify the selected item, by performing one of the following actions.
   - Click on the item in the method source window, and then click in the **Inspect** toolbar button or press Ctrl+I.

     Note The Inspector form is then displayed; that is, the Variable Inspector dialog is not displayed first.
   - Select the **Inspect** command from the Variables menu.
   - Click on the **Inspect** toolbar button.
   - Press Ctrl+I.

   The Inspect Variable dialog is then displayed. To enter longer expressions, you can resize the Inspect Variable dialog.

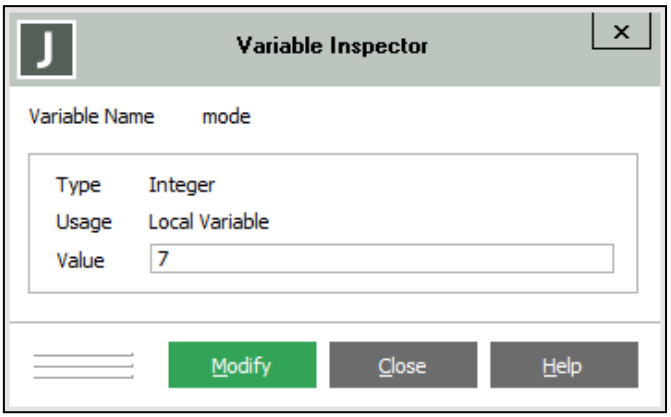
   The dialog accepts expressions that can be compiled and executed. There are no restrictions other than the expression able to be compiled and return a value. Expressions longer than 100 characters are accepted, and it is your responsibility for the effects that the expression may have on the database and execution of the method logic.
2. In the **Variable Name** text box, specify the item to be inspected.
3. Click the **OK** button.

If the item being inspected contains an object reference, an Inspector window is opened for that object. For details, see "Inspecting an Object Reference", later in this chapter.

Tip You can also display the bubble help containing the current value of a local variable in a line of code in the Method Source window, by positioning the cursor over the local variable whose value you want displayed. However, inspection of properties from within the Jade debugger may fail if a user mapping method performs actions not suitable for situations where the execution of the user application is suspended; for example, the creation and display of a form, showing a form modally, display of a message box, deletion of an object in use by the current call stack, and so on.

For details about the circumstances under which bubble help is displayed by default, see "Displaying Code in the Method Source Window", earlier in this chapter.

If the item being inspected contains a primitive type, an invalid object reference, or a **null** object reference, or you right-click in the Local Variables window and then select the **Inspect** command from the popup (context) menu that is displayed, the Variable Inspector dialog, shown in the following image, is displayed.



The name of the item on which the caret is positioned in the Method Source window is displayed and the primitive type, usage, and the current value are displayed in the group box.

You can resize the dialog or scroll the **Value** text box to view a long string or binary value, if required. The **Value** text box is resized as the dialog is resized.

The **Value** text box and **Modify** button are disabled if the variable cannot be modified; that is, it is an **Any** primitive type.

» To modify the value of the variable associated with the current method that is executing in the debugger

1. In the **Value** text box, specify the new value for the selected primitive type variable.

   The type of value that you specify must be the same as that of the selected value; for example, you cannot change the value of a numeric variable to a string value.

2. Click the **Modify** button.

Notes The **Value** text box and the **Modify** button are enabled only if it is valid to change the variable value.

Inspecting an Object Reference

If the item contains an object reference, an Inspector form is opened for the object when you click the **OK** button in the Inspect Variable dialog.

The specified object and any properties defined for that object are then displayed in the left of the Inspector form.

» To view the values for the displayed properties

- Click on the required property.

The values for the property are then displayed in the pane at the right of the form.

For details about using the Schema Inspector, see "Using the Inspector Form" and "Inspecting Variable Instances in the Debugger", in Chapter 6.

Terminating a Debugger Session

» To terminate both the debugger and the active application

- Select the **Terminate** command from the Debug menu.

Both the debugger and your Jade application are then terminated.

Ending a Debugger Session

» To close the debugger

- Select the **End** command from the Debug menu.

The debugger is then terminated and your Jade application continues running.

Notes Your debugger session is also terminated when you end your Jade application.

If you close down the debugger when you are in transaction state (by using the **End** command from the Debug menu, Alt+F4, or by clicking the close icon at the top right corner of the window), your current transaction is aborted.

Chapter 8 Adding External Methods and External Functions to Classes

This chapter covers the following topics.

- Adding External Methods to Classes
  - Writing External Methods
- Defining External Functions
  - Defining and Maintaining External Functions
- Maintaining Libraries for External Methods and Functions
  - Adding a Library

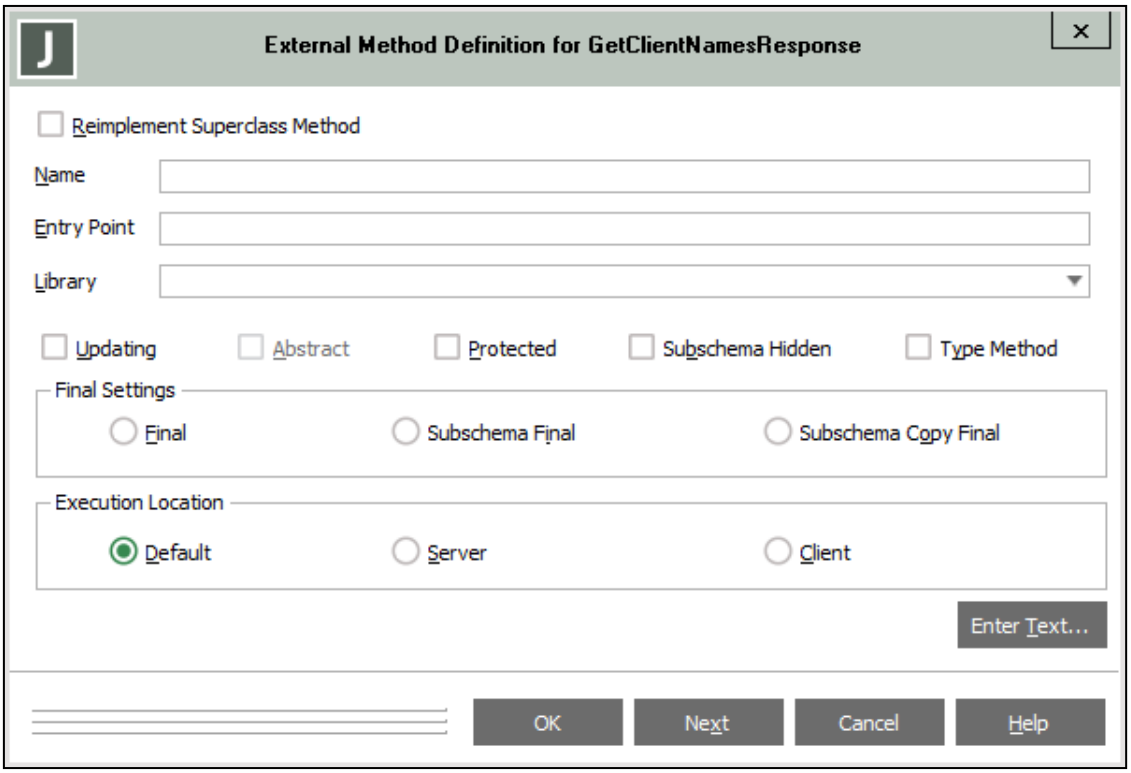
Adding External Methods to Classes

You can write external user-methods (or routines) for Jade classes. External routines can be written in any language that can create a library. For details, see "Writing External Methods", later in this chapter and "Jade Application Programming Interface (API)", in Chapter 3 of the *Object Manager Guide*.

To add an external method to a class

1. In the Class List of the Class Browser, select the class to which the external method is to be added.
2. Select the **New External Method** command from the Methods menu in the Class Browser.

The External Method Definition dialog, shown in the following image, is then displayed.



3. Check the **Reimplement Superclass Method** check box if you want to reimplement an existing external method in a superclass. (You can reimplement an external method only if it is not final in the superclass. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.) A drop-down list containing all superclass external methods that are not final is then displayed.

   Select the method in the parent class that you want to reimplement in the current class. A message box then advises you that you are about to reimplement a superclass method, and prompts you to click the **Yes** button if you want to continue. The parent class and the method that you selected for reimplementation are then displayed in the **Name** text box.

4. Specify the name of your new external method in the **Name** text box; for example, **newExtMethod**. The external method name has a maximum length of 100 characters.

5. In the **Entry Point** text box, specify the name of the function in the library that is to be the entry point for your external method.

   Although the entry point name has a maximum length of 255 characters, if you specify an entry point name greater than 100 characters, it must be enclosed in single quote (**'**) or double quote (**""**) characters or exception 6007 (*Token too long*) is raised. (The schema extract process encloses *all* entry point names in quote characters.)

   The entry point name can include a period, the underscore (**_**), opening parenthesis (**(**), closing parenthesis (**)**), or the ampersand (**&**) special character. In addition, the entry point name is case-sensitive.

   You cannot specify an entry point for an abstract external method of an abstract class.

6. In the **Library** combo box, select the library containing your library (**.dll**) file from the list portion of the combo box or specify the name of an existing library in the text portion.

If you specify a library in this combo box and you specified an entry point in the **Entry Point** text box, subsequently checking the **Abstract** check box (which is enabled only for an abstract class) removes values entered in the entry point and library controls.

For details about libraries for your external routines, see "Maintaining Libraries for External Methods and Functions", later in this chapter.

7. Check the **Updating** check box if the method can modify properties in the object to which it is sent.

8. If you did not specify an entry point and library for your external method, check the **Abstract** check box if the selected class is *abstract*. An abstract method defines a signature with which all subclass implementations must comply.

   The **Abstract** check box is enabled only when the selected class is an abstract class.

9. Check the **Protected** check box if you want the **protected** method option added to the signature of your method so that it can be referenced only by methods in the same class or its subclasses. (A protected method is displayed in the browser Methods List with a padlock icon to the left of the method name.)

   By default, external methods are not protected; that is, they are displayed in the Methods List of the Class Browser or Primitive Types Browser with the public access icon to the left.

10. Check the **Final** check box if you want to specify that the external method cannot be reimplemented in a subclass. The **final** method option makes methods available to other schemas but prevents those schemas from modifying, reimplementing, or circumventing the defined method behavior. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

11. Check the **Subschema Hidden** check box if you want to specify that the external method is available only in the local schema; that is, it is not available for use in any subschemas. For details, see "subschemaHidden Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

12. Check the **Type Method** check box if you want the **typeMethod** method option added to the signature of the external method so that the method is a type method.

    Type methods provide a way of calling a method declared on a type (class, primitive, or interface) without having to have an instance of the type. For more details, see "Type Methods", under "Jade Language Notation" in Chapter 1 of the *Developer's Reference*.

13. In the Final Settings group box, select one of the following option buttons for the appropriate final setting, if required.

    - **Final**, if you want to specify that the external method cannot be reimplemented in a subclass. The **final** method option makes external methods available to other schemas but prevents those schemas from modifying, reimplementing, or circumventing the defined method behavior. For details, see "final Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

    - **Subschema Final**, if you want to specify that the external method can be extended or reimplemented in its local schema but not in a subschema.

      The **subschemaFinal** method option makes external methods available to other schemas but prevents those schemas from modifying, reimplementing, or circumventing the defined method behavior. For details and an example, see "subschemaFinal Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

    - **Subschema Copy Final**, if you want to specify that the external method cannot be reimplemented in a subschema copy class. For details and an example, see "subschemaCopyFinal Option" under "Controlling the Use of Elements in Other Schemas", in Chapter 1.

14. In the Execution Location group box, select the **Server** or **Client** option button if you want the **serverExecution** or **clientExecution** method option added to the signature of your external method so that the method and all methods subsequently called by this external method are executed on the server node or client node, respectively.

    By default, no execution option is added to the external method signature and the node in which the method executes is determined by the location of the calling method. For more details, see "Method Options", in Chapter 1 of the *Developer's Reference*.

15. Click the **Enter Text** button if you want to specify or maintain descriptive text for the external method as part of the definition or maintenance of the method. For details, see "Specifying Text for a Schema Element", in Chapter 3.

    Tip You can also specify descriptive text for the external method at any time, by selecting the **Text** command from the Methods menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

16. Click the **OK** button or the **Next** button.

A template for the method is then displayed in the editor pane of the Class Browser. Modify the template by adding your external routine code to perform the required operations and then compile it. (For more details, see "Compiling Methods", in Chapter 4 of this guide and "Method Options", in Chapter 1 of the *Developer's Reference*.)

Note Before you can define a new external method, the library containing your library file must already exist and you must select the class to which the method is to be added. For details, see "Maintaining Libraries for External Methods and Functions", later in this chapter.

Writing External Methods

The **demodll** folder in the Jade **examples** directory provides you with a demonstration schema and a C++ external method (**demodll.cpp**). For details and an example of a C++ external method, see "Using External Methods", in Chapter 1 of the *External Interface Developer's Reference*.

Defining External Functions

An external function is a routine that is not necessarily associated with a specific class.

Note As external functions are global to the schema in which they are defined and are not supported at class level, an external function name must be unique to the schema in which it is defined.

An external function is a function that has been exported from a Dynamic Link Library (DLL) and can be called directly from a method written in the Jade language. This enables you to directly call entry points exported from existing third-party libraries or the operating system without having to write an external method wrapper.

An external function defines a mapping between Jade parameters and the external parameters. Parameter mapping is made interpretively at run time when an external function is called.

Defining an external function is very much like defining an external method and a subset of the same information is captured. However, external functions are viewed and maintained from the External Functions Browser, accessed from the Browse menu at schema level, unlike external methods that are viewed and maintained from the Class Browser or the Primitive Types Browser.

External function calls are invoked with the Jade language **call** instruction, to make a clear distinction between *function* calls (that are not invoked on an object) and object *method invocations*.

For details about parameter mapping, see "Using External Functions", in Chapter 1 of the *External Interface Developer's Reference*.

Note If you want to add an external function to a library defined in the **RootSchema** (for example, to the **kernel32** or the **user32** library), you must first view the superschema (**RootSchema**) from the External Functions Browser in your own schema. To do this, select the **Superschemas** command from the View menu in the External Functions Browser, select the appropriate library, and then add the external function that you require; for example, **sendMessage**.

Defining and Maintaining External Functions

Before you can define a new external function, the library containing your library file must already exist.

You can use the same library for both external functions and external methods, if required. For details about adding a library, see "Maintaining Libraries for External Methods and Functions", later in this chapter.

For details about maintaining external functions, see the following subsections.

Adding an External Function

You can add an external function to your schema at any time.

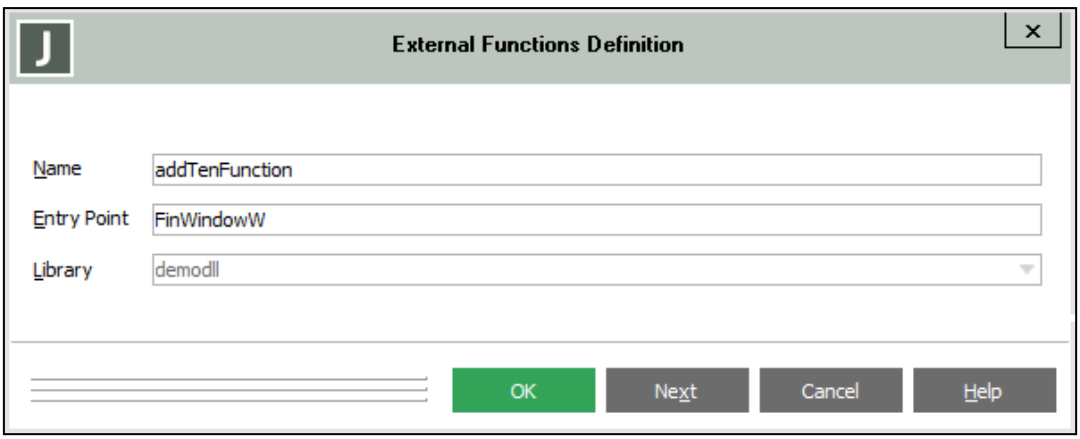
To add an external function to the current schema

1. Select the **External Functions** command from the Browse menu in the Schema, Class, or Primitive Types Browser.

   The External Functions Browser is then displayed. The left pane contains a list of the external functions defined in the current schema and the right pane is an editor window that displays the function header of the external function currently selected in the external functions list.

2. Select the **Add** command from the Functions menu.

   The External Functions Definition dialog, shown in the following image, is then displayed.



3. In the **Name** text box, specify the name of your new external function; for example, **findWindow**. The external function name has a maximum length of 100 characters.

4. In the **Entry Point** text box, specify the entry-point name of the function exported from the library; for example, **FindWindowW**.

Although the entry point name has a maximum length of 255 characters, if you specify an entry point name greater than 100 characters, it must be enclosed in single quote (**'**) or double quote (**"**) characters or exception 6007 (*Token too long*) is raised. (The schema extract process encloses *all* entry point names in quote characters.)

The **Library** list box displays the name of the library that you selected in the External Functions Browser; for example, **user32**. This predefined library is disabled.

The entry point name can include a period, the underscore (**_**), opening parenthesis (**(**), closing parenthesis (**)**), or the ampersand (**&**) special character. In addition, the entry point name is case-sensitive.

5. Click the **Enter Text** button if you want to specify or maintain descriptive text for the external function as part of the definition or maintenance of the function.

   For details, see "Specifying Text for a Schema Element", in Chapter 3.

   Tip You can also specify descriptive text for the external function at any time, by selecting the **Text** command from the Functions menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

6. Click the **Next** button to define your next external function or click the **OK** button to define your external function and display a template for the function in the editor pane of the External Functions Browser.

   The following is an example of an external function template, or signature.

   ```
   findWindow()is FindWindowW in user32 applicationServerExecution;
   ```

   Alternatively, click the **Cancel** button to abandon your selections.

For details about the **presentationExecution** and **applicationServerExecution** external function call options, see "Calling External Functions from Jade Thin Clients", in Chapter 1 of the *External Interface Developer's Reference*.

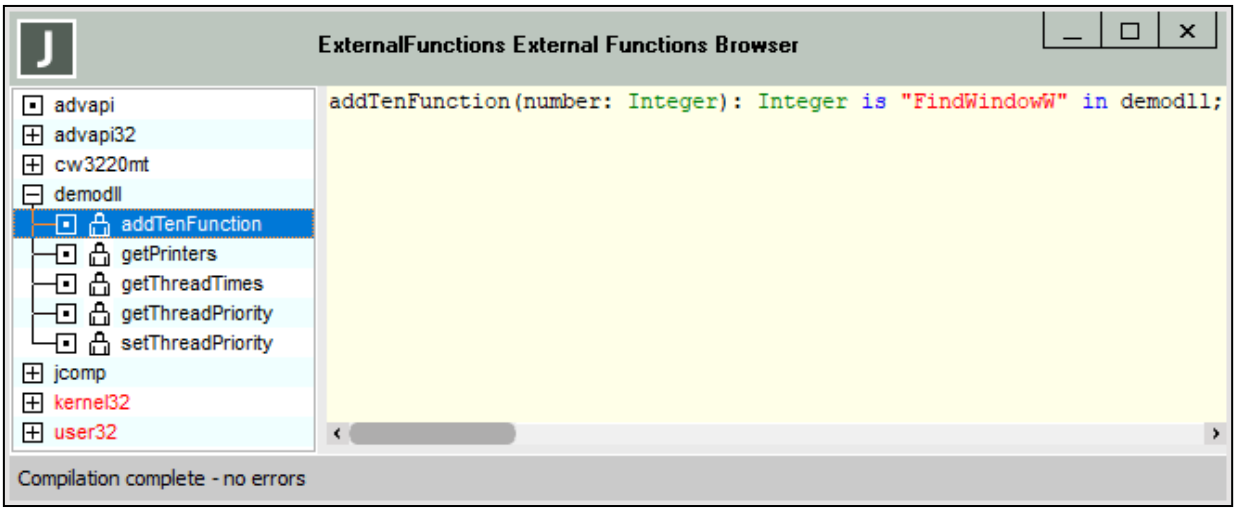
Modifying the External Function Template

When you have defined your external function, you can then modify the template in the editor pane of the External Functions Browser to add parameters and a return type, if required.

When modifying your external function:

- The external function must exist
- You must specify a length for usage **io** or **output** parameters
- Parameter types must be valid
- Unsigned qualifiers are not directly supported
- Structured record type parameters are supported only by passing a fixed-length binary by reference

The following shows an example of the External Functions Browser containing a modified **FindWindow** function.



When you have modified your external function signature, it can then be compiled.

Compiling Your External Function

When you have defined or modified an external function, you must compile it before you can execute it.

» To compile your external function, perform one of the following actions

- Select the **Compile Function** command from the Functions menu.
- Press the F8 key.

The external function is then compiled. The first error that is detected is highlighted and the corresponding error message is displayed in the status line, to enable you to amend the code.

When the method is compiled and no errors are detected, the message *Compilation complete - no errors* is displayed in the status line.

Example of Definition of an External Function

The following example shows the definition of an external function.

```
copyFile(input:  String;
         output: String;
         exists: Boolean): Boolean is CopyFileA in kernel32;
```

In this example, the **value** parameter is usage output (that is, the called function is known to update this string parameter).

The specification of a value for the **length** parameter is mandatory, to allow a string of the correct size to be allocated.

The Unicode version of this WIN32 API has been used in this example, which assumes that callers pass Unicode strings (the default for a Unicode Jade build).

Use the **presentationClientExecution** or **applicationServerExecution** option in the function definition of an external function (that is, *not* in the **call** expression) to specify where the function call is made, as shown in the following example.

```
getProfileString(sectionName : String;
                 keyName     : String;
                 default     : String;
                 value       : String[100] output;
                 length      : Integer): Integer is GetProfileStringW in
                                        kernel32 applicationServerExecution;
```

External functions are called on the Jade thin client workstation by default; that is, **presentationClientExecution** is assumed.

Note The **presentationClientExecution** qualifier has no effect if Jade is not currently running in Jade thin client mode.

Example of Usage of an External Function

The following example shows the usage of an external function.

```
getIniParameter(key: String; default: String): String;
vars
    value : String[100];
    size  : Integer;
begin
    size := call getProfileString("JadeClient", key, default,
                                   value, value.maxLength);
    return value;
end;
```

In this example, a fixed-length string is passed to the **value** parameter. The length of the **value** parameter string is passed in the **length** parameter. (This could also have been hard-coded to the length of the **value** parameter defined in the function signature.)

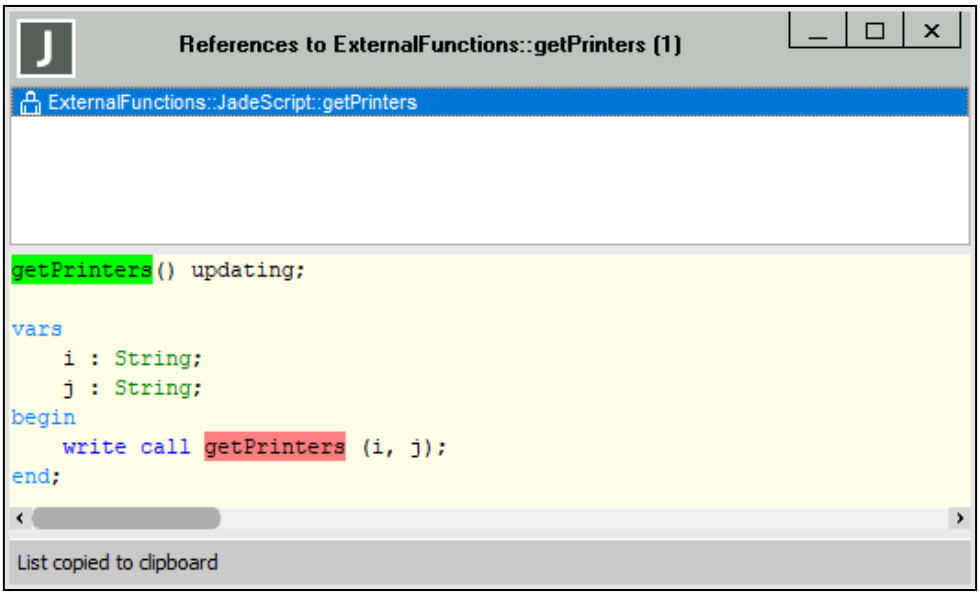
Note An exception is raised at run time if the number or types of parameters in your external function definition does not match the parameters expected by the function in the external library. This may be exception 4038 (*External function caused an exception*) if the error was detected at the time the function call was made.

Exception 4024 (*Out of resources for operation*) may be raised at some time after the external function was called if, for example, an incorrect parameter definition caused the external function to corrupt memory.

Viewing References to an External Function

Use the **References** command from the Functions menu to display all references in the schema to the function selected in the External Functions Browser.

The References window for the selected external function is then displayed, as shown in the following image.



This window lists all methods that reference the selected external function, and enables you to view specific references to the function.

In the References window, select the method whose reference to the selected external function you want to view. The details of the selected method are then displayed in the editor pane. You can maintain a method displayed in the editor pane of the References window.

Extracting an External Function or Library

To extract an external function or library of external functions

- Select the **Extract** command from the Functions menu.

The common Save As dialog is then displayed, to enable you to specify the name and location of your extract.

If the dialog is accessed when an external function library is selected, all external functions in that library are extracted to an .**scm** file, which has a prefix of **ExternalFunctions_*schema-name***.

If the dialog is accessed when an external function is selected, that external function is extracted to a .**mth** file, which has a prefix of ***schema-name_external-function-name***. The location defaults to your Jade working directory.

Maintaining Libraries for External Methods and Functions

The Library Browser enables you to view and maintain the libraries for your current schema. You can use the libraries, which store your external user-written Application Program Interface (API) calls, to import both external methods and external functions, if required.

You must create a library for your external routines before you can define external methods or external functions for the API entry points.

Note Only one Library Browser window can be open at any time.

To open a Library Browser window

- Select the **Libraries** command from the Browse menu

The Library Browser is then opened. The Library Browser is displayed in the form specified by your Browser options; that is, it is user-specific. You can change your default browser options, if required, by using the **Browser** sheet from the Options menu **Preferences** command.

The Jade default display of the Library Browser is as follows.

- User-defined libraries are displayed in black.
- Libraries are displayed in hierarchical order, with nodes collapsed; that is, the external routines in the selected library are not displayed when the Library Browser is first opened.

The actions that you can perform by using the mouse from the Library Browser are listed in the following table.

Action	Result
Click	A selected collapsed node is expanded when a node displaying a plus sign (+) is clicked, or an expanded node is collapsed when a node displaying a minus sign (-) is clicked.
Click-right	Displays the Library menu, to enable you to maintain the selected library.

The Library menu is then displayed, to enable you select the appropriate command. The Library menu commands are listed in the following table.

Command	Description	For details, see…
Add Library	Displays the Add Library dialog	Adding a Library, in the following section
Remove Library	Deletes the selected library	Removing a Schema Element, in Chapter 3

Adding a Library

The **Add Library** command from the Library menu in the Library Browser enables you to add a new library to the current schema.

To add a library

1. Select the **Add Library** command from the Library menu in the Library Browser. The Add Library dialog is then displayed.
2. In the **Name** text box, enter the name of the library that you want to create. The name must begin with a lowercase character and it must be unique to the schema to which it is being added.
3. Click the **OK** button.

Alternatively, click the **Cancel** button to abandon your selections.

The specified library containing your user-routine libraries is then displayed in the Library Browser. The Library Browser includes the full method name for each entrypoint, in the following format.

```
schema-name::class-name::method-name
```

You can create external methods or functions for your external routines (by using the **New External Method** command from the Methods menu or the **External Functions** command from the Schema menu, respectively), specifying the name of an existing library and the library entry points.

For details about adding external routines, see "Adding External Methods to Classes" or "Defining External Functions", earlier in this chapter.

Chapter 9 Defining ODBC Inquiry Relational Views and Ad Hoc Indexes

This chapter covers the following topics.

- Overview
- Maintaining Relational Views
  - Defining Your Relational View
  - Adding a Relational View
    - Naming Your Relational View
    - Specifying Relational View Options
    - Selecting Classes and User-Defined Tables for Your Relational View
    - Setting the Root Class
    - Setting the Default Included Object Features
    - Setting the Visibility of Protected Features
    - Setting the Visibility of Derived Tables
    - Refining the Visibility of Class Features
    - Renaming Tables and Columns
    - Building Your Relational View
  - Removing a Relational View
  - Changing a Relational View
  - Printing a Relational View
  - Extracting a Relational View
  - Loading a Relational View
- Maintaining Ad Hoc Indexes
  - Defining an Ad Hoc Index
  - Adding an Ad Hoc Index
  - Changing an Ad Hoc Index
  - Deleting an Ad Hoc Index
  - Building an Ad Hoc Index
  - Canceling an Ad Hoc Index Build
  - Dropping an Ad Hoc Index

- Loading Ad Hoc Definitions
- Saving Ad Hoc Definitions
- Running the Ad Hoc Index Controller Application

Overview

An Open Database Connectivity (ODBC) interface is provided with Jade. This enables you to apply Structured Query Language (SQL) statements to access a relational view of your Jade database and to maintain ad hoc indexes. You can use the ODBC interface for inquiries using products such as Crystal Reports.

You can use ad hoc indexes to optimize ad hoc queries without requiring database reorganization. The ODBC driver uses available ad hoc indexes if they exist. When it has established which, if any, indexes to use for a query, indexes are session read-locked in OID order, so that indexes are not updated while the ODBC driver is building the result set.

These indexes are used if the query requires **AllInstances** with a **WHERE** clause that includes the properties of the defined index keys; for example:

```
SELECT * FROM Invoice WHERE Invoice.BuyDate > '1/1/13'
```

If an ad hoc index exists for **Invoice** with a key of **BuyDate**, this index is used instead of **AllInstances**. In addition the, ODBC driver handles joins between different tables when there is an ad hoc index that fits the join (equality tests only); for example:

```
SELECT * FROM Withdrawal, Deposit

WHERE Withdrawal.date > '1'1'13' AND Withdrawal.amount = Deposit.amount
```

If there was an ad hoc index over **WithDrawal** by date and another one on **Deposit** by amount, the ODBC driver looks up all **WithDrawal** instances for the date range and directly access all deposits for the **WithDrawal** amount via the **Deposit amount** ad hoc index.

The **OidFieldSeparator** parameter in the [JadeOdbc] section of the Jade initialization file on the server enables you to customize the OID field separator with a single punctuation or similar character (for example, **@**). This can be useful when the default OID String values that are produced are misinterpreted by a third-party tool as a different ODBC type such as a Decimal.

The following sections contain information about relational views and ad hoc indexes for ODBC.

For details about ODBC inquiries, including installing an ODBC driver, see "Obtaining a Relational View of Your Jade Database", in Chapter 2 of the *External Interface Developer's Reference*. For details about the ODBC reserved words, see Appendix A of the *External Interface Developer's Reference*.

Maintaining Relational Views

The Jade Relational Views Wizard, installed with your Jade software, enables you to define relational views of your Jade database. When you have defined a relational view of your Jade database, you must then install an ODBC driver in order to access your relational view.

Notes Only one Relational Views Browser for the current schema can be open at any time. If a Relational Views Browser is already open for that schema, it is brought to the top when you select the **Relational Views** command from the Browse menu. You can have concurrent open Relational Views Browsers for different schemas in the current development environment session.

When you set the **TerminateProcessOnDisconnect** parameter in the [JadeClient] section of the Jade initialization file to **true**, unwanted side-effects may occur when the connection to the Jade database server came via ODBC from an external database application, as the underlying program that opened the connection is also terminated.

You can add user-defined tables (that is, soft entities or attributes) to or remove them from a relational view but you cannot use the Relational View wizard to change them. If your relational view includes user-defined tables, they:

- Are not extracted when the schema or relational view is extracted because the user-defined entities are dependent on user data rather than schema meta data, so it cannot be loaded into other Jade systems.
- Must be created by executing user code for all Jade systems, as required.
- Must be re-created if a full schema load or relational view load is performed.

See the **Schema** class **regenerateRelationalView** method, in Chapter 1 of the *Encyclopaedia of Classes*, for details about dynamically building a relational view at run time after changes to the schema.

The **RelationalView** class provides an Application Programming Interface (API) that supports relational access to a Jade database (as opposed to replicating data to a relational database) through the Jade ODBC driver. This enables you to can apply Structured Query Language (SQL) statements or use products such as Crystal Reports to access your Jade database. For details, see Volume 2 of the *Encyclopaedia of Classes*.

Defining Your Relational View

Use the Browse menu **Relational Views** command from the Schema Browser to develop a relational view of the current schema. This relational view then enables you to access Jade data using SQL select statements by the Jade Open Database Connectivity (ODBC) driver.

Each schema within the Jade database can have a collection of relational views.

Notes A memory-based copy of your relational view is created by the Relational View Wizard and developed at each step of the process. The generation of class features and the saving of the view are the most time-intensive steps of the Relational Views wizard. A large schema the size of the Jade schema should take only a few seconds for these steps.

The Relational Views Browser enables you to maintain your relational views only. To access the Jade data in the relational view using SQL select statements, you must first write your own inquiry programs. For details, see "SQL Examples", in Chapter 2 of the *External Interface Developer's Guide*.

» To open a Relational Views Browser window, perform one of the following actions

- Select the **Relational Views** command from the Browse menu
- Press Ctrl+O

A Relational Views Browser window is then opened.

If you have not yet defined a relational view, nothing is displayed in the Relational Views Browser.

Using the Relational Menu

The Relational Views Browser provides the Relational menu, to enable you to maintain your relational views. The Relational menu contains the commands listed in the following table.

Command	For details, see …	Action
Add	Adding a Relational View	Displays the Relational Views Wizard, to enable you to add a new relational view
Remove	Removing a Relational View	Deletes the selected relational view
Change	Changing a Relational View	Displays the Relational Views Wizard, to enable you to maintain the selected relational view
Print	Printing a Relational View	Outputs your relational view information to the printer
Extract	Extracting a Relational View	Extracts the selected relational view
Load	Loading a Relational View	Loads a relational view from a file

Adding a Relational View

Add a relational view to the current schema from the Relational menu in the Relational Views Browser.

Note You cannot add a relational view with the same name as an RPS mapping to a schema.

To add a relational view

- Select the **Add** command from the Relational menu.

The Relational View Wizard is then displayed.

For details, see the following subsections.

Naming Your Relational View

The Relational View Wizard enables you to first define the name of your new relational view.

To specify a name for your relational view

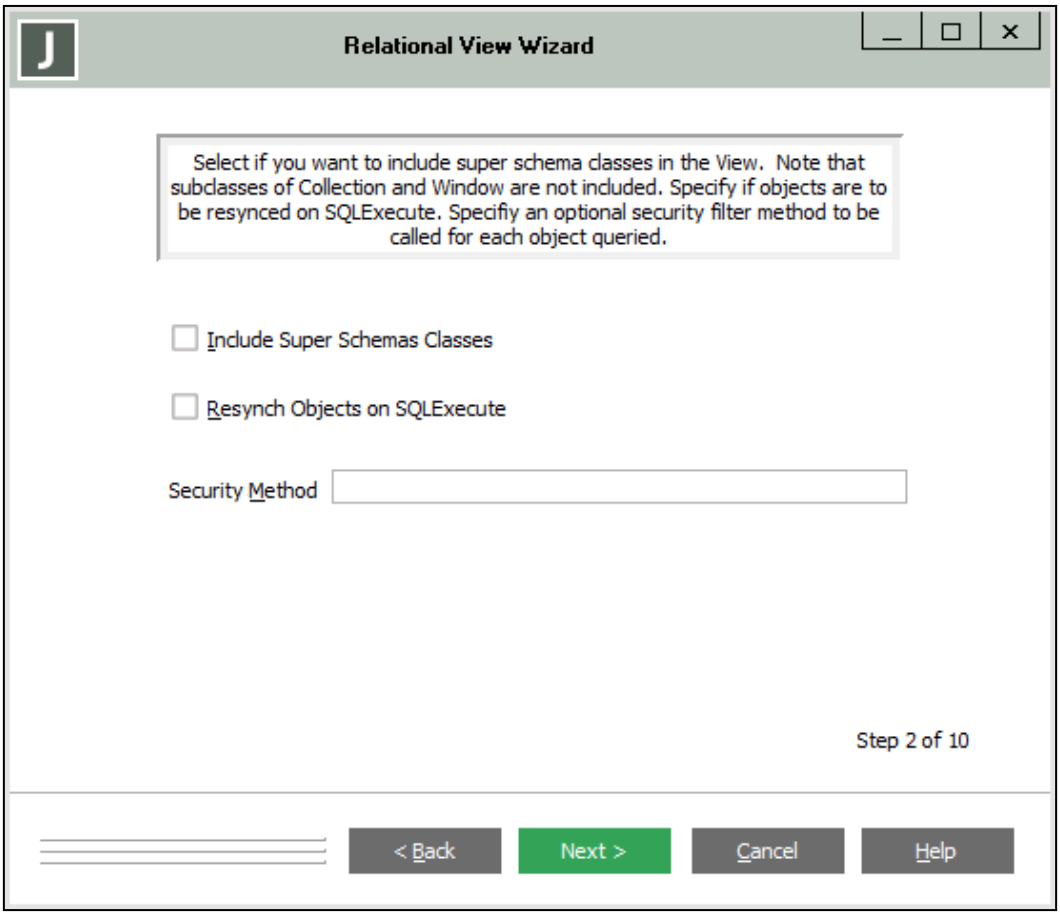
1. In the **Relational View** text box, specify the name of your new relational view.
2. Click the **Next >** button when you have defined your new view. Alternatively, click the **Cancel** button.

When you click the **Next >** button, the Relational View Wizard then enables you to specify if superschema classes are to be included in your relational view.

Specifying Relational View Options

When you have defined the name of your new relational view, the Relational View Wizard then enables you to specify the inclusion of superschema classes, the synchronization of objects when SQL is executed, and an optional security filter method to screen which objects are visible to which users.

An example of the second sheet of the Relational View Wizard is shown in the following image.



To specify your relational view options

1. Check the **Include Super Schema Classes** check box if you want to be able to include superschema classes in your relational view.

   If you select the inclusion of superschema classes, all classes defined in superschemas are included for consideration. By default, superschema classes are not visible and cannot be included in the relational view.

   Note Subclasses of the **Collection** class and **Window** class are not included in your relational view.

2. The optimal way to manage the cache is to use cache coherency in the ODBC node. If cache coherency is enabled, the **Resync Objects on SQLExecute** option is ignored.

   If you are not using cache coherency, you can check the **Resync Objects on SQLExecute** check box if you want objects used in ODBC relational views synchronized when SQL is executed.

3. If you want to specify a security filter method that is called before an object is exposed, specify the name of the method in the **Security Method** text box.

   The security method enables you to screen the objects that are visible to specific users.

If you specify a security filter method, the Jade ODBC driver invokes this method for each object that satisfies the current query before it is added to the result set of the query. The method is passed the name of the current relational view and the name of the user is obtained from the current process object. If this method returns a value of **true**, the object is added to the result set. If the returned value is **false**, the object is ignored.

Your security filter method must first be added to the appropriate class from the Class Browser and must be visible in the schema hierarchy branch in which the relational view is defined. As the security filter method can be reimplemented in subclasses, you would most likely define it at the **Object** class level.

The following is an example of a security filter method that allows only a user named "me" to view objects using a relational view named **MyRelView**.

```
isODBCPublic(relViewName: String): Boolean;
vars
begin
    if relViewName = "MyRelView" then
        if process.userCode = "me" then
            return true;
        endif;
    endif;
    return false;
end;
```

4. Click the **Next >** button when you have specified if classes are to be included in your new view.

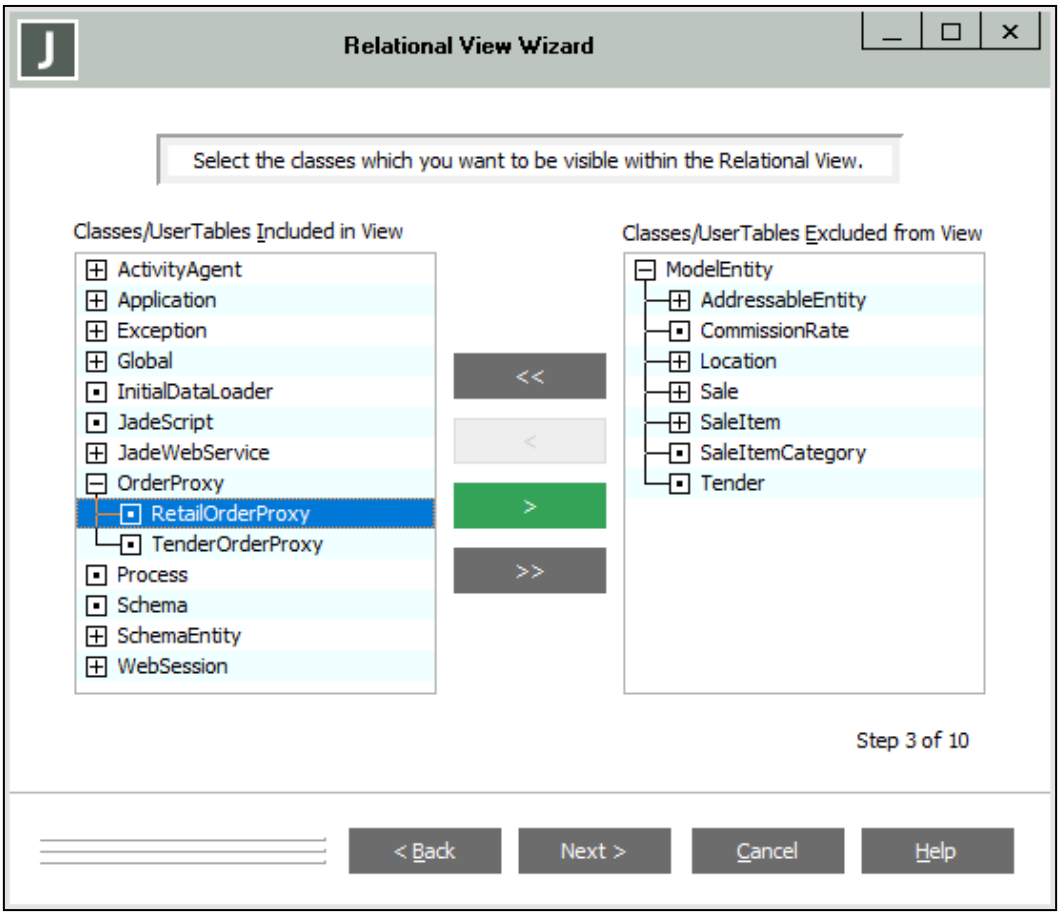
   Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to select the classes to be included in your relational view.

Selecting Classes and User-Defined Tables for Your Relational View

When you have specified if superschema classes are to be included in your relational view, the Relational View Wizard then enables you to select the classes and user-defined tables that are to be visible (included) in your relational view.

An example of the third sheet of the Relational View Wizard is shown in the following image.



Notes The included and excluded classes and user-defined tables are displayed in list boxes in their class hierarchy. If a class has subclasses, it is displayed as a collapsible and expandable node in the list box. When you perform an action on an expanded class item with subclasses, your action applies only to that class. When you perform an action on a collapsed class item with subclasses, your action is applied to that class and each of its subclasses.

To preserve the class hierarchy, a class may need to be displayed in a list box of which it is not a member. When this occurs, the item is grayed (disabled from selection), to indicate to the user that it is omitted from this list box. The **Next >** button is disabled if no classes or user-defined tables are included in the view.

User-defined tables are displayed in red at the end of a list after all of the classes.

To include a class or user-defined table in your relational view

1. In the **Classes/User Tables Excluded from View** list box at the right of the Relational View Wizard, select the class or user-defined table you want to include in your relational view and then click the **<** button to move the selected class or user-defined table to the **Classes/User Tables Included in View** list box at the left of the dialog.

   Alternatively, you can:

   - Double-click a class or user-defined table in the **Class/User Tables Excluded from View** list box to include it in your view.

- Move all classes and user-defined tables for inclusion by clicking the **<<** button.

2. Repeat step 1 for all classes and user-defined tables that you want to include in your relational view.
3. Click the **Next >** button when you have selected all of the classes and user-defined tables that are to be included in your view. (The **Next >** button is enabled when at least one class or user-defined table is included in the Relational View.)

   Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to set the root class of your relational view.

To exclude a class or user-defined table from your relational view

1. In the **Classes/User Tables Included from View** list box at the left of the Relational View Wizard, select the class or user-defined table you want to exclude from your relational view, and then click the **>** button to remove the selected class or user-defined table from your view, and move it to the **Classes/User Tables Excluded from View** list box.

   Alternatively, you can:

   - Double-click a class or user-defined table in the **Class/User Table Included in View** list box to exclude it from your view.
   - Exclude all classes and user-defined tables from your relational view by clicking the **>>** button.

2. Repeat step 1 for all classes and user-defined tables that you want to exclude from your relational view.
3. Click the **Next >** button when you have selected all classes and user-defined tables that are to be excluded from your view.

   Alternatively, click the **< Back** button to redisplay the previous dialog, or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to set the root class of your relational view.

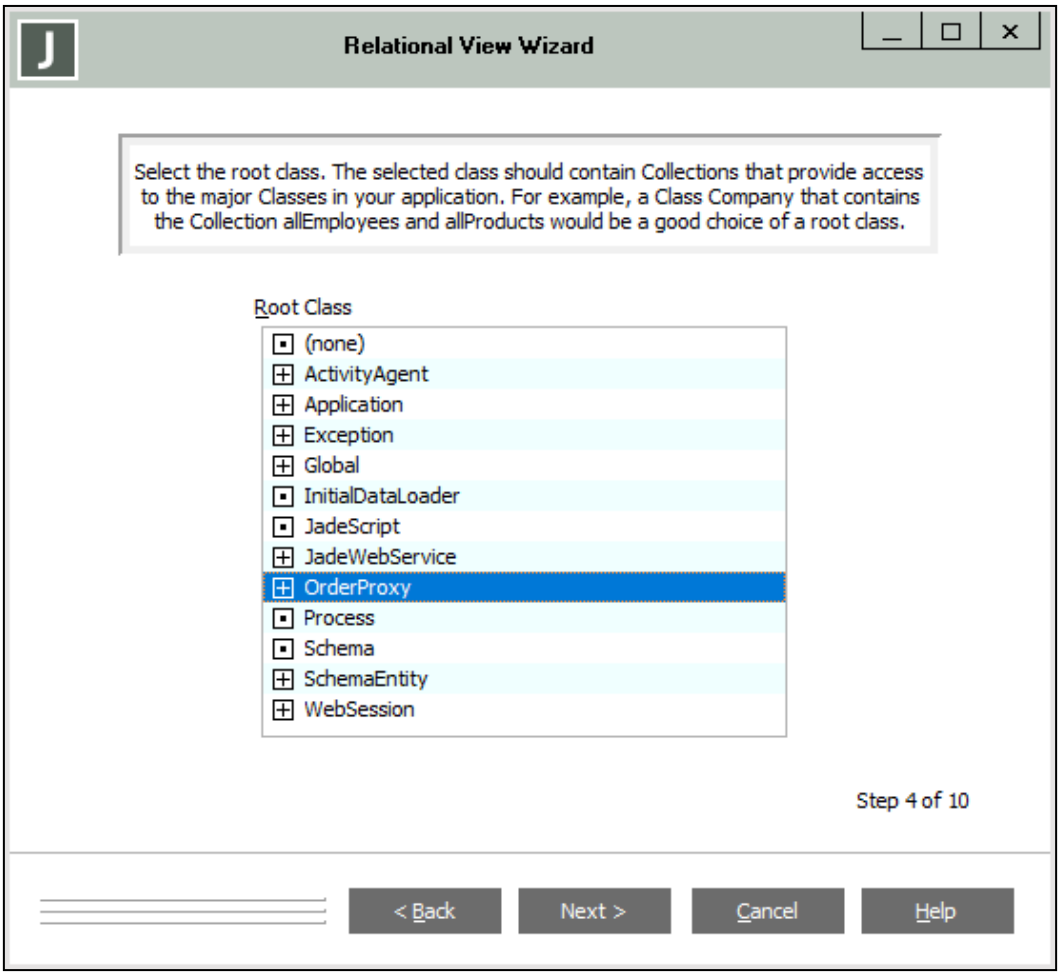
Setting the Root Class

When you have specified the classes for inclusion in your relational view, the Relational View Wizard then enables you to set the root class for your relational view.

Tip Setting a root class provides greater ODBC efficiency.

All classes selected for inclusion in your relational view are available for selection as the root class. The classes are displayed in the same hierarchy as the **Classes Included in View** list box in the previous step.

An example of the fourth sheet of the Relational View Wizard is shown in the following image.



To set your root class

1. In the **Root Class** list box, select the class that is to be the root class in your relational view. By default, no class is set as the root class.

   Select a class that contains collections that provide access to the major classes in your application; for example, the **Company** class that contains the **allEmployees**, **allProducts**, and **allCustomers** collections. Only one class can be set as the root class. You can unset a root class by selecting the **(none)** item at the top of the list.

2. Click the **Next >** button when you have set your root class.

   Alternatively, click the **< Back** button to redisplay the previous dialog, or the **Cancel** button to abandon your selections.

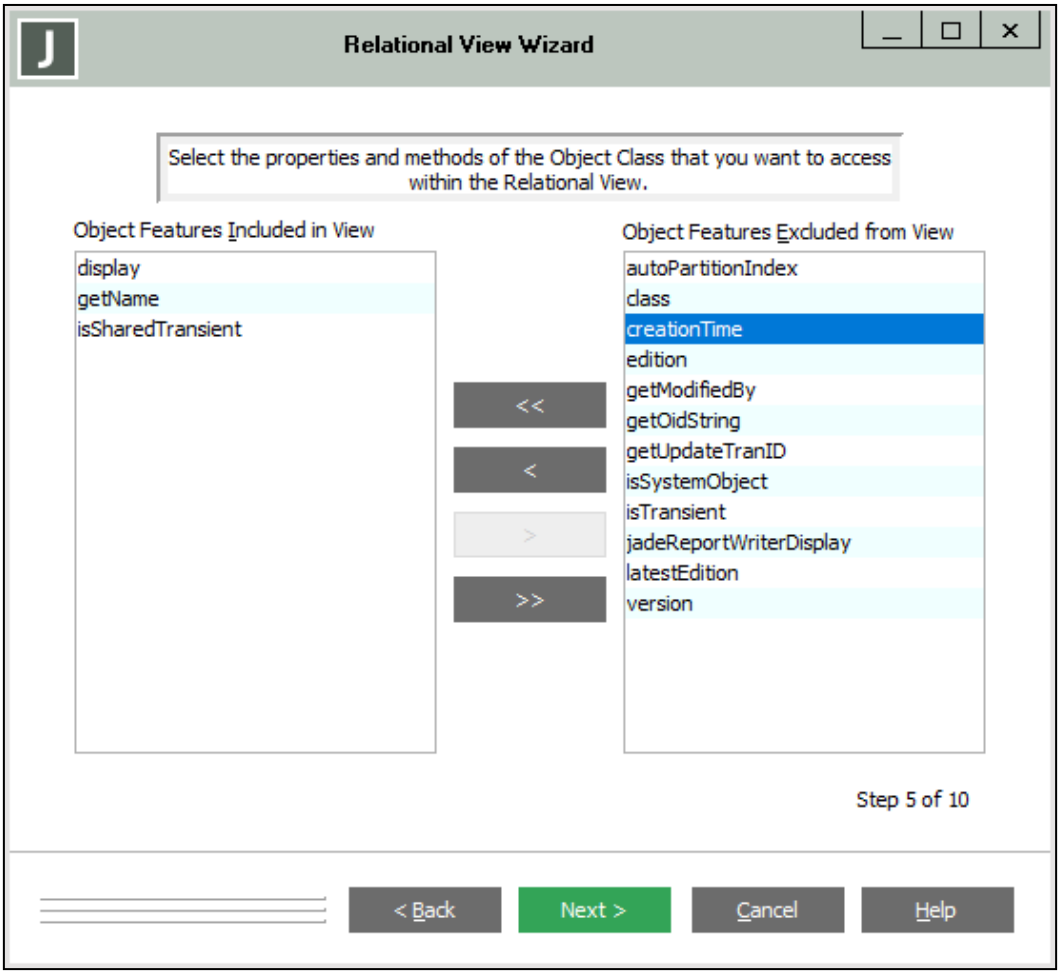
When you click the **Next >** button, the Relational View Wizard then enables you to set the included object features.

Setting the Default Included Object Features

When you have set the root class, the Relational View Wizard then enables you to set the default features (properties and methods) of the **Object** class that are to be included in your relational view. (You can refine your object features selection later in the Relational View Wizard process.)

The default **Object** class features that you set are inherited by each class in the relational view.

An example of the fifth sheet of the Relational View Wizard is shown in the following image.



To set the default Object class features for classes included in your relational view

1. In the **Object Features Excluded from View** list box at the right of the Relational View Wizard, select the object feature you want to include in your relational view and then click the **<** button to move the selected feature to the **Object Features Included in View** list box at the left of the dialog. Alternatively, you can:
   - Double-click an object feature in the **Object Features Excluded from View** list box to include it in your view.
   - Move all object features for inclusion by clicking the **<<** button.
2. Repeat step 1 for all object features that you want to include in your relational view.
3. Click the **Next >** button when you have selected all the object features that are to be included in your view. Alternatively, click the **< Back** button to redisplay the previous dialog, or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to set the visibility of protected features.

To exclude an included Object class feature from the relational view

1. In the **Object Features Included from View** list box at the left of the Relational View Wizard, select the object feature you want to exclude from your relational view and then click the **>** button to remove the selected object feature from your view and move it to the **Object Features Excluded from View** list box.

   Alternatively, you can:

   - Double-click an object feature in the **Object Features Included in View** list box to exclude it from your view.
   - Exclude all object features from your relational view by clicking the **>>** button.

2. Repeat step 1 for all features that you want to exclude from your relational view.
3. Click the **Next >** button when you have selected all object features that are to be excluded from your view.

   Alternatively, click the **< Back** button to redisplay the previous dialog, or the **Cancel** button to abandon your selections.

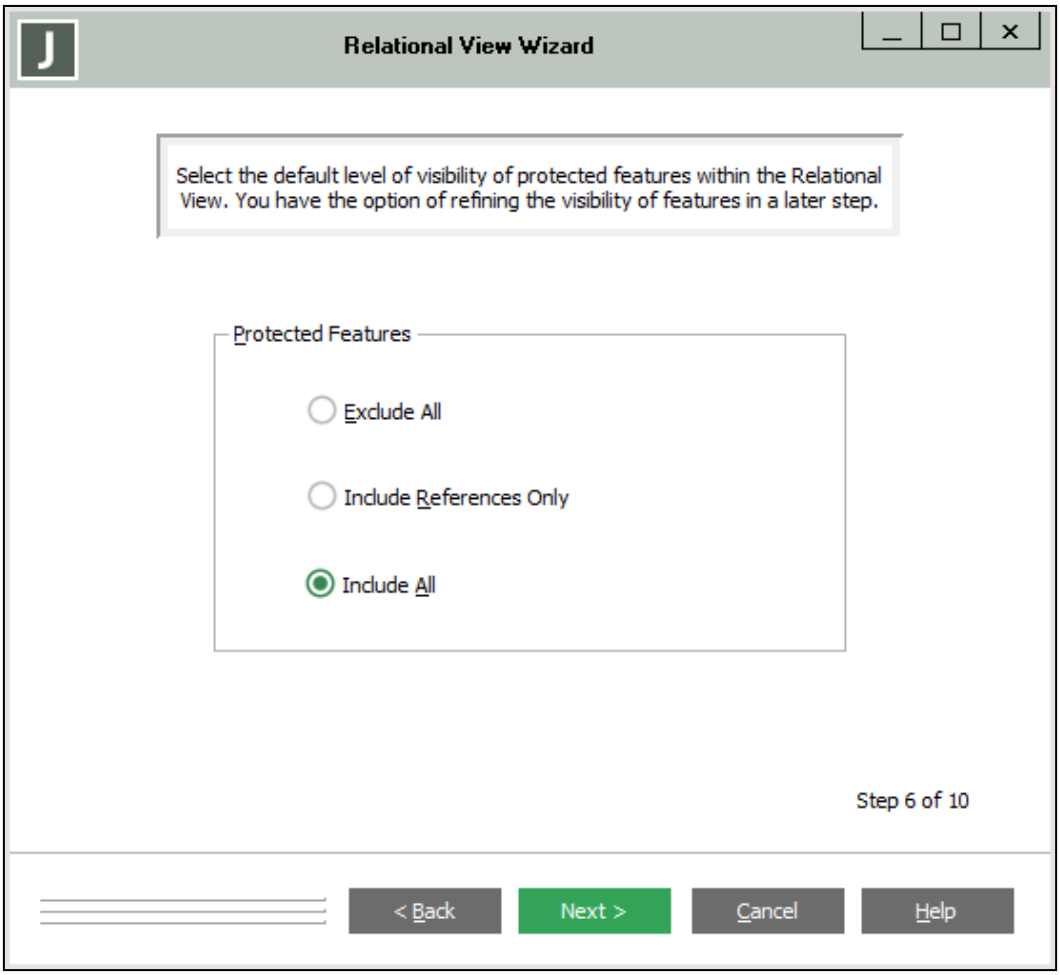
When you click the **Next >** button, the Relational View Wizard then enables you to set the visibility of protected features in your relational view.

Setting the Visibility of Protected Features

When you have set the included object features, the Relational View Wizard then enables you to set the default level of visibility of protected features in your relational view.

Note If you reset the level of visibility of protected features for existing and modified views, all refinements of that view are then lost. For large systems, this can cause a delay while the new defaults are applied across the view.

An example of the sixth sheet of the Relational View Wizard is shown in the following image.



To set the default level of visibility of protected features

1. In the Protected Features group box, select the **Include References Only** option button if you want to exclude all protected features other than those that are references of a class or select the **Include All** option button if you want to include all protected features within your relational view. By default, all protected features are excluded from your relational view.

2. Click the **Next >** button when you have set the default visibility of protected features in your view. Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to set the visibility of derived tables.

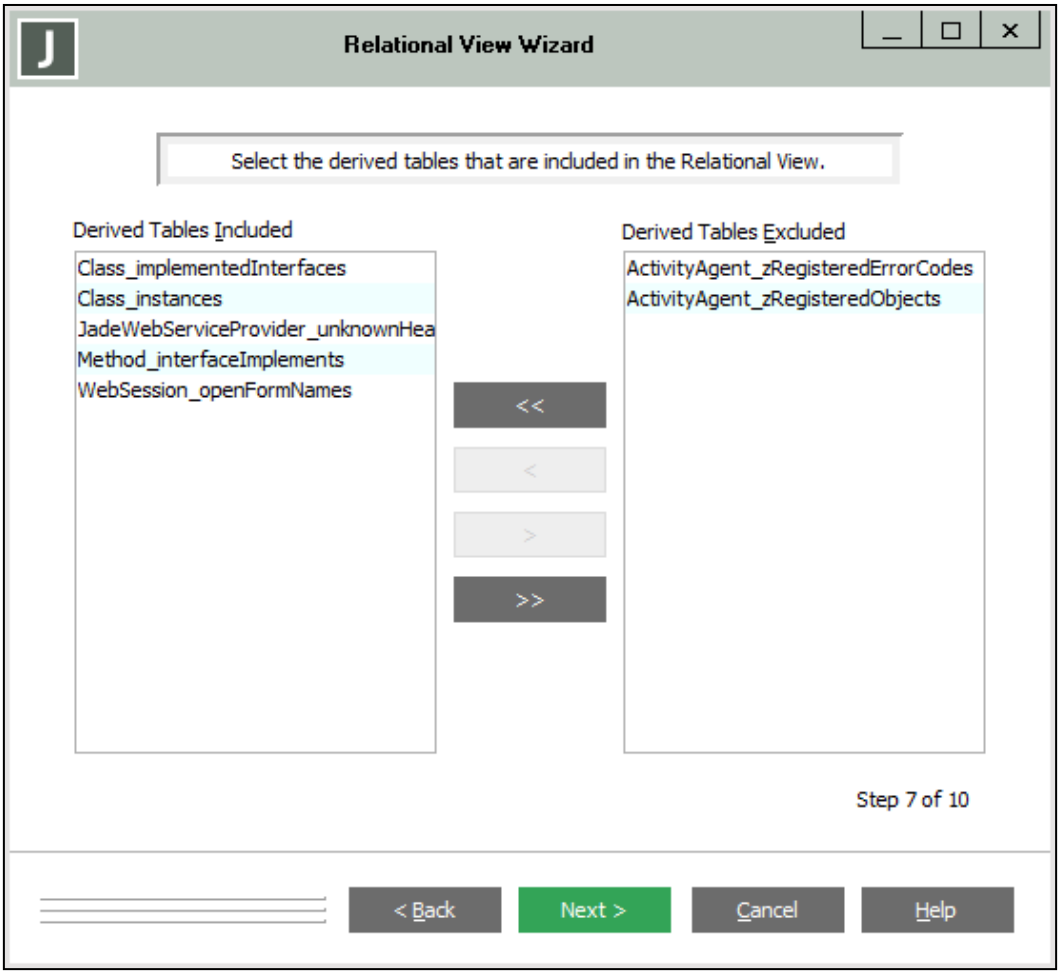
Setting the Visibility of Derived Tables

When you have set the default visibility of protected features, the Relational View Wizard then enables you to set the visibility of derived tables.

Derived tables are tables that result from the inclusion of classes that have collections and references. Derived tables are automatically generated with the following nomenclature.

class-name_property

An example of the seventh sheet of the Relational View Wizard is shown in the following image.



To include a derived table in your view

1. In the **Derived Tables Excluded** list box at the right of the Relational View Wizard, select the derived table you want to include in your relational view and then click the **<** button to move the selected table to the **Derived Tables Included** list box at the left of the dialog.

   Alternatively, you can:

   - Double-click a table in the **Derived Tables Excluded** list box to include it in your view.
   - Move all derived tables for inclusion by clicking the **<<** button.

By default, the **Derived Tables Included** list box displays tables when the setting of the protected visibility of features is as follows.

- When the default level of protection is **Exclude All**, all derived tables where access to the collection is public.
- When the default level of protection is **Include References Only**, all derived tables where access to the collection is public or protected.
- When the default level of protection is **Include All**, all derived tables are displayed.

2. Repeat step 1 for all derived tables that you want to include in your relational view.
3. Click the **Next >** button when you have selected all the derived tables that are to be included in your view.

   Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to refine the visible features in your relational view.

To exclude a derived table from your view

1. In the **Derived Tables Included** list box at the left of the Relational View Wizard, select the derived table you want to exclude from your relational view and then click the **>** button to remove the selected table from your view and move it to the **Derived Tables Excluded** list box. Alternatively, you can:
   - Double-click a table in the **Derived Tables Included** list box to exclude it from your view.
   - Exclude all derived tables from your relational view by clicking the **>>** button.

   By default, the **Derived Tables Excluded** list box displays tables when the setting of the protected visibility is **Exclude All**, and access to the collection is protected.
2. Repeat step 1 for all tables that you want to exclude from your relational view.
3. Click the **Next >** button when you have selected all derived tables that are to be excluded from your view.

   Alternatively, click the **< Back** button to redisplay the previous dialog, or the **Cancel** button to abandon your selections.

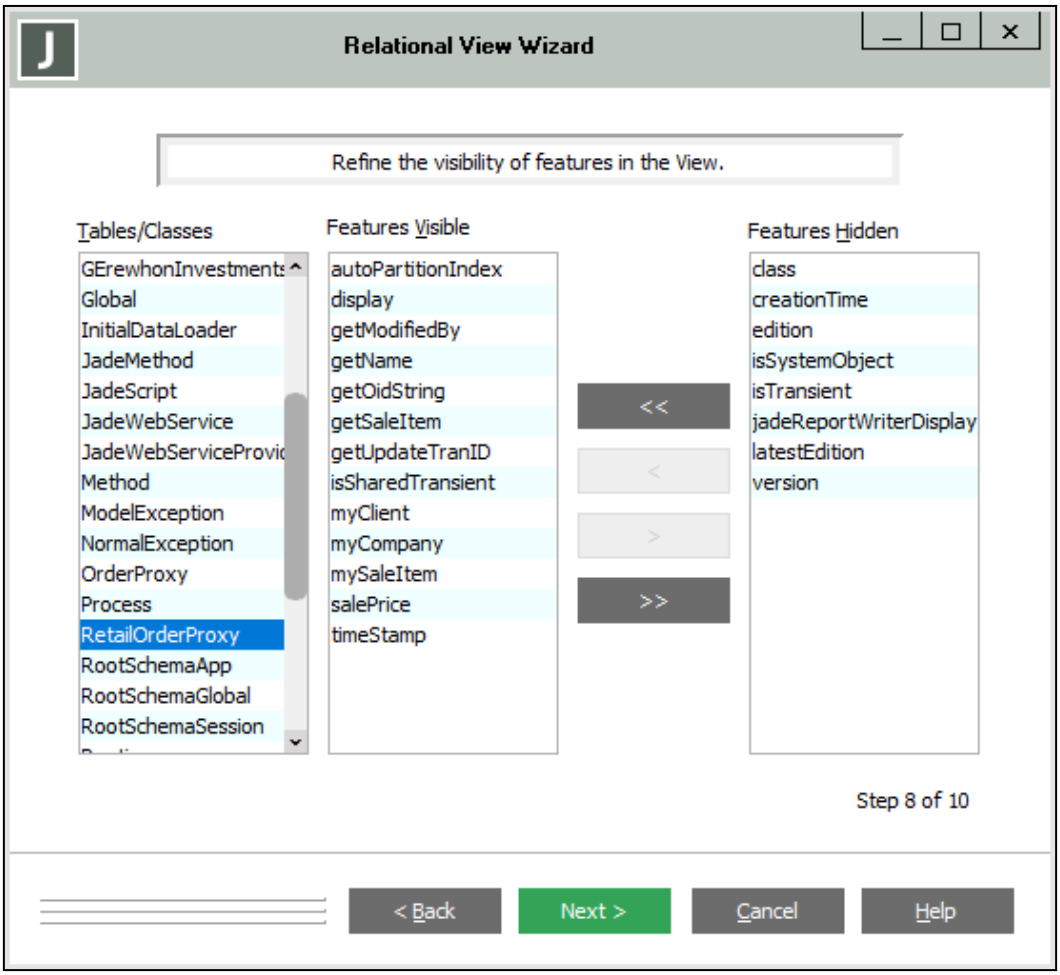
When you click the **Next >** button, the Relational View Wizard then enables you to refine the visibility of features in your relational view.

Refining the Visibility of Class Features

When you have set the visibility of derived tables, the Relational View Wizard then enables you to refine the visibility of class and user-defined table features for each table and class. All classes and user-defined tables that you selected for inclusion in your relational view are displayed in the **Tables/Classes** list box.

As you select a table or class, the class or user-defined table features (properties or methods) of that table or class are displayed in the **Features Visible** or **Features Hidden** list box, according to the default visibility for protected features that you specified, including any previous refinements.

An example of the eighth sheet of the Relational View Wizard is shown in the following image.



Note User-defined tables are displayed in red at the end of a list after all of the classes.

To refine the visibility of features for classes or tables

1. In the **Tables/Classes** list box, select the class or user-defined table whose visibility of class features you want to further refine. (User-defined tables are displayed in red after the classes.)
2. In the **Features Visible** list box, select any feature that you want to exclude from your relational view and then click the **>** button to move the selected object feature to the **Features Hidden** list box. Alternatively, you can:
   - Double-click a feature in the **Features Visible** list box to exclude it from your view.
   - Move all visible class or user-defined table features for exclusion by clicking the **>>** button.

   No class or user-defined table features are included in this list box until you have selected a table or class in the **Tables/Classes** list box. The **Features Visible** list box then displays all class or user\--defined table features according to your specified protected features default visibility, including any previous refinements.

3. In the **Features Hidden** list box, select any feature that you want to include in your relational view and then click the **<** button to move the selected class or user-defined table feature to the **Features Visible** list box. Alternatively, you can:

   - Double-click a feature in the **Features Hidden** list box to include it in your view.
   - Move all class or user-defined table features for inclusion by clicking the **<<** button.

   No class or user-defined table features are included in this list box until you have selected a table or class in the **Tables/Classes** list box. The **Features Hidden** list box then displays all class or user-defined table features for the selected table or class that you previously selected for exclusion, according to your specified protected features default visibility.

4. Repeat steps 1 through 3 of this instruction for each table and class whose visibility of class or user-defined table features in your relational view you want refine.

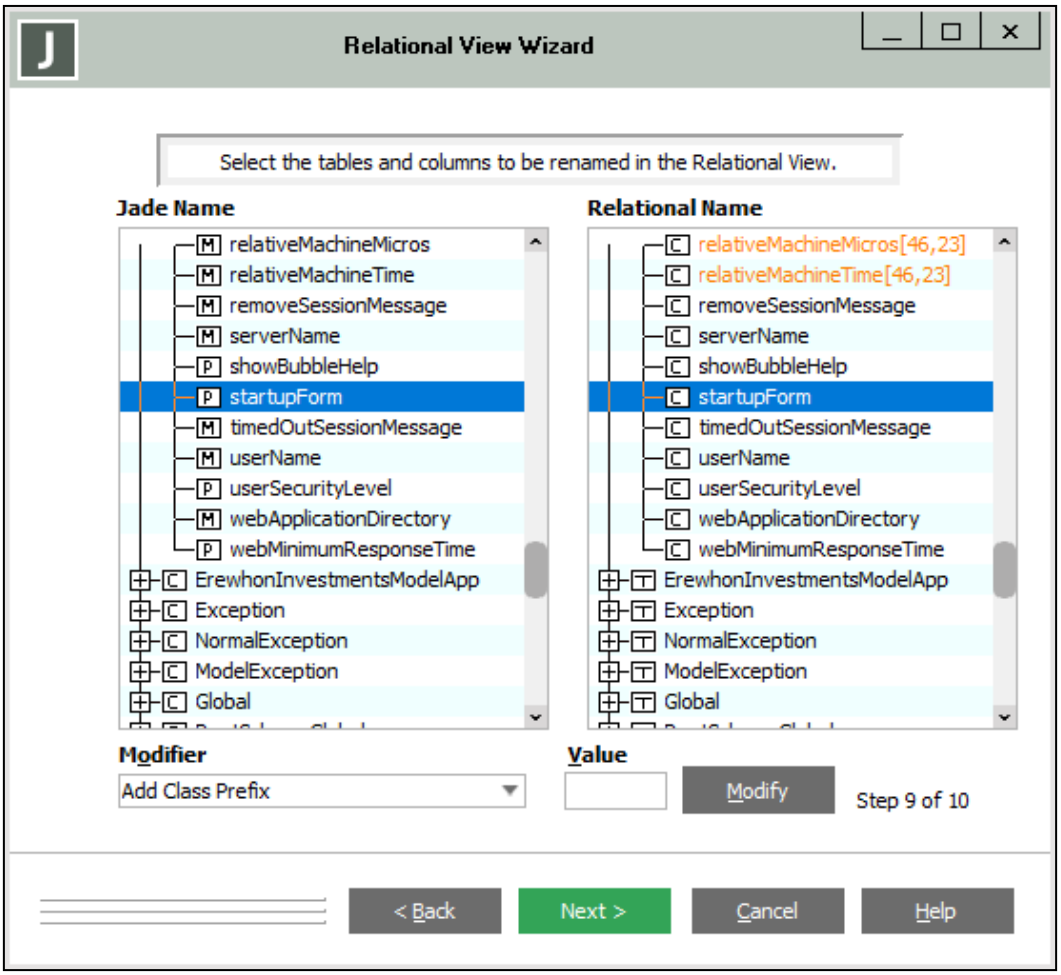
5. Click the **Next >** button when you have refined the visibility of all class or user-defined table features in the required tables or classes. Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to rename tables and columns in your relational view.

Renaming Tables and Columns

When you have refined the visibility of class or user-defined table features, the Relational View Wizard then enables you to rename tables and columns in the relational view; for example, if a class name conflicts with an ODBC reserved word or to define the relational column size for methods that return decimal values.

An example of the ninth sheet of the Relational View Wizard is shown in the following image.



The Jade name of all classes and user-defined tabled and of all properties and methods in those classes and user-defined tables included in the relational view are listed at the left of the sheet. The relational names of the corresponding tables and column are listed at the right of the sheet. In the relational names pane, any relational view names that are:

- ODBC reserved words are highlighted in yellow
- Not the same as the Jade class, method, or property name are displayed in an orange font
- Methods that return decimal values display the decimal designation

You can expand all class or table nodes to show all of their method and property entities or mapped columns, respectively, or you can collapse them to show only the class or table name itself. An expanded node is displayed with a minus sign (**-**) and a collapsed node that has entities displays a plus sign (**+**). A node that has no entities has no sign.

Alternatively, select the **Collapse All Classes** or **Expand All Classes** command from the popup menu that is displayed when you right-click in the **Jade Name** or the **Relational Name** list box.

To rename table and column relational view names

1. In the **Relational Name** list box, double-click the specific table or column whose relational name you want to change. Alternatively, select the **Edit Relational Mapping Name** command from the popup menu that is displayed when you right-click on an entity.

   The entity is then displayed in the editor bar, to enable you to change the name to the required value and then press Enter (if you do not want to change the values of any more entities).

   Tips By default, the editor bar is not hidden after each update; that is, the editor bar remains visible after updating an entity so that you can perform multiple edit actions by scrolling up or down the list box by using the up (↑) and down (↓) arrow keys. The List Editor message box is then displayed, prompting you to click the **Yes** button if you want to update the selected values or the **No** button if you want to abandon your changes. The editor bar is then positioned on the next entity, and stays visible until you change focus. To hide the display of the editor bar after updating an entity, select the **Hide Editor After Each Update** command from the popup menu that is displayed when you right-click on an entity.

   You can change the color of the editor bar by double-clicking the colored portion at the left of the editor bar (which is orange by default) and then selecting the color that you require from the common Color dialog that is displayed. When you click the **OK** button in the Color dialog, the editor bar is then displayed in your selected color.

2. If you want add a class, method, or property prefix to *all* relational view tables, method columns, or property columns, perform the following actions.

   a. Ensure that the appropriate **Add Class Prefix**, **Add Method Prefix**, or **Add Property Prefix** value is displayed in the **Modifier** list box.

   b. Specify the value in the **Value** text box (for example, **test_**) that you want to apply to all tables or columns in the relational view that are mapped to user-defined tables or Jade classes, methods, or properties.

   c. Click the **Modify** button.

   Note If you have multiple schema and relational view pairs defined for your ODBC Data Source Name (DSN), you should perform these steps to add a class or user-defined table prefix so that there are no name conflicts in the included relational views.

3. To remove a class, method, or property prefix from *all* relational view tables, method columns, or property columns, perform the following actions.

   a. Ensure that the appropriate **Remove Class Prefix**, **Remove Method Prefix**, or **Remove Property Prefix** value is displayed in the **Modifier** list box.

   b. In the **Value** text box, specify the user-defined table, class, method, or property prefix you want to remove from the relational view tables or columns (for example, **prod_**).

   c. Click the **Modify** button.

4. As methods that return decimals have the decimal designation following the relational view column name, you can edit this to change the length and scale factor with which the column for that method is to be defined. For example, you can edit the **getMyDecimal[46,32]** method (the default declaration that includes all possible decimal values) to **getMyDecimal[12,2]**.

5. Repeat steps 1 through 4 of this instruction for each table or column that you want to edit.

6. Click the **Next >** button when you have edited all required relational view entities. Alternatively, click the **< Back** button to redisplay the previous sheet or the **Cancel** button to abandon your selections.

When you click the **Next >** button, the Relational View Wizard then enables you to build (generate) your relational view.

Building Your Relational View

When you have renamed all required entities, you have finished specifying your relational view. The Relational View Wizard then enables you to build (generate) your relational view. Use the **< Back** button to review your selections, if required.

To build your relational view

- Click the **Finished** button when you have finished specifying your relational view. Alternatively, click the **< Back** button to redisplay the previous dialog or the **Cancel** button to abandon your selections.

When you click the **Finish** button, Jade then begins to build your relational view. A progress dialog displays the progress of the relational view build.

When the build is complete, the Relational Views Browser is then displayed with your new relational view. When you select the relational view in the Relational Views Browser, the status line displays the creation timestamp and your user identifier. You can now access your relational view by using the ODBC driver and your SQL inquiry programs.

Removing a Relational View

From the Relational Views Browser, the **Remove** command from the Relational menu enables you to remove (delete) the relational view that is currently selected.

Note You cannot remove a relational view that is currently being accessed by an ODBC driver.

To remove a relational view

1. In the Relational Views Browser, select the relational view that you want to remove.
2. Select the **Remove** command from the Relational menu.

   A message box is then displayed, to enable you to confirm that you want to remove the specified relational view.
3. Click the **OK** button to confirm that the selected relational view is to be removed.

   Alternatively, click the **Cancel** button to abandon the deletion.

The Relational Views Browser is then updated to reflect the removal of the selected relational view. There may be a momentary delay while this updating occurs.

Printing a Relational View

From the Relational Views Browser, the **Print** command from the Relational menu enables you to output information about the currently selected relational view to your printer.

To print your relational view

1. In the Relational Views Browser, select the relational view that you want to print.
2. Select the **Print** command from the Relational menu.
3. The Printing progress dialog is then displayed. (You can click the **Cancel** button from the Printing progress dialog to cancel your print request.)

Your print output contains details of your relational view, and details of each of the relational view tables and their columns.

Changing a Relational View

You can modify an existing relational view in the current schema. When you modify a relational view, you cannot change the relational view name.

To change a relational view

- Select the **Change** command from the Relational menu.

The Relational View Wizard is then displayed. For details about the Relational View Wizard steps, see "Adding a Relational View", earlier in this chapter.

When you are changing an existing relational view, the text box in the first dialog of the Relational View Wizard displays the name of your relational view.

As you cannot change the name of an existing relational view, the text box is disabled.

Extracting a Relational View

You can extract a relational view as part of the schema in which it is defined or you can extract only the relational view itself.

User-defined tables are not extracted when the schema or relational view is extracted, because user-defined entities are dependent on user data (not schema meta data) and cannot be loaded into other Jade systems. Any user-defined tables must be:

- Created by executing your own user code for all Jade systems, as required.
- Re-created if a full schema load or relational view load is performed.

To extract a relational view only

1. In the Relational Views Browser, select the relational view that you want to extract.
2. Select the **Extract** command from the Relational menu. The Extract dialog is then displayed.

For details about using the Extract dialog and extracting the relational view as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10.

Loading a Relational View

You can load a relational view as part of the schema in which it is defined or you can load only an extracted relational view itself.

See "Before You Get Started", in the *Schema Load User's Guide*, for details about the order in which to load separate relational view extract files and selective schema extract files into a deployed database.

To load a relational view only

1. Select the **Load** command from the Relational menu. The Load Options dialog is then displayed.
2. In the **Schema File Name** text box, specify the name and location of the relational view schema file (**.scm** file) that you want to load. You must specify a value in this text box. If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.

3. In the **Forms File Name** text box, specify the name and location of the extracted form and data definition (**.ddb** or .**ddx**) file that you want to load. If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.

4. Click the **OK** button to confirm your selections. Alternatively, click the **Cancel** button to abandon your selections.

The load of your relational view is then initiated. For details about loading the relational view as part of the schema in which it is defined, see "Loading Your Schema", in Chapter 10.

Maintaining Ad Hoc Indexes

Ad hoc indexes, which enable you to create indexes suitable for optimizing ad hoc queries without requiring database reorganization, are defined as a subclass of **DynaDictionary**, with instances mapped to a RootSchema map file called **_sindexes.dat**.

The **_sindexes.dat** database file is partitioned (for example, **_sindexes_part0000000001.dat**), with each ad hoc index instance created in a new partition to facilitate a fast drop index operation, which uses the existing **dropPartition** operation. The classes that represent index meta data are mapped to the **_sindexdefs.dat** RootSchema database file. (The **_sindexes.dat** and **_sindexdefs.dat** files are of **Kind_User_Data**.)

The [JadeAdHocIndex] section in the Jade initialization file enables you to specify options for the worker applications that build, drop, and delete an ad hoc index, and for the controller application that starts worker applications when there is an ad hoc index maintenance operation to be performed. For details about the **BuildCommitPeriod** and **MaxBuildWorkers** parameters, see "Jade Ad Hoc Index Section [JadeAdHocIndex]", in the *Jade Initialization File Reference*.

An Index Writer application reads from the main database (or from journals when catch-up is required), and creates and maintains the user-defined index. The ODBC driver function that performs ODBC queries can then use a combination of ad hoc indexes and collections in the main database to satisfy queries.

The Ad Hoc Index Browser enables you to:

- Create or modify ad hoc index definitions.
- Drop unwanted indexes.
- Save an ad hoc index to an Extensible Markup Language (XML) file. As the ad hoc index definition does not define any meta data, the data is not included when the schema is extracted.
- Load a saved ad hoc index.

For details about using the **jadclient** executable to create indexes suitable for optimizing ad hoc queries without requiring database reorganization, see "Ad Hoc Index Batch Interface", in Chapter 1 of the *Runtime Application Guide*.

Following the hostile takeover of a secondary database, any populated ad hoc indexes *may* be left in an inconsistent state with the remainder of the database, as some journals may not have been sent from the primary database and processed before the takeover operation. As existing ad hoc indexes are therefore marked as in error, drop the indexes and then build them, to clear the error state and make them consistent with the state of the rest of the database. To do this, from the Ad Hoc Index menu in the Ad Hoc Index Browser on the primary database:

1. Select the **Run Ad Hoc Index Controller App** command.
2. Select the index definition that has a red background in the **Status** column, indicating that it is in error.
3. Select the **Drop Index** command.

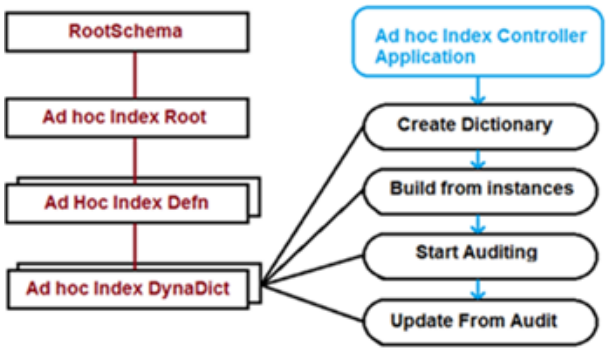
A confirmation message box is then displayed stating the number of instances of that index and prompting you to confirm that you do want to drop (unpopulate) it. Click the **Yes** button to confirm that the selected ad hoc index definition is to be dropped.

4. In the Ad Hoc Index Browser table, select the dropped unpopulated index definition that you want to build.
5. Select the **Build Index** command.

Defining an Ad Hoc Index

The use of query indexes removes the need to reorganize the main database when new indexes are created or dropped. However, as changes to the production database could impact defined indexes, changes to a class and its properties are decoupled from the ad hoc index definition and no checking is done against the ad hoc indexes before allowing a schema change.

The following diagram illustrates the ad hoc index structure and process.



Before an ad hoc definition is used, it is validated against the existing schema and class definitions. If the definition is no longer valid or the key definitions have changed, the ad hoc index is ignored in regard to maintenance of the collection entries and for any ODBC use. If the ad hoc index has become invalid and has been built, the dictionary must be first dropped and the definition must be updated or at least saved again, to verify that the definition structure is correct.

The Ad Hoc Index Browser uses a red background in the index status column and to indicate that an index has become invalid or that it needs attention. This red background occurs on the indexes status column and the column elements (membership class or keys) that need attention. Text also explains what has changed and what action is required to restore the index.

All access to the schema, membership class, and properties is performed by name.

At the end of instantiation (validation) during a reorganization, the reorganization application examines all indexes and performs definition validation for all online indexes and marks any that are broken.

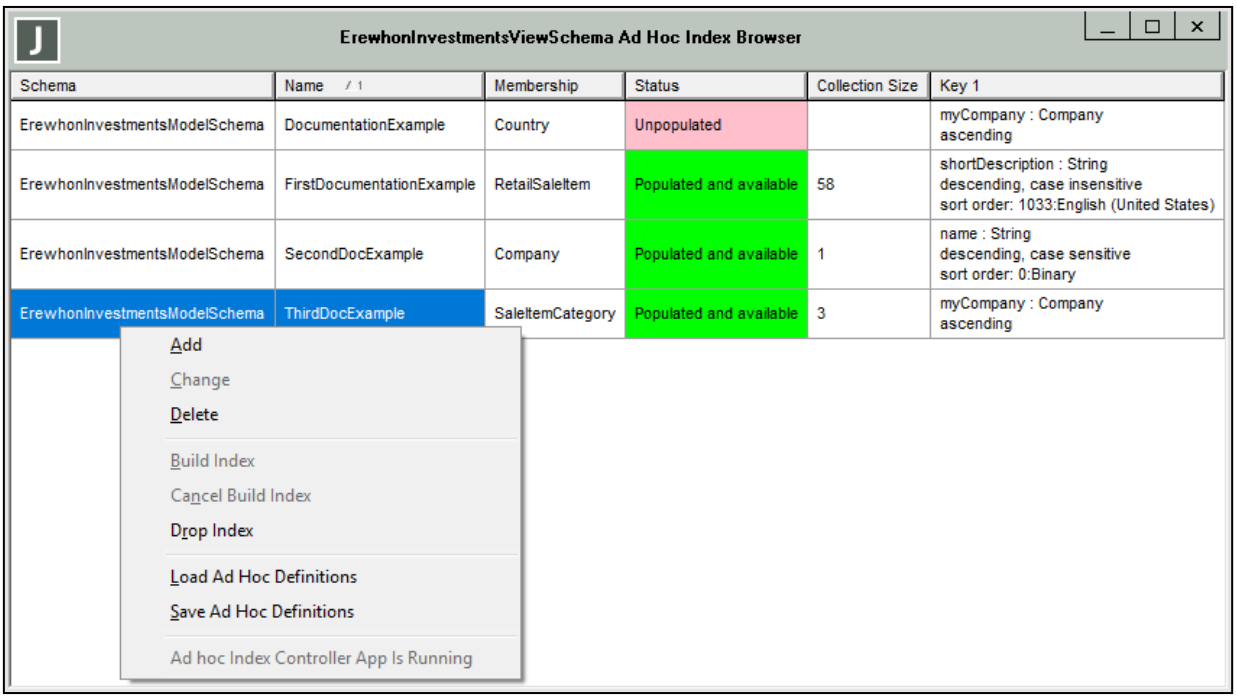
Use the Browse menu **Ad Hoc Indexes** command from a user-defined schema to define an ad hoc index for the current schema. Each schema within the Jade database can have a collection of ad hoc indexes.

Note The Relational Views Browser enables you to maintain your relational views only. To access the Jade data in the relational view using SQL select statements, you must first write your own inquiry programs. For details, see "SQL Examples", in Chapter 2 of the *External Interface Developer's Guide*.

To open an Ad Hoc Indexes Browser window

- Select the **Ad Hoc Indexes** command from the Browse menu

An Ad Hoc Indexes Browser window, shown in the following image, is then opened. If you have not yet defined an ad hoc index or loaded an existing ad hoc index definition, nothing is displayed in the Ad Hoc Index Browser.



Each ad hoc index in the table displays the:

- Schema name.
- Ad hoc index name.
- Member class name and the number of subclasses in the schema of the membership class that will be included, with a list of class names (with **… more**, if there are more than 10 lines).
- Current status, indicated by the following background colors.
  - Red (the index is invalid or needs confirmation of key property attribute changes).

    Other columns can also have a red background; for example, when a key property name is not found.
  - Light yellow (an action such as the cancellation of a build, an index drop, or an index deletion is in progress).
  - Yellow (an initial build is in progress).
  - Green (the index has been built and is available for ODBC use).
  - Light pink (the index is unpopulated and no action is in progress).
- The collection size, if available.
- Each defined key and the attributes of each key.

Note To change or manipulate an existing index, select the index row in the browser.

Using the Ad Hoc Index Menu

The Ad Hoc Index Browser provides the Ad Hoc Index menu, to enable you to maintain your ad hoc index definitions. The Ad Hoc Index menu contains the commands listed in the following table.

Command	For details, see …
Add	Adding an Ad Hoc Index
Change	Changing an Ad Hoc Index
Delete	Deleting an Ad Hoc Index
Build Index	Building an Ad Hoc Index
Cancel Build Index	Canceling an Ad Hoc Index Build
Drop Index	Dropping an Ad Hoc Index
Load Ad Hoc Definitions	Loading Ad Hoc Definitions
Save Ad Hoc Definitions	Saving Ad Hoc Definitions
Run Ad Hoc Index Controller App	Running the Ad Hoc Index Controller Application

Adding an Ad Hoc Index

Add an ad hoc index definition to the current schema from the Ad Hoc Index menu in the Ad Hoc Index Browser.

Notes You cannot add an ad hoc index with the same name as an RPS mapping to a schema.

As you cannot manipulate ad hoc indexes on a secondary (including index creation, update, build, or drop actions), all of the Ad Hoc Index Browser menu items are disabled. These actions must be performed on the primary.

To add an ad hoc index

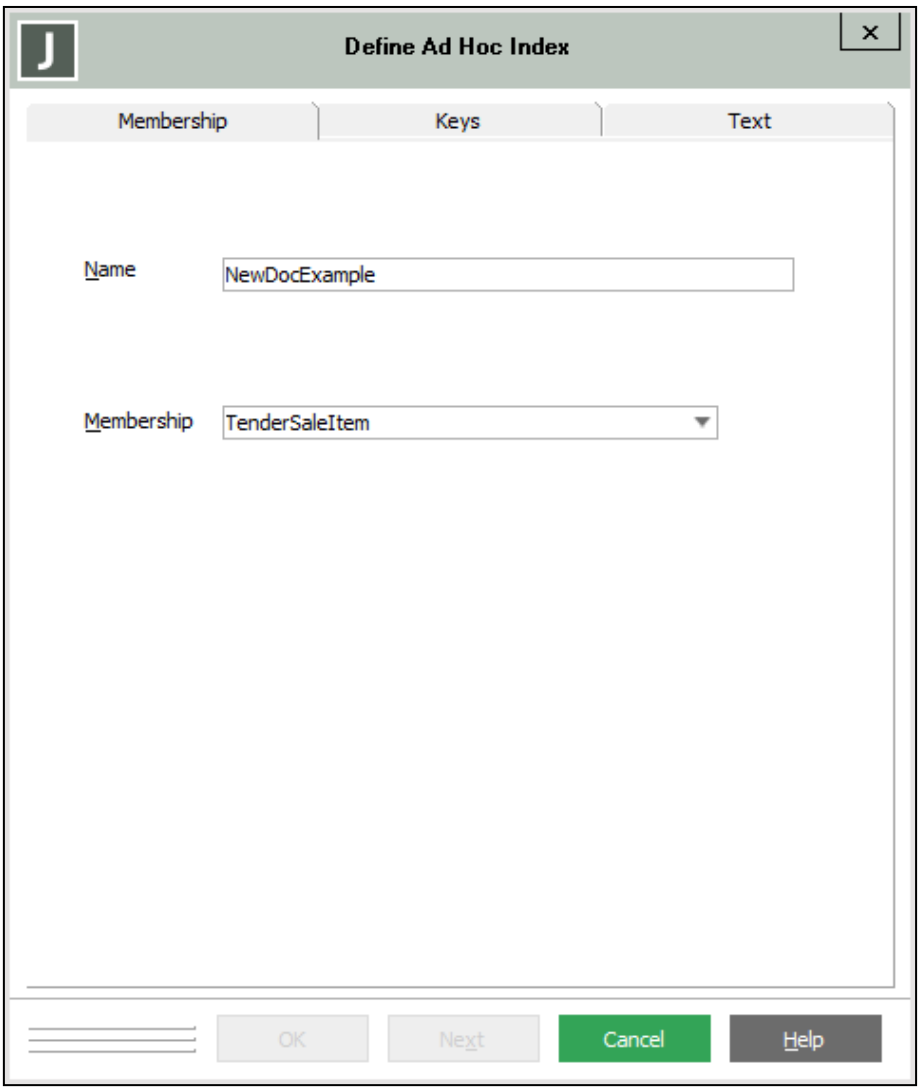
- Select the **Add** command from the Ad Hoc Index menu.

The Define Ad Hoc Index dialog is then displayed. For details, see:

- Defining Ad Hoc Index Membership
- Defining Ad Hoc Index Keys
- Defining Ad Hoc Index Text

Defining Ad Hoc Index Membership

The **Membership** sheet of the Define Ad Hoc Index dialog, shown in the following image, is displayed when the dialog is invoked, to enable you to define the name of the ad hoc index and to select its membership.



To specify a name for your ad hoc index and its membership

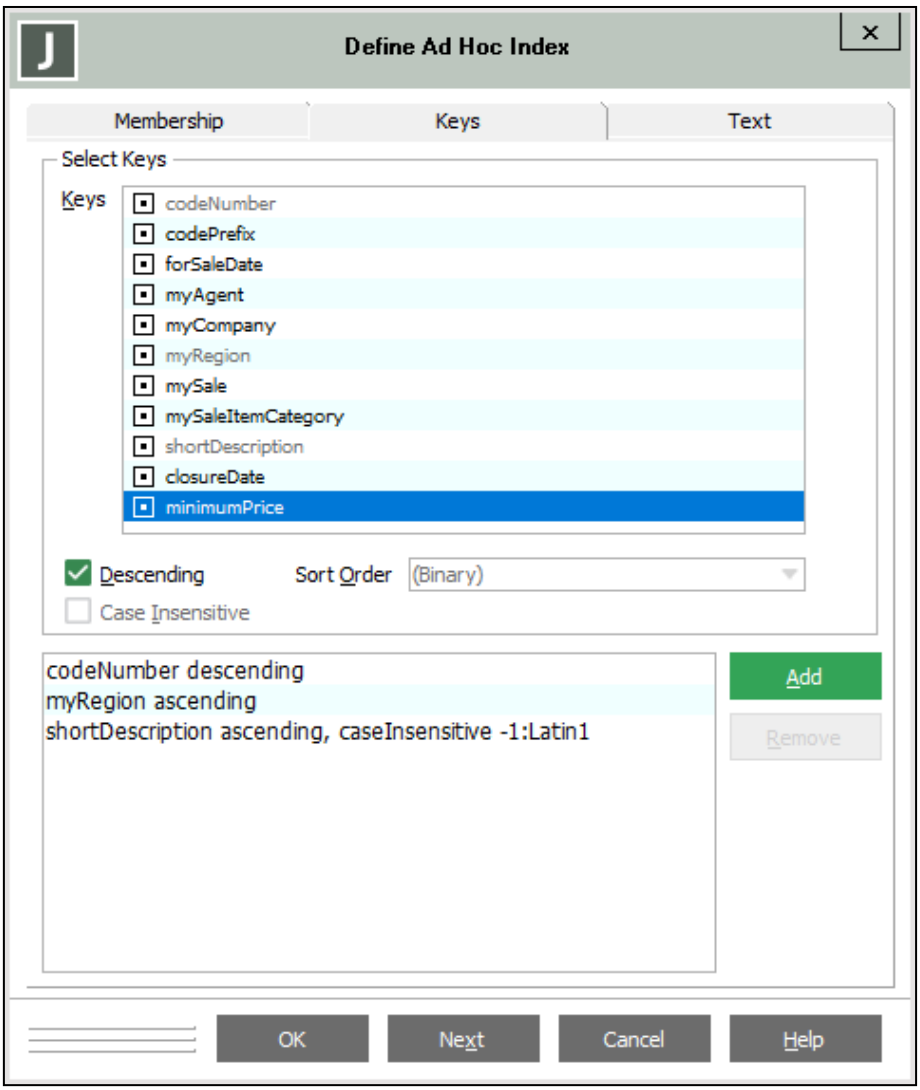
1. In the **Name** text box, specify the name of your ad hoc index. The name must be unique within the current schema and superschemas.
2. In the **Membership** combo box, specify or select the membership of the class to be used for the ad hoc index. Collection classes, transient-only classes, and classes that do not allow persistent instances are not available for selection.
3. Click the **Next >** button. Alternatively, click the **Cancel** button.

When you click the **Next >** button, the **Keys** sheet is displayed, which enables you to select the property or properties to be used for keys in the ad hoc index.

Defining Ad Hoc Index Keys

When you have defined the name and membership class of your new ad hoc index and clicked the **Next** button, the **Keys** sheet of the Define Ad Hoc Index dialog is displayed, to enable you to select the properties of that class to be used as keys in the index.

An example of the **Keys** sheet is shown in the following image.



To define the keys for your ad hoc index

1. Perform one of the following actions.
   - In the **Keys** list box, select the property that you want to use as an index key and then click the **Add** button.
   - Double-click on a property in the **Keys** list box.

   The property is then moved to the pane at the lower area of the dialog. By default, keys are sorted in ascending order and they are case-sensitive.

2. If you want to reverse the sort order, click on a property selected as a key in the pane in the lower area of the dialog and then check the **Descending** check box.
3. If the property selected in the lower area of the form is of type **String** or **StringUtf8**, check the **Case Insensitive** check box if you want the key to be case-insensitive.
4. Repeat steps 1 through 3 of this instruction until you have defined all keys that you require in the ad hoc index.
5. Click the **Next >** button. Alternatively, click the **Cancel** button.

When you click the **Next >** button, the **Text** sheet is displayed, which enables you to specify a description of the ad hoc index, if required.

Defining Ad Hoc Index Text

When you have defined the membership and keys of your ad hoc index and clicked the **Next** button, the **Text** sheet of the Define Ad Hoc Index dialog is displayed, to enable you to specify a textual description of the ad hoc index, if required.

An example of the **Text** sheet is shown in the following image.

To specify descriptive text for your ad hoc index

1. In the editor pane, enter the required descriptive text. (For details about specifying text, see "Specifying Text for a Schema Element", in Chapter 3.)

   The editor determines the behavior of the text editor window. Text is not automatically wrapped to fit the current window size. When you want text to start in the next line, press Ctrl+Enter. (If you want text to wrap in text windows, see "Maintaining Editor Options", in Chapter 2.)

2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your ad hoc index selections.

The ad hoc index definition is then validated and created, with the status of **Unpopulated**. The definition is rejected if it matches an existing ad hoc index definition.

A warning is displayed if the descending attribute varies between keys (as it may impact on the ability for the index definition to be optimized) or if another ad hoc index or an existing **MemberKeyDictionary** with the same membership and keys exists (even if they have more keys).

Note Key paths are not supported, as they cannot be directly referenced by ODBC select statements.

Changing an Ad Hoc Index

You can modify an existing ad hoc index in the current schema if it is not yet populated (that is, it has a light pink background in the **Status** column of the table in the Ad Hoc Index Browser). You cannot change ad hoc definitions if the collection has been built or it is being built.

To change a relational view

- Click in an unpopulated ad hoc index and then select the **Change** command from the Ad Hoc Index menu.

The Define Ad Hoc Index dialog is then displayed. For details about the Define Ad Hoc Index dialog, see "Adding an Ad Hoc Index", earlier in this chapter.

Deleting an Ad Hoc Index

From the Ad Hoc Index Browser, the **Delete** command from the Ad Hoc Index menu enables you to remove (delete) the ad hoc index definition that is currently selected in the table on the browser if the index is not in the process of being built.

Note You cannot delete an ad hoc index that is currently being read by an ODBC driver.

To delete an ad hoc index

1. In the Ad Hoc Index Browser, select the index definition that you want to remove.
2. Select the **Delete** command from the Ad Hoc Index menu. A message box is then displayed, to enable you to confirm that you *do* want to delete the specified index definition.
3. Click the **Yes** button to confirm that the selected ad hoc index definition is to be deleted. Alternatively, click the **No** button to abandon the deletion.

If the index is fully built, the index is dropped and then deleted. The Ad Hoc Index Browser is then updated to reflect the deletion of the selected index definition. There may be a momentary delay while this updating occurs.

Building an Ad Hoc Index

From the Ad Hoc Index Browser, the **Build Index** command from the Ad Hoc Index menu enables you to build an unpopulated ad hoc index definition that is currently selected in the table on the browser.

To build an ad hoc index definition

1. In the Ad Hoc Index Browser, select the unpopulated index definition that you want to build.
2. Select the **Build Index** command from the Ad Hoc Index menu.

The background color of the **Status** column changes to yellow while the index in a dynamic dictionary instance is being built (populated), and to green and **Populated and available** is displayed when the population of the index is complete. A new partition to be used by that instance is created.

The build process drops any existing ad hoc index dictionary and creates a new one. It turns off auditing on the new partition created for the dictionary. It then populates the defined dynamic dictionary. A commit of the transactions is performed every five seconds, by default. When the build completes, auditing is turned back on for the partition. (You can use the **Cancel Build Index** command to cancel the building of an index.)

Canceling an Ad Hoc Index Build

From the Ad Hoc Index Browser, the **Cancel Build Index** command from the Ad Hoc Index menu enables you to cancel a build that is in progress.

» To cancel the building of an ad hoc index definition

1. In the Ad Hoc Index Browser, select the index definition that is in the process of being built (that it, it has a bright yellow background in the **Status** column).
2. Select the **Cancel Build Index** command from the Ad Hoc Index menu.

The background color of the **Status** column then changes to light pink and **Unpopulated** is displayed when the partially built index has been removed from the dynamic dictionary instance.

Dropping an Ad Hoc Index

From the Ad Hoc Index Browser, the **Drop Index** command from the Ad Hoc Index menu enables you to drop the partition containing a fully built ad hoc index. This enables you to drop the contents of a partition without incurring costs of individual object deletions.

If membership of an ad hoc index returns a different property reference, the key is known to have been changed, which results in the ad hoc index being marked as not being currently valid. A built dictionary must be dropped and the definition updated (rebuilt) to confirm its use.

» To drop an ad hoc index definition from a dynamic dictionary instance

1. In the Ad Hoc Index Browser, select the index definition that is populated and available (that it, it has a green background in the **Status** column).
2. Select the **Drop Index** command from the Ad Hoc Index menu. A confirmation message box is then displayed stating the number of instances of that index and prompting you to confirm that you *do* want to drop (unpopulate) it.
3. Click the **Yes** button to confirm that the selected ad hoc index definition is to be dropped. Alternatively, click the **No** button to abandon the drop action.

The background color of the **Status** column then changes to yellow while the partition containing the index is dropped, and to pink with **Unpopulated** displayed when the index has been dropped. When the drop action is completed, the ad hoc index is deleted.

Loading Ad Hoc Definitions

From the Ad Hoc Index Browser, the **Load Ad Hoc Definitions** command from the Ad Hoc Index menu enables you to load an ad hoc index definitions from an XML file; for example, when you have upgraded to a new Jade version and loaded your schema data. For details about saving ad hoc definitions to an XML file, see "Saving Ad Hoc Index Definitions".

If the loaded definition is unchanged from an existing definition, the load is ignored.

To load ad hoc index definitions from a file

1. In the Ad Hoc Index Browser, select the **Load Ad Hoc Definitions** command from the Ad Hoc Index menu.

   The common Open dialog is then displayed, to enable you to select the ad hoc definitions file that you want to load into the current schema.

2. Select the source location and XML file that you want to load, and then click the **Open** button.

All ad hoc index definitions in the file are then loaded into the table in the Ad Hoc Index Browser. As they are all unpopulated, you must build them before they can be used. For details, see "Building an Ad Hoc Index".

Saving Ad Hoc Definitions

As the ad hoc index definition does not define any meta data, the data is not included when the schema is extracted. The Ad Hoc Index menu in the Ad Hoc Index Browser provides the **Save Ad Hoc Definitions** command, which enables you to save ad hoc index definitions to an XML file; for example, so that you can load your ad hoc definitions into a new version of Jade when you have upgraded to a new Jade version and loaded your schema dat instead of having to redefine each ad hoc index. For details about loading ad hoc index definitions into a schema, see "Loading Ad Hoc Definitions".

To save ad hoc index definitions to an XML file

1. In the Ad Hoc Index Browser, select the **Save Ad Hoc Definitions** command from the Ad Hoc Index menu.

   The common Save As dialog is then displayed.

2. Select the source location and specify the name of the XML file prefix to which you want to save the ad hoc index definitions.

3. Click the **Save** button.

All ad hoc index definitions in the file are then saved to the specified XML file. (*All* ad hoc definitions in your schema are saved to the file; that is, you cannot save one definition only.)

An example of an ad hoc index definitions XML file is shown in the following example.

```
<?xml version="1.0" ?>
<AdHocIndexDocument>
  <AdHocIndex Name="A1a2ByAmount" Schema="OdbcTestSchema" MembershipClass="A1A2">
    <Key Name="amount" Descending="false" CaseInsensitive="false" SortOrder="0"
/>
  </AdHocIndex>
  <AdHocIndex Name="BigDataByDate" Schema="OdbcTestSchema"
MembershipClass="BigData">
    <Key Name="tranDate" Descending="false" CaseInsensitive="false" SortOrder="0"
/>
    <Key Name="amount" Descending="false" CaseInsensitive="false" SortOrder="0"
/>
  </AdHocIndex>
</AdHocIndexDocument>
```

Running the Ad Hoc Index Controller Application

When you initiate the building, dropping, or deletion of a defined ad hoc index, the **Ad Hoc Index Controller** application is initiated if it is not running.

From the Ad Hoc Index Browser, the **Run Ad Hoc Index Controller App** command from the Ad Hoc Index menu enables you to run the **Ad Hoc Index Controller** application (for example, if the application terminated unexpectedly and you want to restart the application to continue building ad hoc entries in your Jade system).

To run the Ad Hoc Index Controller application if it is not already running

- In the Ad Hoc Index Browser, select the **Run Ad Hoc Index Controller App** command from the Ad Hoc Index menu.

If the application is currently running in the background, this command is displayed as **Ad Hoc Index Controller App Is Running** and it is disabled.

Notes If the application is run and there are no ad hoc indexes that can be maintained, the **Ad Hoc Index Controller** application terminates again.

If the Ad Hoc Index Browser is run on a Synchronized Database Service (SDS) secondary, the **Run Ad Hoc Index Controller App** command is disabled and the Ad Hoc Index menu displays **Ad hoc Index Controller App not available on secondary**.

Chapter 10 Extracting and Loading Schemas

This chapter covers the following topics.

- Jade Extract and Load Facilities
- Extracting Your Schema
  - Specifying Your Extract Options
  - Extracting Multiple Schemas
  - Specifying Your Schema Options
  - Specifying Your Parameter File Options
  - Specifying Selective Options
  - Specifying Your Application Options
  - Specifying Your Change Options
  - Extracting a Schema View
  - Extracting a Specific Class, Method, or Primitive Type
  - Extracting Schemas as a Non-GUI Client Application
- Loading Your Schema
  - Merging a Schema in the Load Process
  - Invoking the Load Process
  - Specifying Your Load Options
- Encrypting Schema Source Files
  - Extracting Encrypted Schema Source Files
  - Loading Encrypted Schema Source Files

Jade Extract and Load Facilities

Jade provides the following extract and load facilities.

- Extract options enable you to tailor the saving of your current schema or extract a specific schema view, class, interface, constant, or method. For details about automating the extraction of user-defined schemas, see "Extracting Schemas as a Non-GUI Client Application", later in this chapter. See also "Extracting Changes for a Patch Number" under "Maintaining Patch Numbers", in Chapter 3 of the *Development Environment Administration Guide*, for details about extracting changes for a specific patch number. For details about extracting a specific interface, see "Extracting an Interface", in Chapter 14.

- Load options enable you to load your schema and definition (for example, form, database, and ActiveX definitions) files for a specific schema view, class, interface, constant, or method from an extract file to Jade. In addition, you can load a file that can contain multiple schemas and their associated definition files, individual extracted class or method files, or configuration or definition files extracted from the Jade Report Writer (for example, a report view, formats, or definitions file). For details, see "Multiple Schema File Syntax", later in this chapter.

When you extract a class, its associated methods, properties, and so on, are also extracted.

Notes You can extract and load only your user-defined schemas. You cannot extract or load the **RootSchema** (the base schema that contains system objects).

If you load multiple schemas and you want to suppress the display of all warning messages resulting from compiling schema files (for example, class number conflict messages), set the **SuppressWarningDialogs** parameter in the [JadeCompiler] section of the Jade initialization file to **true**. Warning messages are then output only to the message log.

For details about loading locales, see "Loading Locales", in Chapter 11, "Internationalization".

Exception 5012 is raised if you attempt to load a Unicode schema into an ANSI Jade environment.

The **Extract** command from the Schema menu enables you to save (extract) the current schema to a file.

The **Load** command from the Schema menu enables you to load (install) a full or partial schema from an extracted file into your current schema or a new schema.

Use the Schema menu **Extract** and **Load** commands when you are:

- Backing up or restoring a full or partial schema
- Passing code to another Jade developer
- Fully reconstructing a schema in the same Jade release
- Documenting classes or code

The Schema menu **Extract** and **Load** commands enable you to extract and load:

- Full or partial schemas
- Multiple schemas, classes, interfaces, methods, or Jade Report Writer files extracted to one file
- Form, database, and ActiveX definitions in subschemas
- Changed entities only

The Views, Classes, Interfaces, Methods, Types, Relational, and Components menus provide facilities to save a selected schema, relational view, RPS mapping, .NET exposure, ActiveX type library, method, class, interface, constant, or class and its subclasses, and extract them to a file. Saving a schema or relational view, RPS mapping, .NET exposure, or ActiveX type library, class, interface, method, or constant provides you with a quick means of extracting that schema element; for example, for a backup, to pass the code to another user, or to reconstruct.

Using this facility is a simpler, faster way to extract an individual schema element than performing a selective extract of your schema.

Note The Jade extract process extracts definitions of any **OleControl** objects (as it does with other controls painted at design time when the form definitions are extracted) but it does not extract the user data associated with the controls. It is your responsibility to extract the user data and load it into a deployed database, if required.

If you attempt to load an extracted schema containing imported classes, interfaces, and features before the schema containing the export schema is loaded, a schema load error occurs. You must therefore take care when loading a schema containing an imported package that you first load the schema containing the exported package. However, if a package is imported into a schema that is extracted to a multiple extract file and the schema that exports this package is also extracted to that file, the exporting package schema extracted (and therefore loaded) before the importing package schema.

You can avoid problems of this nature if you maintain the export package as a separate schema file from the schema or schemas into which it is to be imported.

A user of the imported schema should first load the schema into which the package is to be imported, and then load the separate schema file for the package into that schema. (For details, see "Extracting and Loading Package Schemas", in Chapter 8 of the *Developer's Reference*.)

For details about extracting a method view that bookmarks a workflow, see "Extracting a Method View", in Chapter 13.

When RPS mappings are extracted during a schema extract, a ***ddb-file-name*.ddbx** or ***ddx-file-name*.ddxx** RPS mapping file is created. This file contains only the RPS mappings extracted during this schema extract. The RPS mappings are included in the .**ddb** or .**ddx** file, and the .**ddbx** or .**ddxx** file. When extracting a schema with an RPS mapping and the:

- DDX format style is selected, a separate .**ddxx** file is produced (the contents of this file are in XML format).
- DDB format style is selected, a separate .**ddbx** file is produced.

When no reorganization is required after the .**scm** file load, the .**ddb** or .**ddx** file can be loaded (as normal) before the reorganization is performed. If a reorganization is required after the .**scm** file load and before the .**ddb** or .**ddx** file load, the .**ddbx** or .**ddxx** file can be loaded before the reorganization, to keep the RPS mappings up-to-date with the schema changes. After the reorganization, the .**ddb** or .**ddx** file is loaded as normal, to complete the changes.

If a .**ddbx** or .**ddxx** file is created for a schema extract, the following comment line is put into the .**scm** file after the **Jade Command File** line (if present) and before the **jadeVersionNumber** line.

```
/* Jade RPS MAPPING FILE <ddbx|ddxx-file-name> */
```

The .**ddbx** or .**ddxx** file:

- Has the same format as a .**ddb** or .**ddx** file, and it is loaded as a .**ddb** or .**ddx** file, respectively.
- Is created for patch extracts, if the patch contains changes to an RPS mapping.
- Is created unconditionally, if the schema extract contains RPS mappings. There is no option to suppress its creation. If the file exists, it will be overwritten without warning.

Extracting Your Schema

Jade enables you to extract multiple schemas, a complete schema, or parts of a schema; for example, as a backup before you reorganize your database, install a new release of Jade, or if you want to pass code to another Jade developer.

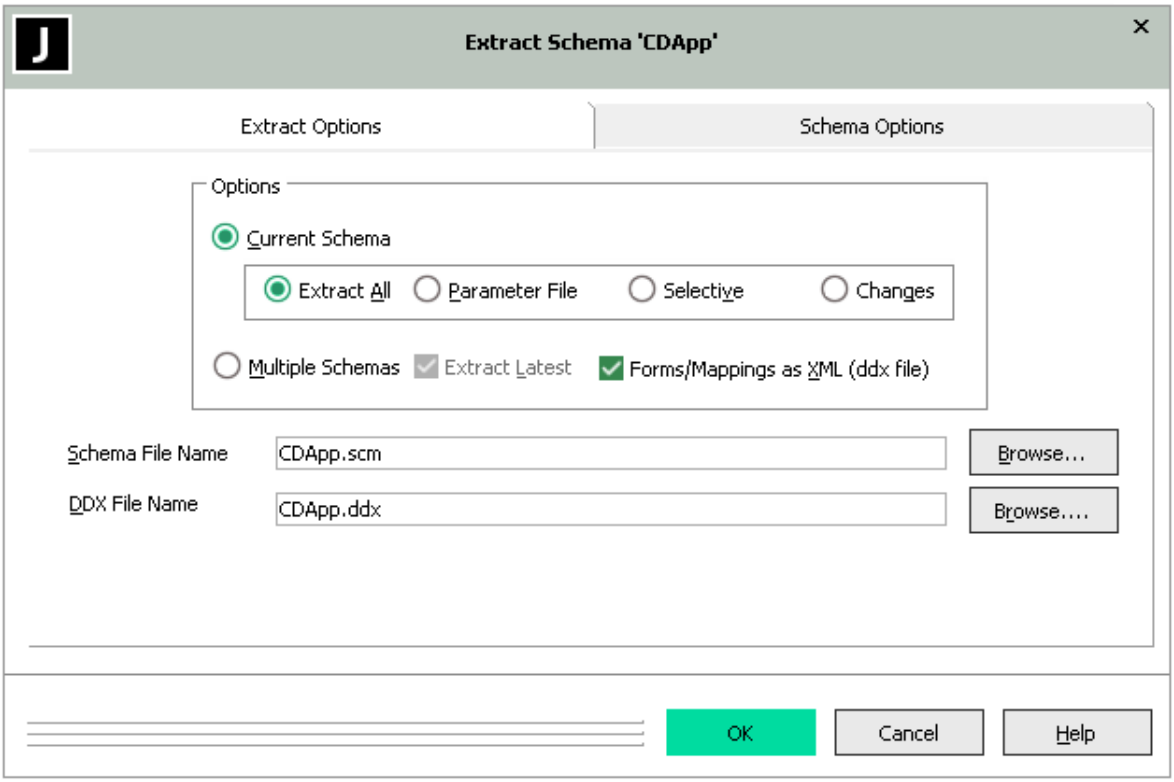
Note If you extract a class that has an inverse to a property of another class that is not part of the same extract operation, it is your responsibility to ensure that you extract the other class as well or that the other class already exists in the target schema with the appropriate property.

You can resize the Extract dialog (by clicking on the form border and then using the mouse to drag the form to the required position when the cursor changes shape to a double-ended arrow); for example, to enable you to display a greater number of classes, interfaces, and methods when performing a selective extract.

To extract a user-defined schema

1. In the Schema Browser, select the schema that you want to extract, if it is not the current schema.
2. Access the Extract dialog by performing one of the following actions.
   - Select the **Schema Extract** command from the File menu.
   - Select the **Extract** command from the Schema menu.
   - Click the **Extract** toolbar button.
   - Press Ctrl+Alt+E.

The Extract dialog, shown in the following image, is then displayed.



For details, see the following subsections.

Specifying Your Extract Options

The **Extract Options** sheet of the Extract dialog is displayed when you select the **Extract** command from the Schema menu, to enable you to specify the type of extract that you require.

To specify your extract options

1. In the Options group box, select the **Multiple Schemas** option button if you want to extract more than one schema. By default, only the current schema is extracted.

   The **Multi Extract File Name** text box is then displayed, instead of the **Schema File Name** and **DDX File Name** text boxes, as both schemas and forms are extracted with default file names when this option is selected.

   In addition, the **Extract Latest** check box is displayed at the right of the dialog when you select the **Multiple Schemas** option button. By default, this check box is checked if the current schema is not versioned or it is versioned and it is the latest version. If the schema is versioned and you want to extract the current (committed) version, uncheck this check box.

2. Select your extract option in the Current Schema group box. By default, the full schema is extracted; that is, the **Extract All** option is selected. Select the:

   - **Parameter File** option button if you want to specify selected classes, interfaces, and methods that are to be extracted. Use this option if you want to set up a parameter file that can be used for this and for subsequent subschema extracts. This enables you to extract the same classes and methods for any extract that is performed with this option selected until you change the parameter text file.

     Note If you select the **Parameter File** option, you must first have defined a parameter text file by using a text editor; for example, Notepad.

   - **Selective ExtractSelective** option button if you want to select specific classes, interfaces, and methods to be extracted.

     The **Selective** sheet is then displayed. Any specified classes, interfaces, and methods that you select are for this extract only. Although the selection is not saved for subsequent extracts of the subschema, you can save the selective extract information, if required, by using the **Save Parameters** check box in the **Selective** sheet.

   - **Changes** option button if you want to extract only the changes made to entities (classes, interfaces, method sources, property or constant details, and so on) in the schema patch versions.

3. If you want to extract form and data definitions in the legacy DDB format instead of the XML DDX format, uncheck the **Forms/Mappings as XML (ddx file)** check box. The value of this check box defaults to the value of the **Extract as DDX (xml format) instead of DDB** check box on the **Source Management** sheet of the Preferences dialog. By default, this check box is checked; that is, schemas are extracted as .**ddx** files.

   The XML format does not include Jade oids. All entities are identified by name and by their position in the XML object hierarchy. You can compare the DDX file to another version of the file, to identify what has changed between the two versions.

   The first line of a .**ddx** file has the **<?xml...** header. The format of the second line is:

   ```
   <schema name="schema-name" JadeVersionNumber="Jade-version"
   JadePatchNumber="patch-number" CompleteDefinition="true|false">
   ```

   The following is an example of the first and second lines of the XML file.

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <schema name="CalculatorSchema" JadeVersionNumber="18.0.01"
   ```

```
JadePatchNumber="0" CompleteDefinition="true">
```

The schema name, which specifies the schema the information is for, must be included.

The **JadeVersionNumber** tag identifies the version of Jade that was used to produce the file.

The **JadePatchNumber** tag specifies the patch number to use for the load; otherwise the current patch number is used.

The **CompleteDefinition** tag, which must be present, specifies whether the file is a complete definition for the whole schema or it is a partial schema. If the value of the **CompleteDefinition** tag is **true**, any existing entities not included in the file are deleted.

4. In the **Schema File Name** text box, specify the name and location of the schema file that you want to extract; for example, **c:\jade\bin\MySchema.scm**. If you do not specify a location, the file is extracted to your Jade working directory.

   If you want to extract the schema to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   For ease of identification, you should prefix your schema file name with at least the first few distinguishing characters of the schema name and with the appropriate suffix (for example, **scm** for an extracted schema, **mth** for an extracted method, or **cls** for an extracted class or interface). An error is raised if you are extracting the schema and you do not specify the name of a file, your specified file name or path is invalid, or an existing file cannot be accessed.

5. In the **DDX File Name** text box, specify the name and location of the forms file that you want to extract; for example, **c:\jade\bin\MySchema.ddx**. If you do not specify a location, the file is extracted to your Jade working directory.

   Note If the **Forms/Mappings as XML (ddx file)** check box is unchecked, this text box is named **DDB File Name** and the file suffix is .**ddb**.

   If you want to extract the form and control definitions to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   For ease of identification, you should prefix your forms file name with at least the first few distinguishing characters of the schema name, with a suffix of .**ddx**. An error is raised if you are extracting form and control definitions and you do not specify the name of a file, your specified file name or path is invalid, or an existing file cannot be accessed.

6. If you selected the **Multiple Schemas** option button from the Options group box, specify the name and location of the multiple schemas file you want to extract (for example, **c:\jade\bin\TestScms.mul**) in the **Multi Extract File** text box. If you do not specify a location, the file is extracted to your Jade working directory. Each schema that you selected is extracted to a separate pair of .**scm** and .**ddx** files. The multiple schemas extract file itself contains merely a list of these file names.

   If you want to extract the schemas to an existing multiple schemas file or you are unsure of existing file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   The file prefix defaults to the name of the current schema, with a file suffix of .**mul**. An error is raised if you are extracting multiple schemas and you do not specify the name of a file or if an existing file cannot be accessed. (See "Multiple Schema File Syntax", later in this chapter, for more details.)

7. If you selected the **Multiple Schemas** option button from the Options group box, click the **Schemas** sheet. For details, see "Extracting Multiple Schemas", in the following section.

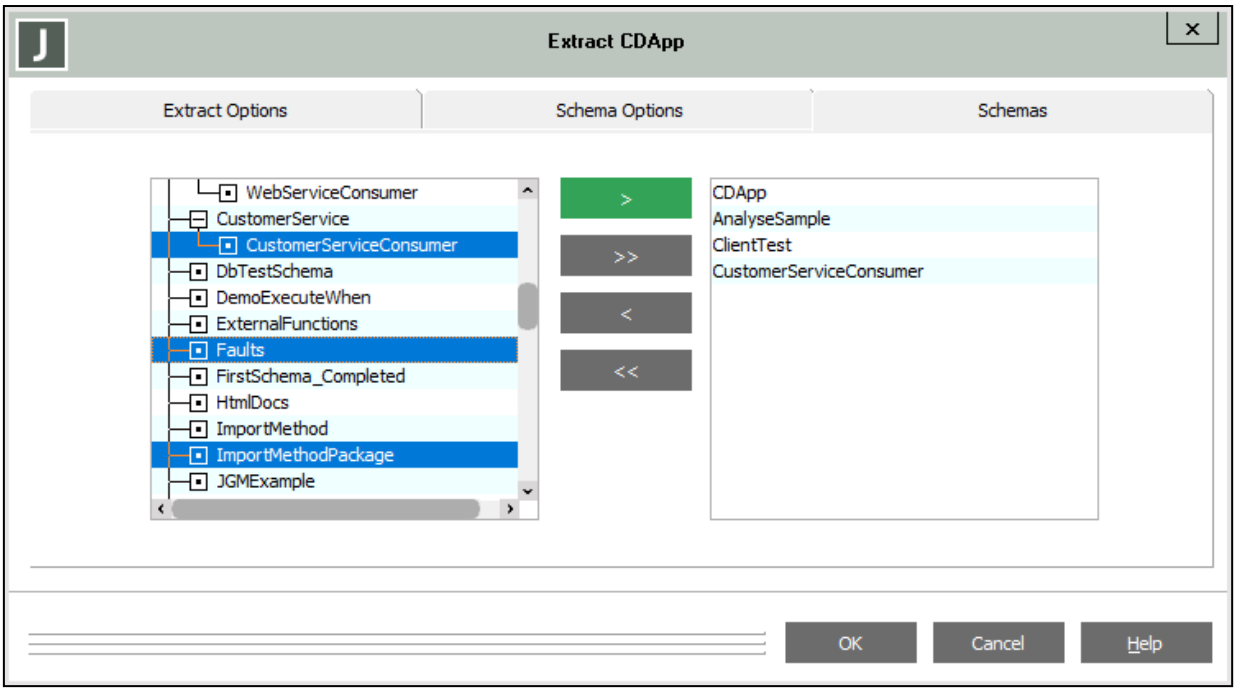
If you selected the:

- **Use Parameter FileParameter File** option button from the Current Schema group box, click the **Parameters** sheet. For details, see "Specifying Your Parameter File Options", later in this chapter.
- **Selective ExtractSelective** option button from the Current Schema group box, click the **Selective** sheet. For details, see "Specifying Selective Options", later in this chapter.
- **Changes** option button from the Current Schema group box, click the **Changes** sheet. For details, see "Specifying Your Change Options", later in this chapter.

Alternatively, click the **OK** button to confirm your selections, the **Schema Options** sheet to specify your schema options or the **Cancel** button to abandon your selections.

Extracting Multiple Schemas

The **Schemas** sheet, shown in the following image, is displayed when you select the **Multiple Schemas** option button in Options group box of the **Extract Options** sheet of the extract multiple schemas dialog.



The **Schemas** sheet enables you to specify the schemas that you want to extract. The current schema is displayed in the selected schemas list box at the right of the sheet.

To specify your schemas

1. The hierarchy of all schemas is displayed, to enable you to make your selections.

   From the candidate schemas list box at the left of the dialog, select one or more schemas that you want to extract to a file. System schemas that cannot be extracted from your schema (for example, the **RootSchema**) are disabled. Use one of the following methods to select the required schemas.

   - Double-click on the appropriate schema.

     The selected schema is then displayed in the selected schemas list box at the right of the dialog. Repeat this action until all of the schemas that you want to extract are displayed in the selected schemas list box.

   - Click on a schema that you want to extract and then click the **>** button to move the selected schema to the selected schemas list box. Repeat this action until all of the required schemas are displayed in the selected schemas list box.

   - To move all user-defined schemas to the selected schemas list box, click the **>>** button. All schemas are then displayed in the selected schemas list box at the right of the dialog.

   - As the candidate schemas list box is a multiple-selection list box, you can use the Shift or Ctrl key to select a group of schemas and then click the **>** button to move the selected schemas to the selected schemas list box.

   When schemas have been selected for extraction and are displayed in the selected schemas list box, they are dimmed in the candidate schemas list box.

2. You can move schemas selected for extract from the selected schemas list box back to the candidate schemas list box, to exclude them from the extract. The actions required to do this are identical to those described in step 1 of this instruction, except that the **<** button is used to move a single schema and the **<<** button to move all schemas selected for extraction back to the candidate schemas list box.

3. When all schemas that you want to extract from the **RootSchema** are displayed in the selected schemas list box on the right of the dialog, click the **OK** button. Alternatively, perform one of the following actions.

   - If you want to return to the **Extract Options** sheet to change the name of location of your schema or forms file, click the tab of that sheet.

   - If you want to return to the **Schema Options** sheet to change the options that apply to all of the schemas that you extract (for example, encrypting the schema source), click the tab of that sheet.

   - Click the **Cancel** button to abandon the extract.

When you click the **OK** button, the extraction of your selected schemas is then started.

Multiple Schema File Syntax

If you specify the extraction of multiple schemas, the names of your selected schemas (and their associated forms) are by default extracted to an .**mul** file, with a prefix name of the current schema.

Notes You can also specify files extracted from the Jade Report Writer or individual class or method files extracted from the Jade Platform development environment. When you edit an .**mul** file (for example, by using a text editor such as Notepad), specify each file on a separate line.

When you specify different file types extracted from the Jade Report Writer, define them in the order in which they should be loaded (that is, in the logical order in which they were defined and extracted), with configuration files and then any report definitions files extracted from the designer application specified last.

If a package is imported into a schema that is extracted to a multiple extract file and the schema that exports this package is also extracted, the exporting package schema extracted (and therefore loaded) before the importing package schema. For details about selecting multiple files to load into a Jade database, specifying the order in which the files are loaded, and then creating a multiple load file, see "Specifying the Load Order when Loading Multiple Files", later in this chapter.

The syntax of the multiple schema file is as follows.

```
#MULTIPLE_SCHEMA_EXTRACT
first-schema-name.scm first-schema-name.ddb|ddx
second-schema-name.scm second-schema-name.ddb|ddx
third-schema-name.scm third-schema-name.ddb|ddx
...
[class-file-name.cls]
[type-name_method-file-name.mth]
[reporting-view-file-name.rwv]
[report-formats-file-name.rwo]
[report-folders-file-name.rwf]
[report-users-file-name.rwu]
[report-definitions-file-name.rwr]
```

The following is an example of a multiple schema extract file.

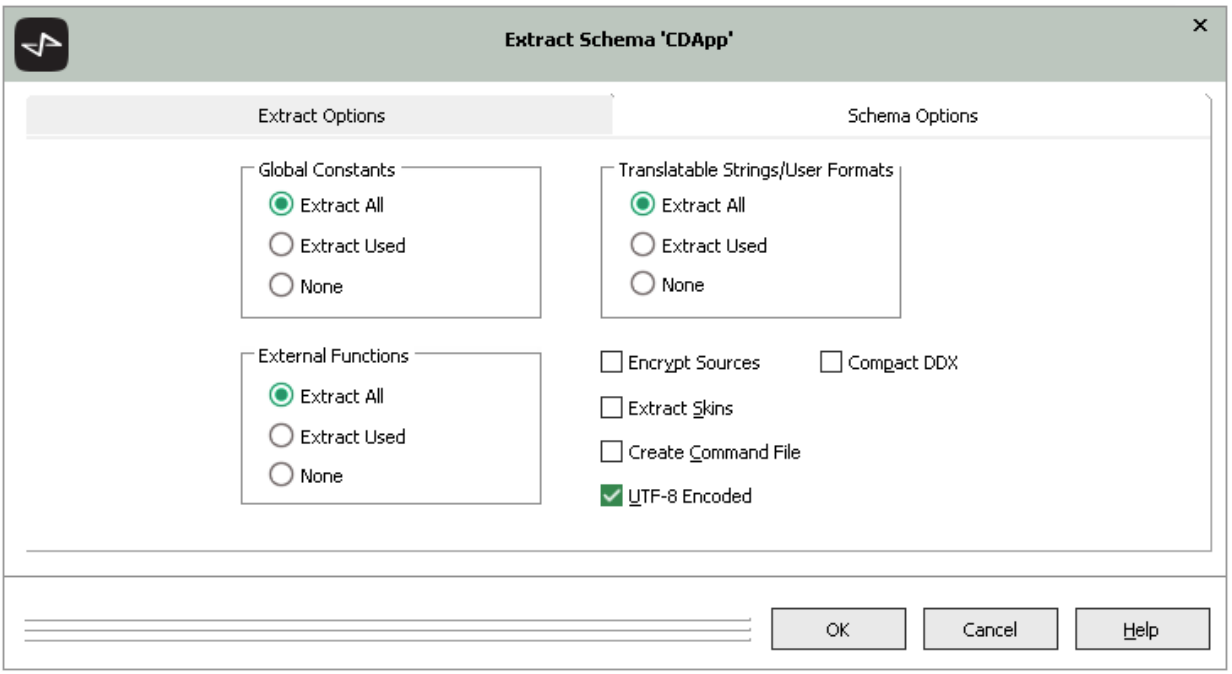
```
#MULTIPLE_SCHEMA_EXTRACT
MyTestSchema.scm MyTestSchema.ddx
LockTest.scm LockTest.ddx
PrintTest.scm PrintTest.ddx
SortTest.scm SortTest.ddx
RemoteNodesDict.cls
LocalConnections.cls
Graphs_smpleGraph.mth
LocalConnections_allRemoteNodes.mth
TestReportViews.rwv
DevelopmentViews.rwv
TestReportFormats.rwo
AllReportFolders.rwf
ContractorUsers.rwu
DevelopmentReports.rwr
PersonnelReports.rwr
SalesReports.rwr
```

For details and syntax about the multiple schema file to which methods added to a method view are extracted, see "Extracting a Method View", in Chapter 13.

Specifying Your Schema Options

The **Schema Options** sheet of the Extract dialog is displayed when you have specified your extract options and you select the **Schema Options** sheet, to enable you to specify the parts of the schema that are to be extracted.

An example of the **Schema Options** sheet is shown in the following image.



When you extract multiple schemas to a file, the schema options apply to all of the schemas that you extract to the file.

To specify your schema options

1. In the Global Constants group box, select the global constants that are to be extracted. For a full extract (the **Extract All** option was selected on the **Extract Options** sheet), all global constants in the schema are extracted by default. For a selective extract, only the global constants that are used in your methods are extracted by default.

   Select the **Extract All** option button if you want all global constants in the schema to be extracted. Select the **None** option button if you do not want any global constants to be extracted.

2. In the Translatable Strings/User Formats group box, select the translatable strings and user formats that are to be extracted.

   For a full extract (the **Extract All** option was selected on the **Extract Options** sheet), all translatable strings and user formats in the schema are extracted by default. For a selective extract, only the translatable strings and user formats that are used in the schema are extracted by default.

   Select the **Extract All** option button if you want all translatable strings and user formats extracted. Select the **None** option button if you do not want any to be extracted.

3. In the External Functions group box, select the external functions that are to be extracted.

   For a full extract (the **Extract All** option was selected in the Extract Options sheet), all external functions in the schema are extracted by default. For a selective extract, only the external functions that are used in your methods are extracted by default.

   Select the **Extract All** option button if you want all external functions extracted. Select the **None** option button if you do not want any to be extracted.

4. Check the **Encrypt Sources** check box if you want to encrypt the Jade method source code in your extract file. By default, extracted source code is not encrypted; that is, this check box is unchecked.

   Source encryption provides security when you release schema extract files, as the source code is not easily visible (for example, when you deploy an application to a third-party). For details, see "Encrypting Schema Source Files", later in this chapter.

   Caution Ensure that you extract the encrypted schema to a location different from that of your source schema. If you subsequently load the encrypted schema (for example, for testing purposes), your method source code is lost if you load it into the same database that contains your original source files, as they are not saved during the decryption process.

   As schema and form definition files are treated as binary files, if the File Transfer Protocol (FTP) is used to transfer schema and forms definition files between machines, you must ensure that the transfer is done in binary mode (rather than ASCII) to prevent the removal of carriage return characters, particularly when schemas are encrypted.

5. Check the **Extract Skins** check box if you want to extract Jade skins defined for runtime applications. By default, skins are not extracted; that is, this check box is unchecked.

   Note Skins are stored in a global collection that is available to all runtime applications in all systems in your Jade database. When you extract skins, all skins in the Jade database defined for runtime applications are extracted to your form and data definition (.**ddb** or .**ddx**) file.

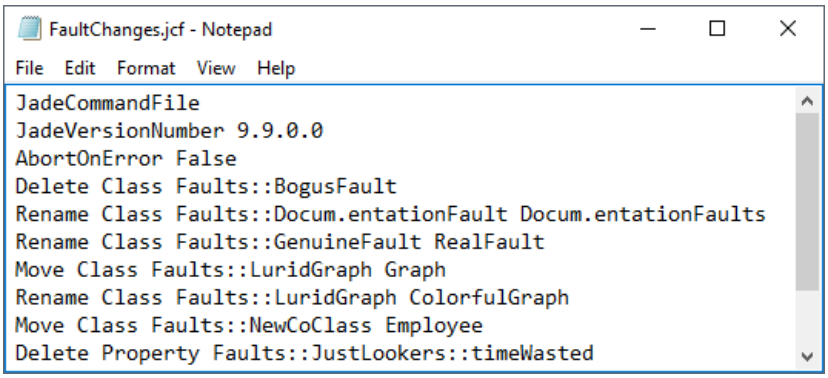
   For more details, see "Using Jade Skins in Your Runtime Applications", in Chapter 2 of the *Runtime Application Guide*.

6. If patch control is disabled and you do not want to create a command file, uncheck the **Create Command File** check box. If patch versioning:

   - Was never used in the system, this check box is disabled and not checked.
   - Has been used but is now disabled, this check box is enabled and checked.
   - Is enabled, this check box is checked but you can uncheck it to stop the creation of command files, if required.

   Note The **Unset Extract option 'Create JCF Command File'** check box on the **Source Management** sheet of the Preferences dialog enables you to specify whether this check box is checked by default.

   The command file, shown in the example in the following image, is created with a suffix of **.jcf** (Jade command file) in the location of the schema and forms definition files.



This file contains a header section, followed by a separate line for each command to be processed.

Notes As the extracted commands are a guide only, you should check the commands before you make the Jade command file available to load into a deployed application by using the batch Schema Load utility. For details, see "Loading Schemas in Batch Mode using jadloadb", in the *Schema Load User's Guide*.

The command file automatically extracts only deleted classes and properties, renamed classes and properties, and moved classes.

If you set the **AbortOnError** command in the file to **True**, processing of the command file is aborted when the first error is encountered in the file during the batch schema load process.

When patch control is disabled, the command file contains header information only and you must enter the commands manually.

7. Check the **UTF-8 Encoded** check box if you want the extract file encoded as UTF-8 with a Byte Order Mark (BOM). If you do not check this check box, the extract file is output in native ANSI or Unicode format.

   Note The **Source Management** sheet of the Preferences dialog enables you to specify whether this check box is checked by default.

8. Check the **Compact DDX** check box if you want to extract compact DDX extract files. If you check this check box, the extracted DDX files exclude properties with null values for applications, forms, standard controls, and RPS mappings.

   Compact DDX files reduce the file size of the extracted file and improve loading performance. By default, this check box is not checked.

   A compact DDX file includes **Compact="true"** in the second line of the DDX file; for example:

   ```
   <schema name="Schema1" JadeVersionNumber="2022.0.02" Compact="true"
   JadePatchNumber="0" CompleteDefinition="true">
   ```

   Notes This check box is enabled only if you are extracting DDX-format files.

   Compact DDX files extracted in Jade 2022.0.02 and higher will not load into Jade 2022.0.01 or any earlier Jade releases.

9. If you selected the **Parameter File** option button from the Extract Options group box, click the **Parameters** sheet. If you selected the **Selective** option button from the Extract Options group box, click the **Selective** sheet.

   Alternatively, click the **OK** button to confirm your selections, the **Applications** sheet to select your applications for extraction, or the **Cancel** button to abandon your selections.

Applies to Version: 2022.0.01 and higher

Specifying Your Parameter File Options

The **Parameter** sheet is displayed when you have selected the **Parameter File** option button in the **Extract Options** sheet and you then click the **Parameter** sheet.

The **Parameter** sheet enables you to select the parameter file that specifies the classes, interfaces, and methods that are to be extracted.

Notes A parameter file must already exist.

All classes, interfaces, or methods specified in an existing parameter file are extracted (which provides less granularity than a selective extract of entities saved to a parameter file, when you can tailor the entities you want to extract as a partial definition).

As the complete interface must be extracted when an interface method is required to be extracted, the extract process validates that methods specified in the parameter file are not defined as part of an interface.

A selective extract file includes the web service exposure only if the web service provider subclass of the **JadeWebServiceProvider** class is included in the parameter file. Although you do not have to perform a full extract of your schema, you *do* have to include the web service class in the partial extract file.

Parameter options are enabled only when you extract the current schema; that is, you cannot specify these options for the extraction of multiple schemas.

To specify your parameter file options

1. In the **Parameter File Name** text box, specify the name and location of your parameter file that contains the classes, interfaces, and methods that are to be extracted or excluded from the schema extraction.

   You must specify a value in this text box. An error is raised if you do not specify the name of an existing text file. (The file must be a text file.)

   If you are unsure of the name or location of your parameter file, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file location and name.

2. Check the **Exclude Elements** check box if you want to extract all of your schema *except* for the classes, interfaces, and methods specified in your parameter file.

3. Click the **OK** button, to confirm your selections. Alternatively, click the **Cancel** button to abandon the extract.

Parameter File Syntax

If you specify a selective extract file with the **Save Parameters** check box checked, the names of your selected elements are by default extracted to an .**unl** file with a prefix name of the current schema.

However, if you want to write your own parameter file using a text editor, the syntax of the parameter file is as follows.

```
Class class-name
...
JadeInterface interface-name
...
Method class-name method-name
...
RPSMapping rps-mapping-name class-name table-list
...
```

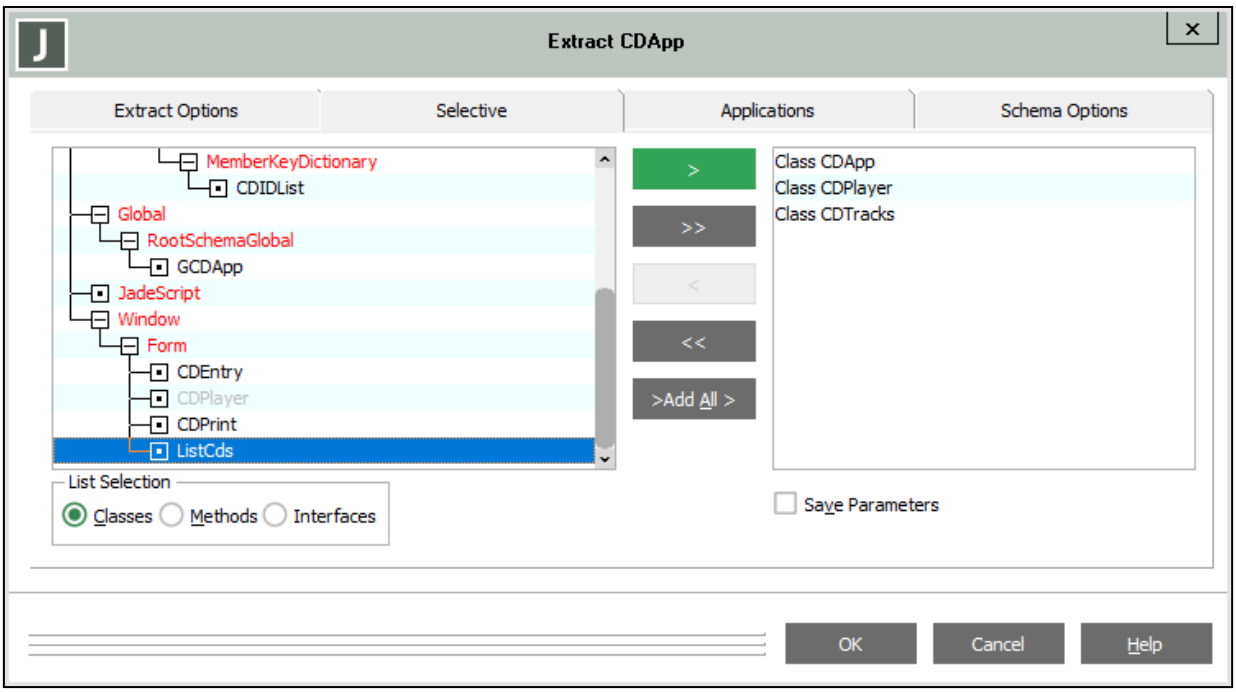
The following is an example of a parameter file.

```
Class Faults
Class CustomerDict
Class EmployeeSet
Class Customer
Class Employee
JadeInterface InterfaceA
JadeInterface TestInterface1
Method DocumentationFault printLetterText
```

```
Method DrawGraph load
Method FaultClose bOK_click
RpsMapping ErewhonRPS Country Location
RpsMapping ErewhonRPS TenderSaleItem SaleItem
```

Specifying Selective Options

The **Selective** sheet, shown in the following image, is displayed when you have selected the **Selective** option button in the **Extract Options** sheet and you then click the **Selective** sheet.



This sheet is also displayed when you click the **Show Items on Selective Sheet** button on the **Changes** sheet if you are extracting patch version changes only.

The **Selective** sheet enables you to specify selected classes, interfaces, and methods that are to be extracted from your schema. Selective options are enabled only when you extract the current schema; that is, you cannot specify these options for the extraction of multiple schemas.

A selective extract file includes the web service exposure only if the web service provider subclass of the **JadeWebServiceProvider** class is included in the parameter file. Although you do not have to perform a full extract of your schema, you *do* have to include the web service class in the partial extract file.

As a selective extract does *not* extract external databases, relational views, RPS mappings, or .NET exposures, you must extract these separately by using the appropriate **Extract** command if the current versions are not included in a full extract that is intended for a deployed application.

For details about the order in which files should be loaded into a deployed database, see "Before You Get Started", in the *Schema Load User's Guide*.

Tip You can resize the Extract dialog (by clicking on the form border and then using the mouse to drag the form to the required position when the cursor changes shape to a double-ended arrow), to enable you to display a greater number of classes, interfaces, and methods.

The hierarchy of all classes in your subschema is displayed by default, to enable you to make your selections.

To specify your selections

1. From the candidate objects list box at the left of the dialog, select one or more classes that you want to extract to a file.

   If you want to extract only selected:

   - Methods for a class selected in the candidate objects list box, click the **Methods** option button in the List Selection group box. The methods for the selected class are then displayed in the list box.
   - Interfaces in the schema, click the **Interfaces** option button in the List Selection group box. The interfaces in the schema are then displayed in the list box.

   Classes, interfaces, or methods that cannot be extracted from your schema (for example, the **Object** class that is part of the **RootSchema** and classes, interfaces, and methods that have already been selected for extraction) are disabled. The **>** button (move selected objects), the **>>** button (move all objects), and the **>Add All >** button (move all classes, interfaces, and methods) are disabled if you select a class or interface that is dimmed.

   Use one of the following actions to select the required classes, interfaces, or methods.

   - Double-click on the appropriate class, interface, or method.

     The selected class, interface, or method is then displayed in the selected objects list box at the right of the dialog. Repeat this action until all required classes, interfaces, and methods are displayed in the selected objects list box.
   - Click on a class, interface, or method that you want to extract and then click the **>** button to move the selected class, interface, or method to the selected objects list box.

     Repeat this action until all required classes, interfaces, and methods are displayed in the selected objects list box.
   - To move all displayed classes, interfaces, or methods to the selected objects list box, click the **>>** button. All classes, interfaces, or methods are then displayed in the selected objects list box at the right of the dialog.
   - To move all classes, interfaces, and methods in the current schema to the selected objects list box, click the **>Add All >** button.

     All classes, interfaces, and methods in the schema are then displayed in the selected objects list box at the right of the dialog. You can then selectively remove elements that you do not want to extract from the selected objects list box.
   - As the candidate objects list box is a multiple-selection list box, you can use the Shift or Ctrl key to select a group of classes, interfaces, or methods and then click the **>** button to move the selected objects to the selected objects list box.

   When classes, interfaces, or methods have been selected for extraction and are displayed in the selected objects list box, they are disabled in the candidate objects list box.

   Tip You can locate a specific class, interface, or method in the candidate objects list box by pressing the F4 key or by right-clicking in the list box and selecting the **Find** command from the popup menu that is then displayed.

   The Find Type dialog is then displayed, to enable you to specify or select the class that you want to locate.

2. You can move classes, interfaces, or methods selected for extract from the selected objects list box back to the candidate objects list box, to exclude them from the extract.

The actions required to do this are identical to those described in step 1, except that the **<** button is used to move single items and the **<<** button to move all items selected for extract back to the candidate objects list box.

3. When all classes, interfaces, and methods that you want to extract from your schema are displayed in the selected objects list box on the right of the dialog, check the **Save Parameters** check box if you want your selected classes, interfaces, and class methods to be saved to a parameter file.

   A parameter file can be reused for subsequent schema extracts, to enable you to extract the same classes, interfaces, and class methods for any extract that is performed with this option selected, until you change the parameter text file. By default, the selections that you make are not saved in a parameter file.

   When you click the **OK** button, the common File dialog is then displayed, to enable you to select the location and name of your parameter file. By default, the file location is your Jade working directory and the file name is the schema name with an .**unl** extension; for example:

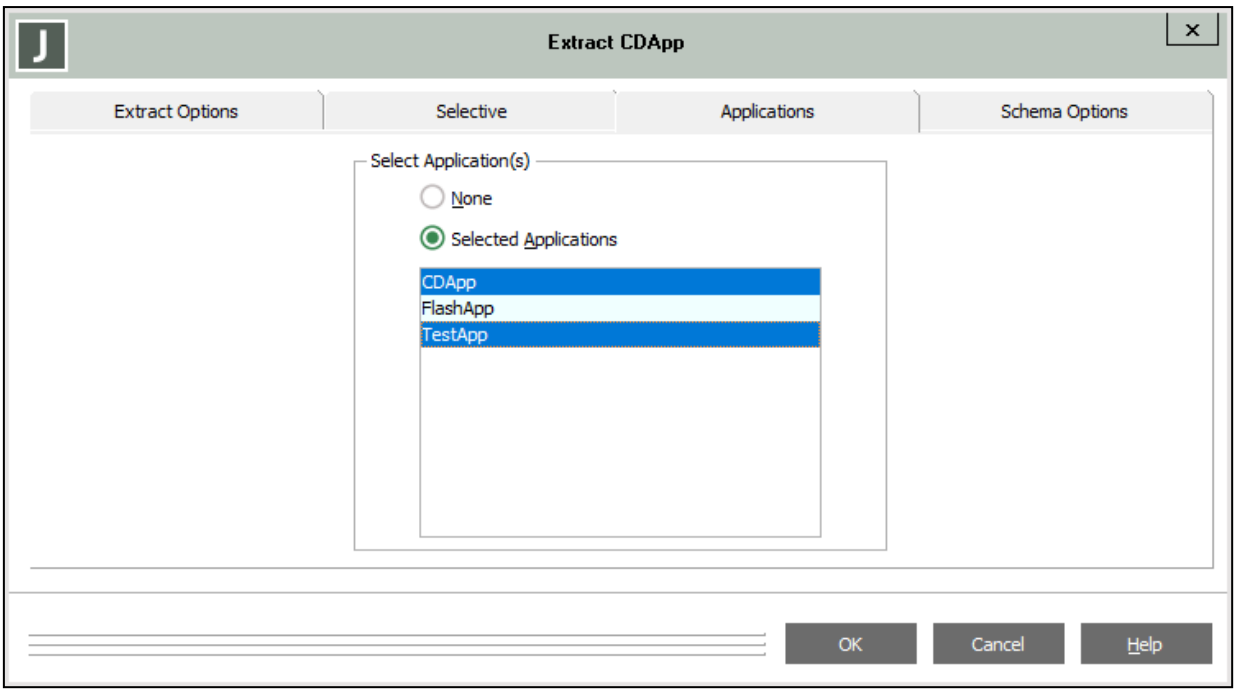
   ```
   FaultsSchema.unl
   ```

4. If you want to specify the applications in the schema (if any) that are to be included in the extract, click the **Applications** sheet from the Extract Options group box. The **Applications** sheet is then displayed. By default, only the current application is selected.

   If you want all applications in the current schema to be extracted, select all of the applications in the list and then click the **OK** button to confirm your selections. Alternatively, click the **Cancel** button to abandon the extract.

Specifying Your Application Options

The **Applications** sheet of the Extract dialog, shown in the following image, is displayed when you have specified a selective or parameter extract and you click the **Applications** sheet.



The **Application** sheet enables you to select the applications that are to be extracted with a selective or parameter extract. (All applications are extracted with a full extract.)

You can select that no application instances are to be extracted or you can select one or more applications for inclusion in the extract file.

If an application is currently set, that application is extracted in a selective extract by default.

Application options are enabled only when you extract the current schema; that is, you cannot specify these options for the extraction of multiple schemas.

To specify your application options

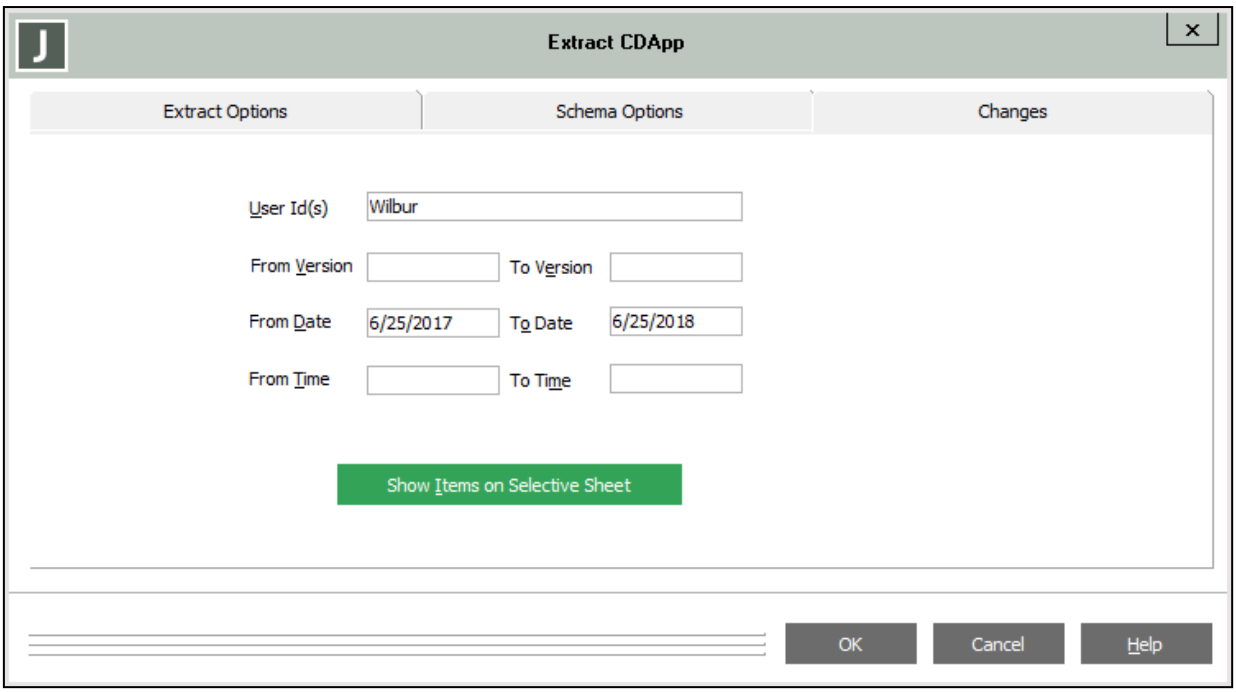
1. In the Select Application(s) group box, select the **None** option button if you do not want any applications included in the extract.

   If you want to extract one or more applications, click to select or deselect the applications that you want to extract. (If an application is currently set, that application is selected for extraction.) The **Selected Applications** option button is selected by default.

2. Click the **OK** button, to confirm your selections. Alternatively, click the **Cancel** button to abandon the extract.

Specifying Your Change Options

The **Changes** sheet of the Extract dialog, shown in the following image, is displayed when you have selected the extraction of patch version changes in the current schema and you click the **Changes** sheet.



Use the **Changes** sheet to select the schema entities that have changed within specified criteria.

You can specify the extraction of patch version changes:

- From one system version or patch version to another specified system version or patch version. For example, **16.0.2** specifies the Jade system version and **16.0.2.4** specifies patch version 4 in that Jade release; that is, in 2016.0.02 (Service Pack 1).
- Within a specified date and time.

You can also select the entities that have changed in the patch versions that match your specified criteria. All applications added to or changed in the specified patch version are included in the schema extract.

Change options are enabled only when you extract the current schema; that is, you cannot specify these options for the extraction of multiple schemas.

Note Only the latest change to an entity is extracted. If a specific entity was changed in any way by any developer in an earlier patch release and it was again changed in a later patch release, only this latest change is extracted and any earlier change is lost. You should therefore consider extracting patch version changes when you set a new patch version, if appropriate. For details, see "Setting up a Patch Number" under "Patch Versioning", in Chapter 3 of the *Development Environment Administration Guide*.

To specify your change options

1. In the **User Id(s)** text box, specify the full user identifier of the developer whose changed entities you want to extract. If you do not specify a value in this text box, the patch version changes made by all developers are extracted. By default, your user id is displayed in this text box.

   You can specify one or more user identifiers in this text box, by specifying each of the required identifiers separated by a space. You must specify each user id in full (for example, **wilburBloggs2**), as abbreviated user ids (for example, **wilbur**) are not valid.

2. In the **From Version** text box, specify the number of the patch version from which the changed entities are to be extracted; for example, **16.0.2.4**. The last number (the **.4**, in this example) indicates the patch version; the preceding numbers indicate the system version and release of Jade itself (**16.0.2**, in this example).

   If you do not specify a value in this text box, the extraction of changed entities starts from your first Jade patch version.

3. In the **To Version** text box, specify the number of the latest patch version from which the changed entities are to be extracted; for example, **16.0.2.7**. The last number (the **.7**, in this example) indicates the patch version; the preceding numbers indicate the version and release of Jade itself (**16.0.2**, in this example).

   If you do not enter a value in this text box, the value defaults to that specified in the **From Version** text box, if present. If you specify a value in neither the **From Version** nor the **To Version** text box, the extraction is performed on all changed methods that match any specified user id or date criteria.

4. In the **From Date** text box, specify the starting date from which patch version changes are to be extracted; for example, **21MAY17**. (The date must be a valid date.)

   By default, the current date is displayed in this text box.

   If you do not specify a value in this text box, entities are extracted starting from the earliest timestamp; that is, the first date on which a patch version entity was changed.

5. In the **To Date** text box, specify the date up to which changed entities are to be extracted; for example, **28November17**. (The date must be a valid date.) By default, the current date is displayed in this text box.

   If you do not specify a value in this text box, the extraction ends with the last entity changed on the current date.

6. If you specified a value in the **From Date** text box and you want to specify the time on that date from which changed entities are extracted, specify the starting time in the **From Time** text box; for example, **09:30:00**. (The time must be a valid time.)

   If you do not specify a value, the time defaults to **00:00:00**. Your specified value is ignored if you have not specified a start date.

7. If you specified a value in the **To Date** text box and you want to specify the time on that date up to which changed entities are extracted, specify the end time in the **To Time** text box; for example, **18:59:59**. (The time must be a valid time.)

   If you do not specify a value, changed entities are extracted up to the current time. Your specified value is ignored if you have not specified an end date.

8. Click the **Show Items on Selective Sheet** button if you want to restrict the extraction of changed entities to specific classes or methods.

   The **Selective** sheet of the Extract dialog is then displayed, to enable you to select only those changed entities that you want to extract. For details, see "Specifying Selective Options", earlier in this chapter.

9. Click the **OK** button to confirm your selections. Alternatively, you can click the **Cancel** button to abandon your selections.

For details about setting a patch version and displaying a summary of patch version changes, see "Patch Versioning", in Chapter 3 of the *Development Environment Administration Guide*.

Extracting a Schema View

Use the **Extract** command from the Views menu in the Schema Views Browser to save a selected schema view and extract it to a file. The **Extract** command is a quick way of extracting a schema view for:

- A backup
- Passing the code to another user
- Reconstructing the extracted schema

Note When you extract a schema view, its associated classes, primitive types, methods, properties, and so on are also extracted.

To extract the schema view selected in the Schema Views Browser

- Select the **Extract** command from the Views menu.

The Extract dialog for the selected schema view is then displayed.

Specifying Your Schema View Extract Options

As you can extract a complete schema view only (that is, you cannot extract selected classes in that view), the **Extract All** option button in the Options group box is selected. The **Parameter File** and **Selective** option buttons are disabled.

To specify your extract options

1. In the **Schema File Name** text box, specify the name and location of the file that you want to extract; for example, **c:\jade\bin\view\SaleView.scm**.

   If you do not specify a location, the file is extracted to your Jade working directory (that is, the directory in which the **jade.exe** program is located).

   If you want to extract the schema view to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

The schema file name defaults to the name of the schema view, with a suffix of **.scm**. An error is raised if an existing file cannot be accessed.

2. In the **DDX File Name** text box, specify the name and location of the forms file that you want to extract; for example, **c:\jade\bin\MySchema.ddx**. If you do not specify a location, the file is extracted to your Jade working directory.

   If you want to extract the form and data definitions to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   The forms file name defaults to the name of the schema view, with a suffix of **.ddb** or **.ddx**, depending on the value of the **Extract as DDX (xml format) instead of DDB** check box on the **Source Management** sheet of the Preferences dialog. An error is raised if an existing file cannot be accessed.

   Note If you are extracting the forms file in the legacy **.ddb** format, this text box is named **DDB File Name** and the file suffix is **.ddb**.

3. Click the **OK** button to confirm your selections, the **Schema Options** sheet to specify your schema view options, or the **Cancel** button to abandon your selections.

Specifying Your Schema View Options

The **Schema Options** sheet of the Extract dialog is displayed when you have specified your extract options and you select the **Schema Options** sheet, to enable you to specify the parts of the schema view that are to be extracted. For details, see "Specifying Your Schema Options", earlier in this chapter.

Extracting a Specific Class, Method, or Primitive Type

Use the **Extract** command from the Classes, Methods, or Types menu in the Class Browser or Primitive Types Browser to save a selected class, method, or all methods in a selected primitive type for extraction to a file.

Use the **Extract All** command from the Classes menu in the Class Browser to extract the selected class and all subclasses, if applicable.

For details about extracting a specific interface, see "Extracting an Interface", in Chapter 14.

Note If you want to encrypt your extracted class, method, or primitive type methods, you must use the **Selective** sheet of the Extract dialog to select the element that you want to extract and then check the **Encrypt Sources** check box in the Extract dialog **Schema Options** sheet. (The common Save As dialog does not enable you to encrypt the saved file.)

You can also extract an external database schema, an ActiveX type or .NET library, a relational view, an exposure, a method view, an RPS mapping, or a .NET exposure. For details, see "Extracting an External Database Schema" in Chapter 3, "Extracting and Loading ActiveX Schema Definition Data" or "Extracting and Loading .NET Schema Definition Data" in Chapter 16, "Extracting a Relational View" in Chapter 9, "Extracting an External Function or Library" in Chapter 8, "Extracting a Method View" in Chapter 13, "Extracting an RPS Mapping" in Chapter 15, or "Extracting a C# Exposure" in Chapter 17 of this guide, respectively.

For details about loading an extracted class or method in a multiple schemas file (**.mul**), see "Multiple Schema File Syntax", earlier in this chapter.

The **Extract** or **Extract All** command is a quick means of extracting a class, method, or primitive type user-defined methods for:

- A backup
- Passing the code to another user
- Reconstructing the extracted class or method

The class, primitive type, or method **Extract** or **Extract All** command is a simpler, faster way to extract an individual method, class, all user-defined methods in a primitive type, or class and subclasses than performing a selective extract of your schema.

Notes When you extract a class, its associated methods, properties, and so on are also extracted. When you extract a selected primitive type, its associated methods are extracted.

You can extract only user-defined classes or methods.

Extracting a Selected Class

To extract the class that is currently selected in the Class List

1. Select the **Extract** command from the Classes menu to extract only the selected class or the **Extract All** command to extract the selected class and its subclasses, if appropriate. The Extract dialog **Extract Options** sheet is then displayed.
2. Click the **Browse** button at the right of the **Schema File Name** text box. The common Save As dialog is then displayed, to enable you to specify the name and location of your class file. The file name defaults to the name of the current class, with a **.cls** suffix. The location defaults to your Jade working directory.
3. Click on the **Schema Options** tab to display the **Schema Options** sheet and then specify what you want to be extracted. For details, see "Specifying Your Schema Options", earlier in this chapter.

Extracting a Selected Method

To save and extract the method that is currently selected in the Methods List

1. Select the **Extract** command from the Methods menu. The Extract dialog **Extract Options** sheet is then displayed.
2. Click the **Browse** button at the right of the **Schema File Name** text box. The common Save As dialog is then displayed, to enable you to specify the name and location of your method file. The file name defaults to the name of the current type and method, with an **.mth** suffix (for example, when extracting the **Customer** class **getAddress** method, the default file name is **Customer_getAddress.mth**). The location defaults to your Jade working directory.
3. Click on the **Selective** tab to display the **Selective** sheet and then specify the methods you want to extract. For details, see "Specifying Selective Options", earlier in this chapter.
4. Click on the **Schema Options** tab to display the **Schema Options** sheet and then specify what you want to be extracted. For details, see "Specifying Your Schema Options", earlier in this chapter.

Extracting All User-Defined Methods in a Selected Primitive Type

To extract all user-defined methods in the type selected in the Types Browser

1. Select the **Extract** command from the Types menu. The Extract dialog **Extract Options** sheet is then displayed.
2. Click the **Browse** button at the right of the **Schema File Name** text box. The common Save As dialog is then displayed, to enable you to specify the name and location of your primitive type file. The file name defaults to the name of the current primitive type, with a .**cls** suffix. The location defaults to your Jade working directory.
3. Click on the **Schema Options** tab to display the **Schema Options** sheet and then specify what you want to be extracted. For details, see "Specifying Your Schema Options", earlier in this chapter.

Extracting All Schema-Defined Methods

In addition to the **Extract** command in the Methods menu that enables you to extract only the method selected in the Methods List of the Class Browser or Primitive Types Browser to a .**mth** file, the Methods menu in other browsers (for example, the Methods View, Messages, References, Methods, and Methods Status List browsers) provide the **Extract All Schema-Defined** command. Use this command to extract all schema-defined methods displayed in that methods list to a partial schema (.**scm**) file.

To extract all schema-defined methods in the current Methods List

1. Use the **Extract All Schema-Defined** command from the Types or the Methods menu. The Extract dialog **Extract Options** sheet is then displayed.
2. Click the **Browse** button at the right of the **Schema File Name** text box. The common Save As dialog is then displayed, to enable you to specify the name and location of your partial schema file. The file name defaults to the name of the current schema, with a .**scm** suffix. The location defaults to your Jade working directory.
3. Click on the **Selective** tab to display the **Selective** sheet and then specify the methods you want to extract. For details, see "Specifying Selective Options", earlier in this chapter.
4. Click on the **Schema Options** tab to display the **Schema Options** sheet and then specify what you want to be extracted. For details, see "Specifying Your Schema Options", earlier in this chapter.

See also "Extracting Selected Entities", in Chapter 2, "Change Control", of the *Development Environment Administration Guide*.

Caution Ensure that you extract the schema-defined methods to a location different from that of your source schema or change the partial schema file name to differentiate it from the full schema (for example, **DemoSchemaAllMethods**).

Extracting Schemas as a Non-GUI Client Application

You can use the **JadeBatchExtract** application in the **jadclient** non-GUI client application to automate the extraction of your user-defined schemas, passing command line arguments after the **startAppParameters** parameter to specify your extract requirements. For details about:

- The **jadclient** non-GUI client application, see "Running a Non-GUI Client Application using jadclient", in Chapter 1 of the *Runtime Application Guide*.
- Extracting schemas in sections (for example, to store schemas in Subversion or any other suitable version control system in smaller chunks than complete schema files), see "Extracting Schemas in Sections", in the following subsection.

- Extracting a specified class from the current or latest schema context, see "Extracting Classes as a Non-GUI Application", later in this chapter.
- Extracting and loading a patch history, see "Extracting and Loading a Patch History", in Chapter 3 of the *Development Environment Administration Guide*.

To extract schemas as a non-GUI application, specify the following parameters in the **jadclient** program.

```
jadclient path=database-path
          ini=jade-initialization-file
          schema=JadeSchema
          app=JadeBatchExtract
          startAppParameters
          extract-arguments
```

The arguments that you can specify after the **startAppParameters** parameter are listed in the following table.

Argument	Format	Description
Class	*schema-file-name forms-file-name schema-name class-name*	Extract a class from the current schema context of the specified schema and type
CL	*schema-file-name forms-file-name schema-name class-name*	Extract a class from the latest schema context of the specified schema and type
Multiple	*multiple-schema-file-name schema-name-list*	Multiple schema extract from the current schema context for the listed schemas
ML	*multiple-schema-file-name schema-name-list*	Multiple schema extract from the latest schema context for the listed schemas
Patch	*schema-file-name forms-file-name patch-number schema-name*	Extract a patch from the current schema context of all changes made against the patch number for the specified schema
PL	*schema-file-name forms-file-name patch-number schema-name*	Extract a patch from the latest schema context of all changes made against the patch number for the specified schema
Version	*multiple-schema-file-name patch-version-number*	Extract a patch from the current schema context of all changes made to all schemas against the patch number
VL	*multiple-schema-file-name patch-version-number*	Extract a patch from the latest schema context of all changes made to all schemas against the patch number
Single	*schema-file-name forms-file-name schema-name*	Extract from the current schema context of the specified schema

Argument	Format	Description
SL	*schema-file-name forms-file-name schema-name*	Extract from the latest schema context of the specified schema
File	*schema-file-name forms-file-name parameter-file-name schema-name*	Extract from the current schema context using a parameter file
Every	*multiple-schema-file-name*	Multiple schema extract from the current schema context of every user-defined schema
EL	*multiple-schema-file-name*	Multiple schema extract from the latest schema context of every user-defined schema
B	*schema-file-name forms-file-name schema-name*	Extract a stub version of a schema from the current schema context of a schema containing a package
BL	*schema-file-name forms-file-name schema-name*	Extract a stub version of a schema from the latest schema context of a schema containing a package
<encrypt>	"<encrypt>"	Encrypted schema source files
delta	*delta-identifier*	Extract a specific delta when extracting a schema from a parameter file
<compactddx>	<compactddx>	Optionally extract a compact DDX (XML) file

For details about extracting:

- Multiple schemas or extracting selected classes and methods to a parameter file, see "Extracting Multiple Schemas" or "Specifying Your Parameter File Options", earlier in this chapter.
- Patch versions, see "Extracting a Patch Version", in Chapter 3 of the *Development Environment Administration Guide*.
- A version of the schema that contains only stub versions of each package, see "Stub Packages", in Chapter 8 of the *Developer's Reference*.

When specifying extract arguments:

- When extracting a patch, you must specify a valid patch number.

  Note When performing a batch extract by patch number (for single or multiple schemas), the Jade command file (JCF) is always created.

- For all types of batch extract other than an extract by patch number, the command file is not created by default. If you want the command file to be created, you must specify this on the command line; for example:

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini server=singleUser
schema=JadeSchema app=JadeBatchExtract startAppParameters Single
c:\temp\myTest.scm c:\temp\myTest.ddb Test "<jcf>"
```

The **<jcf>** value indicates that you want the command file created. As you can specify more than one option between the **<** and **>** characters, you must separate each one with a comma; for example, if you want to encrypt the source and create the command file, your command would look similar to the following.

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini server=singleUser
schema=JadeSchema app=JadeBatchExtract startAppParameters Single
c:\temp\myTest.scm c:\temp\myTest.ddx Test "<encrypt,jcf>"
```

The **<compactddx>** option enables you to extract compact DDX files; for example:

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters CL "c:\temp\WidgetForm.cls"
"c:\temp\WidgetForm.ddx" Sales Widget "<ddxFormat,compactddx>"
```

Compact DDX files exclude properties with null values for applications, forms, standard controls, and RPS mappings. Compact DDX files reduce the size of the extracted file and improve loading performance.

Notes The **<compactddx>** option has no effect when extracting DDB (legacy) format files.

Compact DDX files extracted in Jade 2022.0.02 and higher will not load into Jade 2022.0.01 or any earlier Jade releases.

- Separate the argument and its associated values with a space.
- Specify the fully qualified schema file and forms file names.
- Enclose command line arguments that contain spaces in double (**""**) or single (**'**) quotation marks.
- The **jadclient** program treats processing arguments enclosed in double (**""**) or single (**'**) quotation marks after the **startAppParameters** parameter as single-string entries in the **HugeStringArray**. The handling of strings in this **HugeStringArray** is application-specific. For example, **dir=program files** is treated as a two-string entry and **dir="program files"** is treated as a one-string entry. How these entries are handled is determined by your application.
- Because the **<** and **>** characters have special significance in Windows when you extract encrypted schema sources, you must enclose the **<encrypt>** argument within double quotation marks (that is, **"<encrypt>"**). In addition, the **<encrypt>** argument must be the last argument after the **startAppParameters** parameter.
- The extract arguments and the associated parameters are case-sensitive (that is, **Multiple** is a valid argument but **multiple** is not).
- If the multiple schema file name for the extraction of multiple schemas does not exist, the file is created.

Note The schema names specified in the schema names list is the name of the schema itself, which may differ from the name of the schema metadata (.**scm**) and form and data definition (.**ddb** or .**ddx**) files. For example, if your **Test** schema is contained in **ProdTest.scm** and **ProdTest.ddb** files, specify **Test** following the schema file name in the command line for a multiple schema extraction. The **Test** schema is then extracted to **Test.scm** and **Test.ddb** files during the non-GUI client extract process.

- You can extract (and load) form and data definition files in the legacy DDB format instead of the default XML DDX format that does not include Jade oids. All entities in the XML (DDX) format are identified by name and by their position in the XML object hierarchy. You can compare the DDX file to another version of the file, to identify what has changed between the two versions.
- When extracting classes or methods of a specific schema to a parameter file using the **File** argument, the .**unl** file containing the elements defined in the specified schema file name must already exist. If you do not specify the path and name of a valid parameter file, an error is output.

- You can specify a delta code when extracting a schema from a parameter file, which enables you to automate the checking in of a delta at run time so that you can create the release of a change for testing purposes. To set a delta for a batch extract, the last command line parameter must be **delta=*delta-identifier***; for example, if the delta identifier is **DeltaTwo**, the command line is as follows.

```
jadclient path=e:\jadesystems\jade6211\system ini=c:\jade\system\jade.ini
app=JadeBatchExtract schema=JadeSchema server=singleUser startAppParameters
File d:\temp\delta.scm d:\temp\delta.ddb|ddx d:\temp\param.unl
ErewhonInvestmentsViewSchema delta=DeltaTwo
```

Note You cannot use the **jadclient** program to automate the extraction of schemas from a deployed environment, as the internal system-only **JadeSchema** schema from which batch extractions are run is not present in deployed databases.

The following examples show the use of the **jadclient** program to extract schemas.

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters Every c:\temp\myschemas.mul
// The EL argument is a synonym of the Every argument

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters File c:\jade\test\Sales.scm
c:\jade\test\Sales.ddb|ddx c:\SalesBits.unl Sales

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters SL "c:\temp\Sales.scm"
"c:\temp\Sales.ddx" Sales

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters Version c:\temp\versionchgs.mul 4

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters Multiple c:\temp\selected.mul Sales
Accounts Orders

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters Single "c:\dir with space\Sales.scm"
"c:\dir with space\Sales.ddb|ddx" Sales

jadclient path=r:\jade\system ini=r:\jade\system\myjade.ini server=multiUser
host=127.0.0.1 port=6005 schema=JadeSchema app=JadeBatchExtract
startAppParameters Single Sales.scm Sales.ddb|ddx Sales

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini server=singleUser
schema=JadeSchema app=JadeBatchExtract startAppParameters Single
c:\temp\myTest.scm c:\temp\myTest.ddb|ddx Test "<encrypt>"

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters Patch c:\jade\Test\Sales.scm
c:\jade\test\Sales.ddx 2 Sales

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters B c:\temp\S1Stub.scm
c:\temp\scmS1.ddb|ddx S1

jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
```

```
app=JadeBatchExtract startAppParameters PL c:\Extracts\SpeedSend.scm
c:\Extracts\SpeedSend.ddb|ddx 5 SpeedSend
```

Extracting Schemas in Sections

You can use the **JadeBatchExtract** application in the **jadclient** program to extract schemas in sections. This enables you to store schemas in Subversion or in any other suitable version control system in smaller chunks than complete schema files.

To extract schemas in sections, specify the optional **"<sections>"** argument in the **JadeBatchExtract** application for the **Every** or **EL**, **Multiple** or **ML**, and **Single** or **SL** extract arguments.

The following example extracts in sections every schema to directory **C:\temp2\userSchemas**.

```
jadclient path=e:\jadesystems\jade\system
          ini=c:\jade\system\jade.ini
          app=JadeBatchExtract
          schema=JadeSchema
          server=singleUser
          startAppParameters
          Every C:\temp2\userSchemas\schemas.mul "<sections>"
```

When a schema is extracted in sections, a ***schema-name*** folder is created in addition to the ***schema-name*.scm** and ***schema-name*.ddb** or ***schema-name*.ddx** files. Within the folder are a number of .**sec** files, a **Types** subfolder, and possibly **addedFiles** and **deletedFiles**.

The **Types** subfolder contains a ***.cls** file for each type (that is, class, primitive type, and interface) in the schema. If a class is a **Form** class, there is also a .**ddx** file, which is an XML format form representing the layout of the form in every locale.

The .**sec** files, which do not have to be present, can be broken into the following two types.

- Those that represent the corresponding section of the .**scm** file, as follows.
  - schemaDefinitions.sec
  - importedPackageDefinitions.sec
  - constantDefinitions.sec
  - localeDefinitions.sec
  - libraryDefinitions.sec
  - externalFunctions.sec
  - inverseDefinitions.sec
  - databaseDefinitions.sec
  - schemaViewDefinitions.sec
  - _exposedListDefinitions.sec
  - exportedPackageDefinitions.sec
  - externalFunctionSources.sec

- Those that contain a list of types whose .**cls** file contributes to the corresponding section of the .**scm** file, as follows.
  - typeHeaders.sec
  - interfaceDefs.sec
  - membershipDefinitions.sec
  - typeDefinitions.sec
  - externalKeyDefinitions.sec
  - memberKeyDefinitions.sec
  - typeSources.sec

If **addedFiles** and **deletedFiles** files are created, these contain a list of .**cls** or .**ddx** files that have been added or deleted during the current extract, by comparing with the files that were present before the extract began. These files can be useful when you need to add new files to or delete files from the version control system. For example, in Subversion you could use these as arguments to the delete and add commands of **svn**, as shown in the following examples.

```
svn delete --targets deletedFiles

svn add --targets addedFiles
```

Rejoining Sections of an Extracted Schema

You can use the **JadeJoinSchema** application in the **jadclient** program to rejoin extracted schema sections into a .**scm** file. The following example, based on the example under "Extracting Schemas in Sections", in the previous section, joins the schemas that were extracted in sections to directory **C:\temp2\userSchemas**.

```
jadclient path=e:\jadesystems\jade6216\system
          ini=c:\jade\system\jade.ini
          app=JadeJoinSchema
          schema=JadeSchema
          server=singleUser
          startAppParameters
          c:\temp2\userSchemas\UserSchema
          c:\temp2\UserSchema.scm
```

In this example, **c:\temp2\userSchemas\UserSchema** is the folder in which the schema sections of the existing schema **UserSchema** are located and the resulting schema file containing the rejoined sections will be located in the **c:\temp2\UserSchema.scm** directory.

Extracting Classes as a Non-GUI Client Application

You can use the **JadeBatchExtract** application in the **jadclient** non-GUI client application to automate the extraction of a class in the *current* schema context (by specifying the **Class** argument) or the *latest* schema context (by specifying the **CL** argument) of the specified schema and type after the **startAppParameters** parameter to specify your class extract requirements.

The case-insensitive options that you can specify between **<** and **>** characters when extracting a class are listed in the following table.

Option	Description
<compactddx>	Extracts DDX (XML) format files in compact format, which excludes properties with null values for applications, forms, standard controls, and RPS mappings. Compact DDX files reduce the size of the extracted file and improve loading performance. It has no effect when extracting in DDB (legacy) format.
<ddbformat>	Force the forms and mapping files to be in the DDB (legacy) format.
<ddxformat>	Force the forms and mapping files to be in the default DDX format.
<nativeencoding>	Encode the extract files as ANSI for a native ANSI system or UTF-16LE (**LittleEndian** architecture) for a native Unicode system. This option is the opposite of specifying the **utf8encoding** option. **Note** Unicode DDB and DDX files are still extracted as UTF-8 due to portability and parsing considerations.
<nomodifiedinfo>	A specialist option to strip **setModifiedTimeStamp** lines from extracted source files. This can reduce the noise when storing source in Git or another Version Control System (VCS). **Note** This has the drawback that this information is not preserved or available in the extract files, which will cause features dependent on the modified timestamp and user information to be inaccurate.
<nomodelnumbers>	A specialist option to strip schema model number information (for example, **number=1, ordinal = 1**) from extracted source files. This can reduce the noise when storing source in Git or another Version Control System (VCS). **Note** This has the drawback that this information is not preserved or available in the extract files. While this is often not an issue for development purposes, it can cause issues when deploying to an existing production system if the model numbers are not reinstated.
<utf8encoding>	Encode the extract files as UTF-8 with a Byte Order Mark (BOM). This encoding is particularly useful for Unicode systems, as the extracted files can be stored in Git as a text file rather than the native UTF-16LE (**LittleEndian** architecture) encoding that Git would interpret as a Binary file. It also means that the same information extracted from both ANSI and Unicode systems produces identical extract files.

When extracting a class, you can specify multiple case-insensitive options between the **<** and **>** characters, with each one separated with a comma; for example, **<encrypt,ddxFormat,utf8encoding>**.

For details about specifying extract arguments, see "Extracting Schemas as a Non-GUI Client Application", earlier in this chapter.

The following example shows the use of the **jadclient** program to extract a class.

```
jadclient path=c:\jade\system ini=c:\jade\system\jade.ini schema=JadeSchema
app=JadeBatchExtract startAppParameters CL "c:\temp\WidgetForm.cls"
"c:\temp\WidgetForm.ddx" Sales Widget "<ddxFormat,compactddx,utf8Encoding>"
```

Loading Your Schema

You can load (install) an extract file into the current schema at any time; for example, when you are:

- Restoring a full or partial schema
- Passing code to another Jade developer
- Fully reconstructing a schema in the same Jade release

You can load:

- A full or partial schema or multiple schemas extracted to a single file
- Forms in a schema
- An extracted schema view
- An extracted class, method, primitive type method, external database schema, relational view, RPS mapping, .NET exposure, or ActiveX type library
- Patch version changes

For details about the order in which files should be loaded into a deployed database, see "Before You Get Started", in the *Schema Load User's Guide*.

Notes You can load a schema or forms file separately, if required.

A schema view is not set to the current view during the load process. (For details about setting a schema view, see "Setting a Schema View", in Chapter 3.)

If you have mapping logic on subclassed controls, other processes such as the Jade Painter, Translator utility, or the loading of schemas may also execute that logic. The logic therefore may need to perform checks to determine if it is running in the user application environment, to ensure that exceptions are not generated in these other situations.

As schema and forms definition files are treated as binary files, if the File Transfer Protocol (FTP) is used to transfer schema and forms definition files between machines, you must ensure that the transfer is done in binary mode (rather than ASCII) to prevent the removal of carriage return characters and the failure of the schema load process, particularly when schemas are encrypted.

You can also load an external database schema, ActiveX type library, relational view, or an RPS mapping. For details, see "Loading an External Database Schema", in Chapter 3 or "Extracting and Loading ActiveX Schema Definition Data" or "Extracting and Loading .NET Schema Definition Data", in Chapter 16, "Loading a Relational View", in Chapter 9, or "Loading an RPS Mapping", in Chapter 15, respectively. For details about loading an extracted class, method, or extracted Jade Report Writer file in a multiple schemas file (**.mul**), see "Multiple Schema File Syntax", earlier in this chapter.

If a schema load is attempted when a reorganization is in progress (regardless of whether the reorganization progress dialog is displayed), the load fails. For details about reorganizing schemas, see "Reorganizing Your Schema", in Chapter 3.

You can load form and data definitions in XML DDX format instead of the legacy DDB format when the **Extract as DDX (xml format) instead of DDB** check box on the **Source Management** sheet of the Preferences dialog is checked (it is unchecked, by default) and form and data definitions were extracted with the **Forms/Mappings as XML (ddx file)**check box on the Extract dialog checked.

From Jade 2020, the DDX extract process extracts all defined non-virtual primitive properties and primitive array properties on controls. (Virtual properties on other included classes are not extracted.) This change means that if you load DDX files into Jade 2020 and higher that were extracted in Jade 2018.0.01, there will be many differences in the contents of the DDX file.

DDX files extracted in Jade 2022.0.01 will not load into the Jade 2020 or 2018 releases because they contain additional information and therefore are not backwards-compatible. In addition, compact DDX files extracted in Jade 2022.0.02 and higher will not load into Jade 2022.0.01 or any earlier Jade releases.

The XML format does not include Jade oids. All entities are identified by name and by their position in the XML object hierarchy. You can compare the DDX file to another version of the file, to identify what has changed between the two versions.

The first line of a .**ddx** file has the **<?xml...** header. The format of the second line is:

```
<schema name="schema-name" JadeVersionNumber="Jade-version" [Compact="true"]
JadePatchNumber="patch-number" CompleteDefinition="true|false">
```

The following is an example of the first and second lines of the XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<schema name="CalculatorSchema" JadeVersionNumber="18.0.01" JadePatchNumber="0"
CompleteDefinition="true">
```

The schema name, which specifies the schema the information is for, must be included.

The **JadeVersionNumber** tag identifies the version of Jade that was used to produce the file.

The **Compact="true"** tag and value are displayed only if the file was extracted as a compact DDX file.

The **JadePatchNumber** tag specifies the patch number to use for the load; otherwise the current patch number is used.

The **CompleteDefinition** tag, which must be present, specifies whether the file is a complete definition for the whole schema or it is a partial schema. If the value of the **CompleteDefinition** tag is **true**, any existing entities not included in the file are deleted.

The form and data definition (.**ddb** or .**ddx**) load process displays a warning message if a control does not have a property reference on the form when handling form translations, the form becomes invalid, and a **jommsg.log** entry with the following format is output.

```
******Warning: Control name on form name has no form control reference.
```

Ensure that the schema metadata (.**scm**) and form and data definition (.**ddb** or .**ddx**) files match.

For details about automating the loading of schemas in batch mode (for example, from a command script), see "Loading Schemas in Batch Mode using jadloadb" or "commandFile", in the *Schema Load User's Guide*.

Merging a Schema in the Load Process

You can load a schema file containing a new definition of a schema that already exists in your Jade database.

When you perform a load, the existing schema is updated to match the definition in the schema file. Specifically, the following actions are performed.

- New schema elements (for example, global constants, classes, interfaces, methods, or properties) are added.
- Changed schema elements are updated.

- Schema elements that are present in the database but not in the schema file are deleted, subject to user confirmation. For more details, see "Specifying Advanced Load Options", later in this chapter.

Note An application in the database that is used in a package but is not in the schema file is not deleted.

For example, consider the following two definitions of a class (one in the Jade database and the other in the incoming schema file).

Database	Schema File
Customer	Customer
Attributes	**(**
address: String[31];	attributeDefinitions
balance: Decimal[12,2];	balance: Decimal[12,2];
customerNumber: Integer;	customerNumber: Integer;
longName: String[21];	longName: String[31];
	phone: String[16];
Jade Methods	
compareDetails(cust: Customer);	jadeMethodDefinitions
display(): String;	display(): String;
	printOn(f: Form);
	)

In this example, the following actions are performed during the load process.

- **balance**, **customerNumber**, and **display** are unchanged
- **phone** and **printOn** are new, so they are added to the existing class
- The length of **longName** has been changed from 21 to 31, so this change is applied
- **address** and **compareDetails** are not defined in the schema file, so they are deleted from the class

The class that exists in the database is modified so that it becomes identical to the schema file definition.

Changes Requiring Data Reorganization

If properties of a class are changed as a result of loading a schema file and that class has instances, the class is marked as requiring reorganization.

You must reorganize the class from within Jade, by selecting the schema in the Schema Browser and then selecting the **Reorg** command from the Schema menu. If you are loading a schema file by using the Schema Load utility (**jadload**), the reorganization (if required) is initiated automatically at the end of the load process.

The following changes to a class cause that class to be marked for reorganization.

- Adding a property
- Deleting a property
- Changing the type or length of a property

- Adding an inverse to an existing property
- Changing an existing inverse reference
- Changing the keys of a dictionary
- Changing the tuning of a class with instances

An error is raised during the load process if the superclass of a class in the incoming file has changed. For details about reorganization, see Chapter 14 of the *Developer's Reference*.

Loading Partial versus Complete Schema Definitions

When loading a complete schema definition file, new locales that are in the schema and forms definition files but not in the database are loaded into an existing schema. When loading a partial schema definition file, you must create the new base locales before loading the schema. (For details, see "Adding Locales" under "Maintaining Locales for Your Schema", in Chapter 11.) If you do not do so, the new locales are ignored.

When a selective extract is performed, the resulting schema file contains only selected parts of the schema, and class and interface definitions in the schema file may define entire classes or interfaces or only selected methods from a class or interface. Clearly, we would not want to delete missing elements in this situation, where only a partial schema, class, or interface definition has been extracted. For this reason, the **completeDefinition** and **partialDefinition** keywords are used to identify whether an extract is complete or partial. These keywords have the following effect.

- If the schema definition in a schema file is qualified with **completeDefinition**, the schema file is assumed to define a complete schema and any existing classes, interfaces, global constants, translatable strings, locale formats, or libraries that are not present in the schema file are deleted from the database (that is, the existing schema is modified so that it exactly matches the incoming schema file).
- If the schema definition in a schema file is qualified with **partialDefinition**, the schema file is assumed to define a partial schema only and no existing classes, interfaces, global constants, translatable strings, locale formats, or libraries are deleted. The load process modifies existing elements and adds new elements only.
- If a class or interface definition in a schema file is qualified with **completeDefinition**, the class or interface definition is assumed to be complete and any existing properties, methods, constants, or dictionary keys that are not present in the class or interface definition are deleted from the class or interface in the database (that is, the existing class or interface is modified so that it exactly matches the class or interface definition in the incoming schema file).
- If a class or interface definition file is qualified with **partialDefinition**, the class or interface definition is assumed to be partial, and no existing properties, methods, constants, or dictionary keys are deleted. The load process modifies existing elements and adds new elements only.

Note An application in the database that is used in a package but is not in the schema file is not deleted.

Confirmation of Deletion

Even if a class, interface, or schema definition is identified as being complete, you may not want to delete missing elements. For example, you might extract the definition of a class or definition and pass it on to another developer. In the meantime, that developer may have added methods of his or her own to that class or interface. (In a sense, developers each have their own "complete" definition of the class or interface, and these must be merged.)

The second developer does not want methods to be deleted when the extracted class or interface that you provided is loaded. The situation may be even more serious at the schema level, where entire classes or interfaces may be deleted inadvertently without some safeguards. To protect against this possibility, you are prompted to confirm deletion of elements when you load a schema file.

A confirmation dialog is displayed for each element or group of elements that are candidates for deletion. The Confirmation dialog provides the buttons listed in the following table.

Button	Action
Yes	Confirms the current deletion only.
Yes to All	Confirms the current and all subsequent deletions. No further confirmation is requested.
No	Prevents the current deletion only.
No to All	Prevents the current and all subsequent deletions. No further confirmation is requested.
Cancel	Cancels the load.

By default, you are prompted for confirmation of deletion. To override this default action, select the appropriate option button from the Schema Merge group box of the Advanced Load Options dialog. For details, see "Specifying Advanced Load Options", later in this chapter.

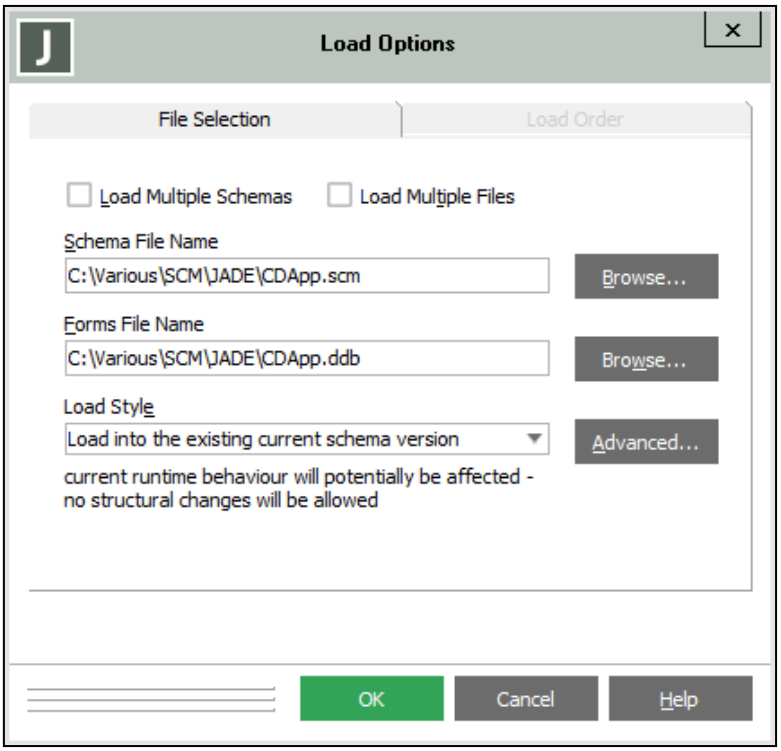
For details about using target arguments to control the behavior of schema element deletion when loading a schema using the Schema Load utility (**jadload**), see "Loading Schemas using the Schema Load Utility", in the *Schema Load User's Guide*.

Invoking the Load Process

To load a full or partial schema

- Access the Load Options dialog by performing one of the following actions.
  - Select the **Schema Load** command from the File menu.
  - Select the **Load** command from the Schema menu.
  - Click the **Load** toolbar button.
  - Press Ctrl+Alt+L.

The Load Options dialog, shown in the following image, is then displayed.



Specifying Your Load Options

The Load Options dialog enables you to specify the type of file that you are loading and the location and name of the file.

The **File Selection** and the **Load Order** sheets enable you to select the schema, file, schemas, or files to be loaded, and when loading multiple files, the order in which the files are loaded. For details, see the following subsections.

- Selecting the File or Files to Load
- Specifying the Load Order when Loading Multiple Files
- Specifying Advanced Load Options

If you are unsure of a file name or location, click the **Browse** button. The common File dialog that is then displayed lists all schema, class, method, forms definition files, and so on, in the directory for which the dialog is opened, to enable you to select the appropriate file.

When you have specified your load options, click the **OK** button to confirm your selections. (Alternatively, click the **Cancel** button to abandon your selections.)

The load process is then initiated. If the schema file contains methods that are in error when loading a single schema, the file is loaded without interruption.

On completion of the file load, a message box:

- Displays the number of methods that are in error, and enables you to view those methods. If the file contains methods that are in error when loading multiple schemas from a multiple schemas extract file, the file is loaded without interruption.

- Informs you that the multiple load completed and lists the schemas that have methods in error, if applicable. You are advised to use the Method Status List window for each schema to display and correct these errors.

Any errors outside method implementation (for example, an invalid class or property definition) result in a halt of the schema load process. The editor window then displays the extract file with the error highlighted. The error description is displayed in the status line. Amend the error, and then click the **OK** button to restart the load process. Alternatively, you click the **Cancel** button to abort the load.

A compile error is raised during the load process if the file contains more than one **Application** or **Global** subclass. If the extract file does not define **Application** or **Global** subclasses, the Add Schema dialog is displayed, to enable you to define an application and global class. If you do not specify these classes, the load is cancelled. (For more details, see "Defining a Schema", in Chapter 3.) When the load process is complete, you are now ready to use your newly loaded schema, application forms, class, interface, method, or primitive type method.

To cancel the load process

- Click the **Cancel** button in the Add Schema dialog.

Selecting the File or Files to Load

You can drag a file or files from the Windows Explorer and drop one or more files onto the **Load** button on the Jade Platform development environment toolbar or the Schema Browser list box, to open the Load Options dialog and populate the **Schema File Name** text box with the file or files. If you drop:

- A single file with a .**mul** file type, the **Load Multiple Schemas** check box is checked.
- More than one file, the **Load Multiple Files** check box is checked and all of the dropped file names are displayed in the **Schema File Name** text box, so that you can add more files, if required.

Dropping a folder is rejected.

You can drag files from the Windows Explorer and drop them on the Load Options dialog **Schema File Name** and **Forms File Name** text boxes. Dragging and dropping a file on the **Schema File Name** text box has the following effect.

- If a single .**mul** file is dropped on the text box, the **Load Multiple Schemas** check box is checked and the text box displays the name of the multiple schemas file. (The file must be of type .**mul**, for this action to be recognized.)
- If a single schema file is dropped on the text box and the **Load Multiple Schemas** check box is unchecked, the name of the dropped schema file replaces any file that was already displayed in the text box.
- If a single file is dropped on the text box and the **Load Multiple Files** check box is checked, the file is added to the list of file names in the **Select Multiple File Names** text box.
- If multiple form and data definition (.**ddb** or .**ddx**) or method (.**mth**) files are dropped and the **Load Multiple Files** check box is unchecked, the **Load Multiple Files** check box is checked, any previously selected files are cleared, and all of the dropped file names are displayed in the **Select Multiple File Names** text box.
- If multiple files are dropped and the **Load Multiple Files** check box is checked, the files are added to the list of files names displayed in the **Select Multiple File Names** text box.

Dropping a single form and data definition (.**ddb** or .**ddx**) file on the **Forms File Name** text box displays that file name, replacing any previously displayed forms definition file or files. Dropping multiple files on the **Forms File Name** text box in the same action is rejected.

Applies to Version: 2018.0.01 and higher

Notes You can drop only files on the **Schema File Name** and **Forms File Name** text boxes; that is, dragging and dropping a folder is rejected.

The file types are not verified, as it is your responsibility to do so.

To specify your load options on the File Selection sheet

1. Check the **Load Multiple Schemas** check box if you want to load an extract file containing multiple schemas. A single schema is loaded by default; that is, this check box is unchecked.

   The **Multi Extract File Name** text box is then displayed instead of the **Schema File Name** text box, and the **Forms File Name** text box is disabled. (A multiple schemas extract file itself contains merely a list of extracted files, with each extracted schema having a separate pair of **.scm** and .**ddb** or .**ddx** files.)

2. Check the **Load Multiple Files** check box if you want to load multiple files. A single file is loaded by default; that is, this check box is unchecked.

   The common Open dialog is then displayed, to enable you to select the schema files (that is, any combination of .**scm**, .**ddb** or .**ddx**, .**cls**, .**mth**, or .**mul** files) that you want to load into your Jade database. When you have selected the files from the appropriate directory or directories, click the **Open** button.

   The **Select Multiple File Names** text box containing your selected files is then displayed instead of the **Schema File Name** text box, the **Forms File Name** text box is disabled, and the **Load Order** sheet is enabled.

3. If you are loading a single schema, in the **Schema File Name** text box, specify the name and location of the schema file that you want to load. You must specify a value in this text box. An error is raised if you do not specify the name of an existing file or if the file cannot be accessed.

   If you are loading multiple schemas from a multiple schemas extract file, in the **Multi Extract File Name** text box, specify the name and location of the multiple schemas extract file you want to load (with a default **.mul** file suffix). You must specify a value in this text box. An error is raised if you do not specify the name of an existing file, if the file cannot be accessed, or the extract file and the schema metadata (.**scm**) and form and data definition (.**ddb** or .**ddx**) files are not located in the same directory. If you are unsure of your file name or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file.

4. If you are loading a single schema, specify the name and location of the extracted forms file in the **Forms File Name** text box if you want to load an extracted file containing application forms as a separate transaction in the same load process. An error is raised if the file cannot be accessed.

   Tip Click the **Forms File Name** text box to display the forms file with the same prefix as the schema file in the text box.

5. In the **Load Style** combo box, select the load style, as follows.

   - Load a new schema or load the schema as the latest schema version (a new version will be created if required), and allow structural changes. This is the default option.
   - Load the schema into the existing current schema version, which may potentially affect current runtime behavior. As structural changes are not allowed, the load will not proceed if structural changes are to be introduced. If you are loading a schema file containing only method changes into a multiuser system, select this option, to ensure that no structural changes are attempted that would require a reorganization of the database and impact existing users. If a structural change is detected, the schema load returns an error.
   - Load into latest versioning only structural changes, which loads only structural changes into the latest version and does not version methods or other non-structural entities. This may potentially affect current runtime behavior.

If you want to set the default load style that is selected in the **Load Style** combo box when the Load Options dialog is displayed, specify the **LoadStyleDefault** parameter and the optional **LoadStyleSecond** and **LoadStyleThird** parameters in the [Jade] section of the Jade initialization file. For details, see your *Jade Initialization File Reference*.

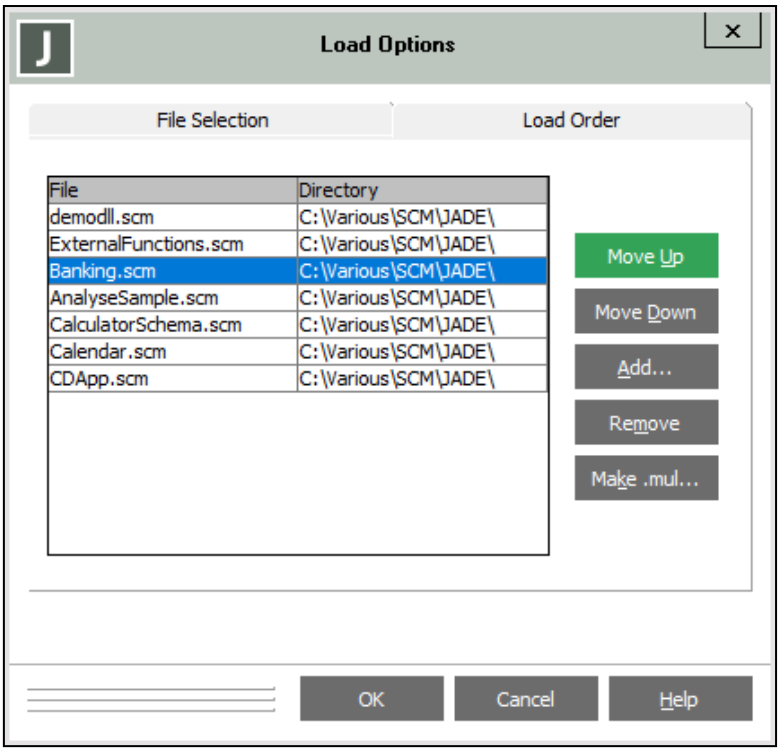
6. If you are loading multiple files and you want to specify the order in which the files are loaded, click the tab of the **Load Order** sheet. For details, see the following section.

7. If you want to select additional load options (for example, loading checked out methods or changing the target schema), click the **Advanced** button.

   The Advanced Load Options dialog is then displayed. For details, see "Specifying Advanced Load Options", later in this chapter.

Specifying the Load Order when Loading Multiple Files

The **Load Order** sheet, shown in the following image, is displayed when you click the **Load Order** tab on the Load Option dialog.



To specify your load order of multiple files

1. To move a file selected in the table at the upper left of the sheet up the file load order, click the **Move Up** button. The selected file is then moved up one position so that it is now positioned above the file that it previously followed in the file load process.

   Repeat this action until the file is positioned in the order in which you want it loaded; for example, to display the fifth file after the second file value, select the fifth file in the table and then click the **Move Up** arrow button three times.

2. To move a file selected in the table at the upper left of the sheet down the file load order, click the **Move Down** button. The selected file is then moved down one position so that it is now positioned below the file that previously followed it in the file load process.

   Repeat this action until the file is positioned in the order in which you want it loaded; for example, to display the second file after the fifth file, select the second file and then click the **Move Down** button three times.

3. To add a file to the multiple file load process, click the **Add** button.

   The common Open dialog is then displayed, to enable you to select the schema file or files (that is, one or more .**scm**, .**ddb** or .**ddx**, .**cls**, .**mth**, or .**mul** files) that you want to add to the multiple file load process. When you have selected the file or files from the appropriate directory or directories, click the **Open** button.

   The selected file or files are then displayed at the end of the list of selected files. (See step 1 of this instruction for details about moving an added file up the load order.)

4. To remove a file from the multiple load process, select the file that you do not want to load in the table and then click the **Remove** button.

5. To make the selected files in the table into a multiple load file, click the **Make .mul** button.

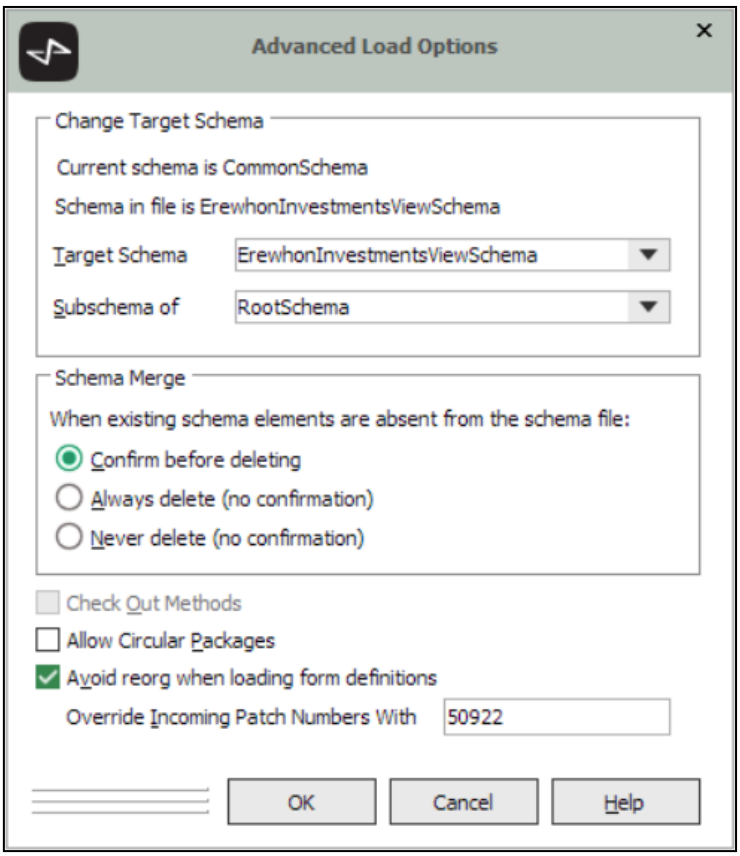
   The common Open dialog is then displayed, to enable you to specify the name of your multiple load file in the **File name** text box (for example, **TestScms.mul**) and to select the directory in which the schema files and the multiple load file is located.

   The multiple schemas load file itself contains merely a list of the file names in the order displayed in the table on the **Load Order** sheet.

   When you have specified the file and its location, click the **Open** button. A message box then advises you that the file does not exist, and prompts you to click the **Yes** button to create the file.

Specifying Advanced Load Options

The Advanced Load Options dialog, shown in the following image, is displayed when you click the **Advanced** button in the Load Options dialog.



The Advanced Load Options dialog enables you to specify a different target schema, the merge option for a partial schema, and the checking out of methods during the load.

Note If you are loading multiple schemas, the combo boxes in the Change Target Schema group box are disabled.

To specify your advanced load options

1. In the **Target Schema** combo box of the Change Target Schema group box, specify the new name for your schema or select the name of an existing schema from the list box name if you want to load into a schema other than the one defined in the schema file. This combo box is disabled if you are loading multiple schemas.
2. If you specified the name of a new schema in the **Target Schema** combo box, you can define its superschema, if required, by selecting the appropriate schema in the **Subschema of** combo box. (The **Subschema of** combo box is disabled if you specify the name of an existing schema in the **Target Schema** combo box.) This combo box is disabled if you are loading multiple schemas.
3. The options in the Schema Merge group box enable you to control the deletion of schema elements that are defined in the database but are absent from the schema file.

The **Confirm before deleting** option button is selected by default; that is, you are prompted to confirm that any schema element defined in the database but absent from the schema file is deleted from the existing schema. For more details, see "Merging a Schema in the Load Process", earlier in this chapter.

Select the **Always delete (no confirmation)** option button if you want to delete from the existing schema each element defined in the database but absent from the schema file. If this option is selected, you are not prompted for confirmation.

Select the **Never delete (no confirmation)** option button if do not want to delete from the existing schema any element defined in the database but absent from the schema file. If this option is selected, you are not prompted for confirmation.

For more details, see "Confirmation of Deletion", earlier in this chapter.

4. If a delta is set and it contains methods checked out prior to a schema load, they are updated with any change in the schema file.

   If you want to preserve your current system, set a delta with no methods checked out prior to a schema load, and make sure the **Check Out Methods** check box is checked (it is checked by default if a delta is set). Any changes to existing methods are then checked out to the currently set delta.

   The **Check Out Methods** check box is disabled if no delta is set in the schema.

5. In the **Override Incoming Patch Numbers With** text box, specify a value in the range **1** through **Max_Integer - 1** (that is, 2,147,483,646) if you want to override patch versioning in incoming schema entities. All entities loaded in the schema file then have their patch version number set to the specified value. When you specify a valid value, it is effective regardless of whether patch versioning is enabled for the schema.

   Note You can override the incoming patch number so that the specified patch number is used when loading multiple schemas or files.

   If the **PatchNumberRequired** parameter in the [JadePatchControlExtensions] section of the Jade initialization file is set to **false**, you do not have to specify a value in this text box if you want to retain the patch number specified in the schema file. If the parameter is set to **true**, you must enter a patch number.

   The specified value does not have to match an existing defined patch number. If the value is the same as an existing patch number, the state of the existing patch version is ignored (that is, the existing patch number in the schema file is closed).

   If patch control is enabled, the patch history will be stored. In addition, if patch control is enabled, the patch number set for a schema or a user is not affected by the override patch number, which is used only for the duration of the load process.

   The default value for the override patch number is the patch number set up for patch control. If patch control is disabled, the default value will be blank (that is, there is no override).

6. Check the **Allow Circular Packages** check box if you want to allow a circular dependency between packages in the schema hierarchy; that is, to permit the loading of an incomplete package (for example, **Schema1** exports **Package1** and imports **Package2**, while **Schema2** exports **Package2** and imports **Package1**).

   When you subsequently create a package that would result in circularities, you are prompted to confirm that you want to continue and allow a circular dependency between packages in the schema hierarchy.

   By default, this check box is unchecked; that is, the packages that are available for import are those that would not result in circular dependencies in the schema hierarchy.

If you load two schemas that are not circular and you load an importer before the exporter:

- An incomplete schema is loaded.
- The form and data definition file (.**ddb** or.**ddx**) is not loaded.
- Exception 8527 (*Load results in one or more incomplete schemas*) is raised.

  When the **StandardExitValues** parameter in the [FaultHandling] section of the Jade initialization file is set to **true**, you can map this exception to generic exit value **8**. (For details about generic exit values, see "Enabling the Use of Generic Exit Values for Windows", in Appendix A of the *Installation and Administration Guide*.)

Incomplete schema definitions and their usages are resolved by loading the export definition and then reloading the importing files. (To completely resolve all of the interdependencies, more than two iterations may be required.)

An incomplete schema is displayed with the background color of a versioned schema (which defaults to red) in the Schema Browser. The incomplete schema, its subschemas, Class Browser, and any wizards for that schema and subschemas, are not available. To make the Class Browser available, it is your responsibility to make the entire schema complete (by loading the exporting schema and then reloading the incomplete schema).

Note An incomplete schema can be versioned, changing the background color, and not allowing the opening of a Class Browser on either version.

7. Uncheck the **Avoid reorg when loading form definitions** check box (which is checked by default) if you want to leave the existing, source and code of .**ddb** or .**ddx** files untouched during a schema load process; that is, when loading changes in user-defined control classes with modified properties, the schema has to be reorganized before loading and then reorganizing the form definition files. (If a reorganization was not performed after loading and reorganizing the schema before loading the form definition files and performing another reorganization, the form data is not merged into the schema, resulting in loss of data on properties.)

   By default, this check box is checked; that is, when loading changes in user-defined control classes with modified properties, the schema has to be reorganized before the forms definition .**ddb** or .**ddx** files are loaded and then reorganized. The existing source for the .**ddb** or .**ddx** files is not removed and they are marked as needing compilation. This allows you to efficiently load in multiple schemas and forms definition files at once by performing minimal reorganizations at the end of the load process so that you can load a schema *and* form definition files while performing only one reorganization at the end of the load process without loss of data.

8. Click the **OK** button to confirm your selections. Alternatively, click the **Cancel** button to abandon your selections.

The Load Options dialog is then displayed, to enable you to change or confirm your load options.

Note If you changed the name of your target schema to the name of an existing schema that exists in the specified superschema, the schema is merged into the existing schema when it is loaded. For details, see "Merging a Schema in the Load Process", earlier in this chapter.

Encrypting Schema Source Files

The schema extract and load facilities provide hooks that enable you to encrypt the Jade method source code in your schema extract files so that you can release schema extract files without making their source code easily visible.

These encryption hooks enable you to incorporate data encryption algorithms of your choice, to make it difficult for anyone to view your method source code. For example, you may want to encrypt your source code before releasing a schema containing Jade applications for a third-party.

Jade provides a default encryption algorithm that is used if you do not specify an encryption library and you check the **Encrypt Sources** check box in the Extract dialog **Schema Options** sheet. You can specify an encryption library of your choice, by using the **SchemaEncryptionHookLibrary** parameter in the [JadeSecurity] section of the Jade initialization file.

Note As schema and forms definition files are treated as binary files, if the File Transfer Protocol (FTP) is used to transfer schema and forms definition files between machines, you must ensure that the transfer is done in binary mode (rather than ASCII) to prevent the removal of carriage return characters and the failure of the schema load process, particularly when schemas are encrypted.

For details about the encryption hook library parameter, see "Jade Security Section [JadeSecurity]", in the *Initialization File Reference* and for details about the encryption hooks, see "Schema Source File Security" under "Jade Security", in Chapter 2 of the *Object Manager Guide*.

Extracting Encrypted Schema Source Files

To indicate that extracted schema source files are encrypted, first check the **Encrypt Sources** check box in the Extract dialog **Schema Options** sheet. (For details, see "Specifying Your Schema Options", earlier in this chapter.) You can encrypt schema source files extracted to a multiple extract file, if required.

When extracting method source code and source encryption is enabled, the following occurs.

1. Jade looks in the **SchemaEncryptionHookLibrary** parameter in the [JadeSecurity] section of the Jade initialization file for the name of your user-defined encryption library. Jade uses a default encryption algorithm if you do not supply your own library.
2. Jade attempts to load the encryption library by calling **LoadLibrary** and gets the address of your predefined encryption hook routine by calling **GetProcAddress**. An error is raised if the library cannot be located.
3. The source code is passed to your specified encryption routine. The encryption routine can encrypt the source in any way, and can change its length. For example, the encryption routine could embed an identifier at the beginning of the source to identify the type of encryption algorithm that is used.
4. The encrypted source code is written to the extract file.

Loading Encrypted Schema Source Files

When loading a schema extract file containing encrypted source code, the following occurs.

1. Jade looks in the **SchemaEncryptionHookLibrary** parameter in the [JadeSecurity] section of the Jade initialization file for the name of your user-defined encryption library. Jade uses a default decryption algorithm if you do not supply your own library.
2. Jade attempts to load the encryption library by calling **LoadLibrary** and gets the address of your predefined decryption hook routine by calling **GetProcAddress**. An error is raised if the library cannot be located.
3. The compiler passes the encrypted source to your specified decryption routine, which restores the source code to its original form.
4. The method source code is compiled.

Note To retain confidentiality, the source code is not saved by the load process. If the load process therefore detects errors when compiling method source, only the error code and message are displayed. The developer of your Jade application must then make the appropriate modification to the source, before providing you with a new schema file.

For details about overriding patch versioning in incoming schemas, see "Specifying Advanced Load Options", earlier in this chapter.

Chapter 11 Internationalization

This chapter covers the following topics.

- Overview
- Maintaining Locales for Your Schema
  - Selecting Your Locales
  - Adding Locales
  - Removing Locales
  - Cloning Locale Forms and Strings
- Adding and Maintaining Formats
  - Adding a Short Date Format
  - Adding a Long Date Format
  - Adding a Time Format
  - Adding a Numeric Format
  - Adding a Currency Format
  - Maintaining Formats
  - Viewing References to Formats
  - Removing a Format
- Translating Strings
  - Using the String Browser
  - Adding Strings
  - Searching for Text in Existing Strings
  - Searching for Strings that Reference the Current String
  - Viewing References to a String
  - Running an Application from the String Browser
  - Handling Translatable Strings on Message Box Button Captions
  - Programmatically Maintaining Translatable Strings
- Translating Forms

- Jade Translator Utility
- Translating Messages

Overview

Jade provides you with the following facilities to enable you to internationalize (translate) Jade systems to meet your requirements.

- **Locales** command, to specify the language (locales) supported by your schema. (For details, see "Maintaining Locales for Your Schema", later in this chapter.)
- **Formats** command, to add and maintain format strings for use with primitive types in your schema. (For details, see "Adding and Maintaining Formats", later in this chapter.)
- **Strings** command, to maintain strings for the locales supported by your schema. (For details, see "Translating Strings", later in this chapter.)
- **Translate Forms** command, to enable you to translate forms for the locales supported by your schema. (For details, see "Translating Forms", later in this chapter.)
- Standalone Jade Translator utility, to enable you to translate strings and forms for a specified schema and locale. (For details, see "Jade Translator Utility", later in this chapter.)
- Translating messages and errors. (For details, see "Translating Messages", later in this chapter.)

The locale structure in each schema is independent of the locale structure in any other schema.

Each schema has a defined default locale, which you can change by using the Locales dialog or the **jadloadb** program and a command file. (For details, see "Maintaining Locales for Your Schema", later in this chapter, or "**commandFile**" under "Loading Schemas in Batch Mode using jadloadb", in the *Schema Load User's Guide*, respectively.)

Notes If you want to select the translation of all forms in the application from the schema default locale, regardless of the current locale of the application, set the **FormsUseDefaultSchemaLocale** parameter in the [Jade] section of the Jade initialization file to **true**.

Changing the default locale of a schema may take a significant amount of time if the schema has many subclass forms in subschemas.

The **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the Jade initialization file enables handling of standard client and presentation client locale-sensitive formatting of date, time, and numeric values (that is, numbers and currency) to be standardized and the regional overrides set on the presentation client to be forwarded to the application server so that both nodes use a consistent set of locale settings for the application when running in Jade thin client mode.

Enhanced locale support is enabled (or disabled) throughout a complete Jade environment, with the single setting in the Jade initialization file on the database node. When running with enhanced locale support, the locale of the presentation client must be present on the machine running the application server.

When using multiple locales, it is recommended that:

- The schema default locale should be the same as your current session locale in the Jade Platform development environment.
- All schemas in a schema hierarchy share that same locale structure.

For details about replacing a form in one locale with a copy of the form in another locale in the same schema, see "Copying a Form into another Locale", in Chapter 5. For details about forwarding regional overrides set on the presentation client to the application server so that both use consistent locale settings for the application, see the **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the Jade initialization file, in your *Jade Initialization File Reference*.

Forms Translation Styles

The **Schema** class read-only **formsManagement** property contains the style of forms management used by a schema and its subschemas.

You can change the forms definition style of an existing schema and its subschemas only by using the **Modify Schema** command with the **FormsManagementStyle** expression in a command file of the Jade Schema Load **commandFile**. For details, see the *Schema Load User's Guide*.

The forms management style, represented by **Schema** class constants, are as follows.

- Multiple form definitions, multiple translations (**FormsMngmt_Multi_Multi** (**0**))

  In this default style, each base locale has a full definition of each form. The translation (form data) of each form, using the Jade Translator utility, is built using the private definition of that form, which allows the position and size of each control along with its caption (if present) to vary between translations. (For details, see "Jade Translator Utility", later in this chapter.)

  Note This provides the maximum flexibility at the cost of the greatest maintenance effort.

- Single form definition, single translation (**FormsMngmt_Single_Single** (**1**))

  At run time in this style, Jade displays only a form from the default locale of the schema, regardless of the locale with which the user is running. It is your responsibility to write methods that display alternative form translations, by assigning a translatable string to caption text during the form **load** method.

  Controls are expected to be painted large enough to hold all translations. The loading of a form and data definition (.**ddb** or .**ddx**) file loads only form and control data from the default locale of the schema.

  Note This results in savings when loading forms definition (.**ddb** or .**ddx**) files into the Jade database.

- Single form definition, multiple translations using translatable strings (**FormsMngmt_Single_Multi** (**2**))

  In this style, both the Jade Painter in the Jade Platform development environment and the form data builder at run time detect the presence of a translatable string name and display the value of the translatable string from the appropriate locale.

  Controls are expected to be painted large enough to hold all translations. The Jade Painter maintains only a single definition of each form in the default locale of the schema.

  The extract of the forms definition (.**ddb** or .**ddx**) file extracts only the default locale of the schema, which is used by the form data builder to build a translation of the form for each base locale defined in the schema. At run time, Jade displays a form using the translation associated with the current locale of the user.

  Note This results in significant saving when displaying forms, as each of the property values listed in the following table do not have to be updated at run time and network traffic is therefore reduced compared to either of the other styles; that is, the value of each translation for the locale is placed in the form build data and in the transient object that is being constructed so that the values do not have to be fetched at run time.

You can use the Add String dialog or the String Browser to translate the painted values of the properties listed in the following table, and then select the translated translatable string in the Jade Painter to apply the required translation to the property. For details, see "Translating Strings", later in this chapter, or "Translating Control Properties", in Chapter 5.

Property	Class and subclasses of…
bubbleHelp	Window
caption	Button, CheckBox, JadeDockBase, Form, Frame, GroupBox, JadeMask, Label, MenuItem, OptionButton, Sheet, StatusLine
helpKeyword	MenuItem, Window
mask	JadeEditMask
text	JadeEditMask, TextBox

When the form is displayed at run time, each of these properties in the current schema and its superschemas are loaded. If any of these properties has a specified translatable string value, the **$** string name is replaced by the string translation value for the locale currently in effect. That value for the locale is placed in the form build data and in the transient object that is being constructed.

Compiling Translatable Strings

When a translatable string translation in the default locale is compiled successfully, any other translations of that translatable string that does not have the same parameter count is marked in error.

When a translatable string translation not in the default locale is compiled, the compile fails if the translatable string does not have the same parameter count as the default locale translation.

Jade Painter

The locale used for selecting the translation to display in the Jade Painter is set on the **Miscellaneous Options** sheet of the Preferences dialog.

If you do not select a locale in the **Base Locale** combo box, the current session locale is used. (For details, see "Maintaining Miscellaneous Options" under "Setting User Preferences", in Chapter 2.)

When the Jade Painter assembles a form that is a subclass from a superschema for display, if the current locale does not exist in the superschema, the form data is taken from the default locale of the superschema.

The status line of the Jade Painter displays the three-letter abbreviated country code for the locale of the form being edited (for example, **USA** or **NZL**).

Loading Locales

New locales defined in a schema file (but that are not in the database) are never added to the database. Existing locales are not modified or deleted by the loading of a schema. Locale attributes defined in a schema file do not override existing locale attributes. If you add, delete, or modify the locales of a schema, a subsequent reload of the schema does not restore the locale structure defined in the schema file.

The default locale defined in a schema file is applied only to new schemas. If you change the default locale of a schema, a subsequent reload of the schema does not restore the default locale to the schema file defined value.

When you first add a schema to a database in the Jade Platform development environment, an initial base locale is created for the current session locale.

When you first load a schema into a database, locales are created as specified in the schema file. If no locale was found to match the current session locale at the end of the load process, one is created as a clone of the schema default locale.

When loading a schema definition file, you must create the new base locales before loading the schema. If you do not do so, the new locales are ignored.

Maintaining Locales using Jade Command Files

When using the **jadloadb** batch Schema Load utility to maintain locales by using command files, you can:

- Use a command file to create and delete locales for a schema.
- Create both base and clone locales.
- Create a base locale as a copy of another base locale.
- Create a clone locale as a clone of a base locale.
- Delete a locale also deletes any clones of that locale. (Deleting a non-existent locale is not considered an error.)
- Change the default locale for a schema.

You cannot delete the schema default locale. At least one base locale must always be present.

For details about maintaining locales or deleting translatable strings or locale formats, see "**commandFile**", under "Loading Schemas in Batch Mode using jadloadb", in the *Schema Load User's Guide*.

Maintaining Forms Management Styles using Jade Command Files

You can change the forms management style of a schema and its subschemas only by using a command file in the **jadloadb** batch Schema Load utility.

For details about changing the forms definition style of an existing schema, see "**commandFile**" under "Loading Schemas in Batch Mode using jadloadb", in the *Schema Load User's Guide*.

Searching for Translatable Strings and Forms at Run Time

When Jade searches for a translation of a translatable string at run time, it searches in the schema where the compiler found the translatable string for the **currentLocale** of the application. If it is not found, the default locale of the schema is selected. If it is found and it is a clone, the locale of which it is a clone is selected.

The selected locale is searched for the translatable string by name and a runtime exception is raised if it is not found or it takes parameters.

When Jade searches for a translation of a form at run time, it searches the schema in which the form is defined. If the **currentLocale** does not exist, the default locale of the schema is selected. If **currentLocale** does exist and it is a clone, the locale of which it is a clone is selected.

The resulting locale is searched for the translation and a runtime exception is raised if it is not found.

Subschema Copies of Subclassed Forms

Forms store their build data persistently in the database. When a form is constructed for display, the build data is updated if it is out of date. The build data consists of details about form and control layout, sizing, and configuration.

When a form is subclassed from a superschema, some of the necessary build data is copied from a translation of the superform. This means that a translation of the form may have build data from a superform that is from a different locale. For example, a locale present in the schema of the form does not exist in the superschema, in which case the translation from the default locale of the superschema is used.

The use of persistent form build data provides a significant performance gain, but it can lead to unexpected results when multiple locales are being used. In particular, if all schemas in the schema hierarchy do not share the same locale structure, subschema subclass forms will have translations built using more than one locale.

When the locale structure of a schema is changed (that is, the default locale changed, a base locale added or deleted, or the **Locale**::**cloneOf** value of a clone changed), in some instances subschema subclass forms will not have their build data reconstructed so that the effect of the locale change is not visible when they are displayed.

When the default locale of a schema is changed, a search is made for form subclasses in that schema. Each located form class is checked for subschema subclasses. If any translation of a subschema subclass form belongs to a locale that does not exist in the superschema, the form data of the translation is rebuilt.

Saving a form in the Jade Painter always causes the form data of all translations to be rebuilt. In addition, if the form has any subclasses (including in subschemas), any translations for the subclasses are rebuilt.

Maintaining Locales for Your Schema

Use the **Locales** command from the Schema menu to specify the locale (language) from which you want the forms and strings of a schema to be presented to you at development time. Each schema can support multiple locales (languages), and forms and strings in the schema can be independently translated for each locale.

At run time, applications automatically use the appropriate form and string translations for the current client locale. For example, if a schema supports the French (France) and English (United Kingdom) locales, a user whose current Windows locale is English (United Kingdom) is presented with the English translation of forms and strings for any applications they run from that schema. Similarly, a user whose current Windows locale is French (France) is presented with the French translation of forms and strings.

As inconsistent results could be returned to the application server when running in Jade thin client mode and there are regional overrides, all overrides on the application server are suppressed when the **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the Jade initialization file is set to **false**. Formatting of locale data is done on the application server, based on the locale of the corresponding presentation client. When running with enhanced locale support, the locale of the presentation client must be present on the machine running the application server.

When you create a new user schema as a subclass of **RootSchema**, one locale that matches the current locale of the application server is created. You can then use locale functionality to define other locales.

When loading a:

- Complete schema definition file, new locales that are in the schema and forms definition files but not in the database are loaded into an existing schema.
- Partial schema definition file, you must create the new base locales before loading the schema. If you do not do so, the new locales are ignored. For more details, see "Loading Locales", earlier in this chapter.

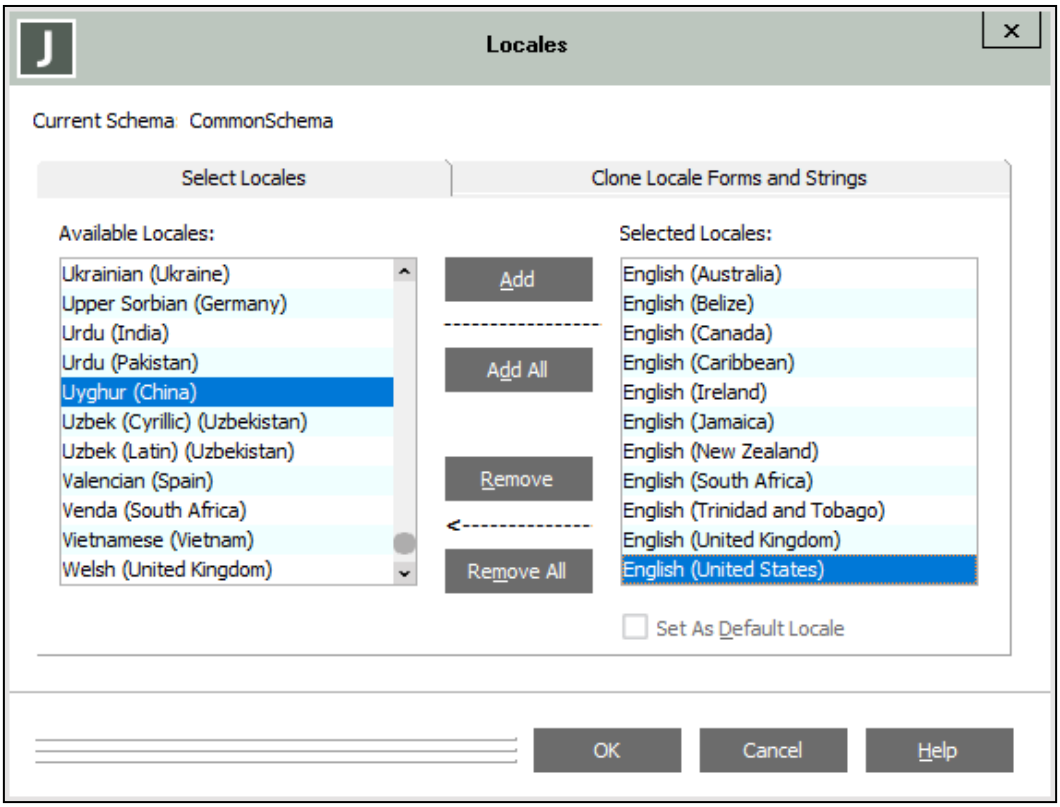
In the Jade Platform development environment, use the:

- **Base Locale** combo box on the **Miscellaneous** sheet of the Preferences dialog to specify the locale from which you want the forms and strings of a schema to be presented to you at development time.
- **Locales** command from the Schema menu to specify the locales that are to be supported by a schema.

To access the Locales dialog

1. In the Schema Browser, select the schema whose locales you want to specify.
2. Select the **Locales** command from the Schema menu.

The Locales dialog, shown in the following image, is then displayed.



Selecting Your Locales

The **Select Locales** sheet is displayed by default, to enable you to first select the locales that are to be supported by the schema.

Adding Locales

Use the **Select Locales** sheet from the Locales dialog to specify locales supported by the current schema.

To add locales

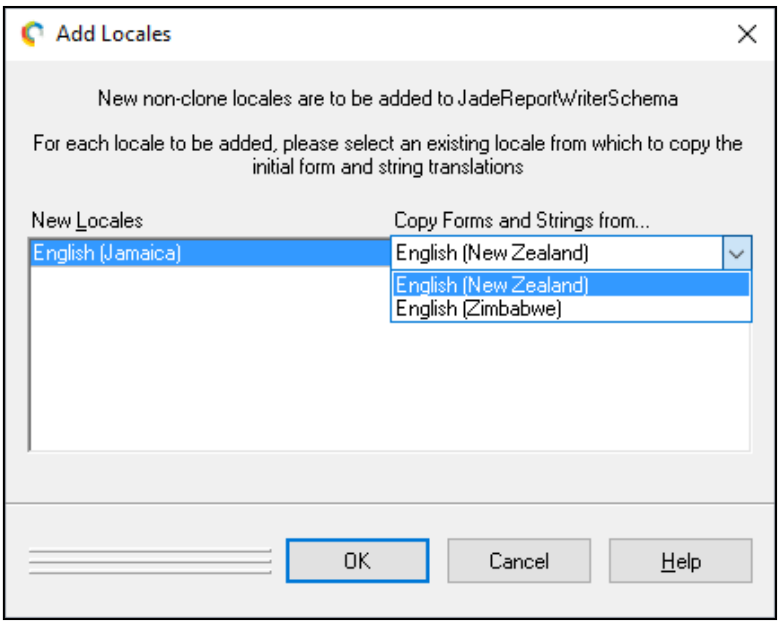
1. In the **Available Locales** list box, select the locale that you want to support.
2. Click the **Add** button, to move the selected locale to the **Selected Locales** list box.

   Repeat this step for each locale that you want to select. Alternatively, to select all locales, click the **Add All** button.
3. Check the **Set As Default Locale** check box if you want to set a locale selected in the **Selected Locales** list box as the default locale for the schema.

You can select only a base (that is, non-clone) locale as the default locale.

4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your changes.

The Add Locales dialog, shown in the following image, is then displayed.



For each locale that you have added, the Add Locales dialog enables you to specify the locale from which the initial form and string translations are copied.

To specify the locale from which to copy initial translations

1. In the **New Locales** list box of the Add Locales dialog, select a locale to be added. The combo box is then displayed.
2. From the **Copy Forms and Strings from…** combo box, select the locale from which to copy the initial translations for the new locale.
3. Repeat steps 1 and 2 of this instruction for each new locale that you added.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The Updating Locales progress dialog is then displayed, followed by the Schema Browser when the initial locale translations have been performed.

What Happens when You Add a Locale

As an example of adding a locale, you might have an **Example** schema that supports two locales and has one form and one translatable string.

The form instances in the **Example** schema are listed in the following table.

Locale	Form Name	Form Caption
English (United Kingdom)	LogonForm	Welcome
French (France)	LogonForm	Bien Venu

The translatable string instances in the **Example** schema are listed in the following table.

Locale	String Name	Definition
English (United Kingdom)	GoodMorningStr	"Good Morning"
French (France)	GoodMorningStr	"Bon Matin"

You then use the Locales dialog to add German (Germany) and Spanish (Spain) locales. When you click the **OK** button, the Add Locales dialog is then displayed, to enable you to specify that the initial translations for German (Germany) are to be taken from English (United Kingdom) and the initial translations for Spanish (Spain) are to be taken from French (France).

When you click the **OK** button, the following table lists the updated form instances in the **Example** schema.

Locale	Form Name	Form Caption
English (United Kingdom)	LogonForm	Welcome
French (France)	LogonForm	Bien Venu
German (Germany)	LogonForm	Welcome
Spanish (Spain)	LogonForm	Bien Venu

The following table lists the updated translatable string instances in the **Example** schema.

Locale	String Name	Definition
English (United Kingdom)	GoodMorningStr	"Good Morning"
French (France)	GoodMorningStr	"Bon Matin"
German (Germany)	GoodMorningStr	"Good Morning"
Spanish (Spain)	GoodMorningStr	"Bon Matin"

The **Example** schema now supports four independent translations of **LogonForm** and **GoodMorningStr**. Any controls or menu items added to or removed from **LogonForm** are added to or removed from all translations of that form.

However, any changes to attributes (for example, captions, size, position, and so on) of a form or its controls or menu items affect only the translation being modified. Any changes to a translatable string affect only the current translation that is being modified.

Having added German and Spanish locales to the **Example** schema, you could then further translate **LogonForm** to produce the form instances listed in the following table.

Locale	Form Name	Form Caption
English (United Kingdom)	LogonForm	Welcome
French (France)	LogonForm	Bien Venu
German (Germany)	LogonForm	Hertzlich Wilkommen
Spanish (Spain)	LogonForm	Bienvenido

The following table lists how you could then further translate **GoodMorningStr**.

Locale	String Name	Definition
English (United Kingdom)	GoodMorningStr	"Good Morning"
French (France)	GoodMorningStr	"Bon Matin"
German (Germany)	GoodMorningStr	"Guten Morgen"
Spanish (Spain)	GoodMorningStr	"Buenos Dias"

Removing Locales

Use the **Select Locales** sheet from the Locales dialog to remove locales from the current schema.

» To remove locales

1. In the **Selected Locales** list box, select the locale that you no longer want to support.
2. Click the **Remove** button to move the locale from the **Selected Locales** list box to the **Available Locales** list box.
3. Repeat steps 1 and 2 for each locale that you want to remove. Alternatively, you can click the **Remove All** button if you want to remove all selected locales except inherited locales.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

You cannot remove a locale if it is being used by active applications. Before a locale is removed, Jade checks to ensure it is not in use. If it is in use, a warning message is displayed and the locale is not removed. When a locale is removed, all form and string translation for the locale are also deleted.

Cloning Locale Forms and Strings

Use the **Clone Locale Forms and Strings** sheet of the Locales dialog to specify that one locale (the clone locale) uses the forms and strings of another locale.

Making a locale the clone of another locale is useful if you want to be able to distinguish between the two locales at run time but you want them to share form and string translations. For example, Windows supports several English locales, including English (United Kingdom), English (Australia), and English (New Zealand).

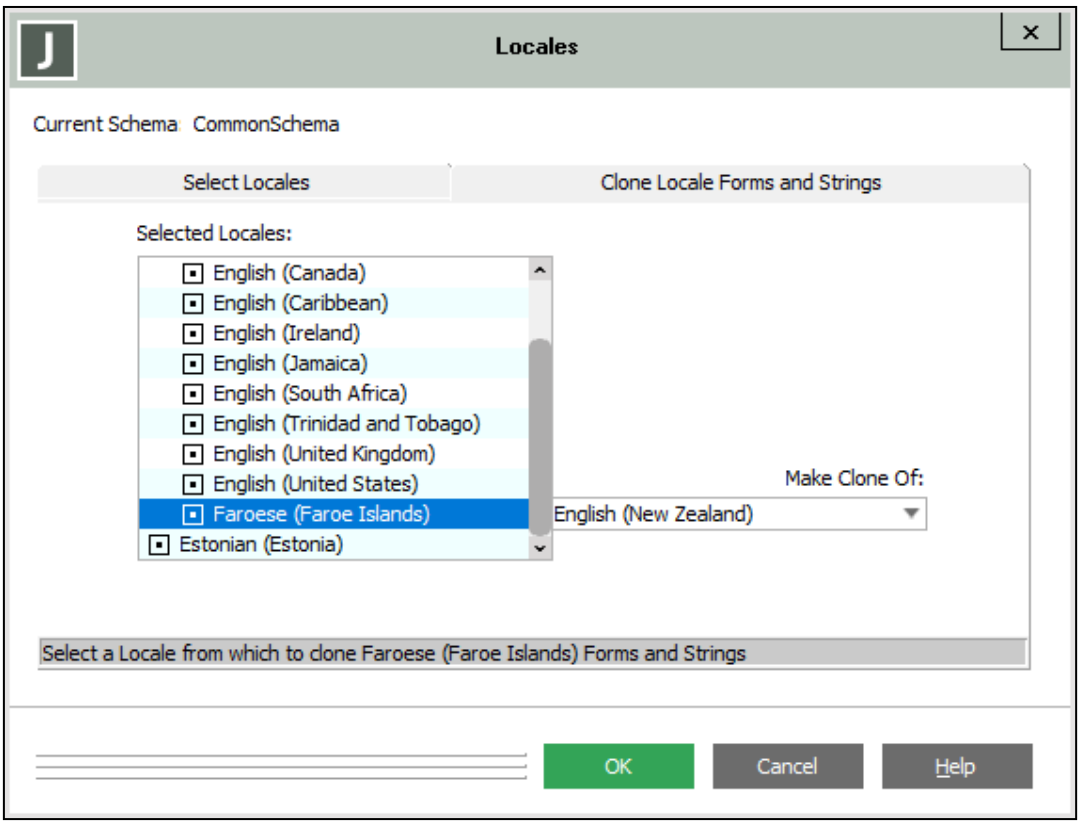
You might want to distinguish between all of these locales at run time in order for your application to customize some behavior based on the country. However, as all of these locales are English, you probably want only one set of English translations of your forms and strings. In this case, you would make two of the locales clones of the third locale. You can then differentiate between all three locales at run time, but you have only one set of form and string translations to maintain.

Before you can create clones, a schema must support at least two locales, one of which must not be inherited.

To specify that a locale is to be a clone of another locale

1. Click the **Clone Locale Forms and Strings** sheet in the Locales dialog.

   The **Clone Locale Forms and Strings** sheet, shown in the following image, is then displayed.



2. In the **Selected Locales** list box, select the locale that you want to make the clone of another locale.
3. In the **Make Clone Of** combo box, select the locale from which the selected locale is to be cloned. For example, if you want to make Austrian German forms and strings clones of the German (Germany) locale, select German (Austria) in the **Selected Locales** list box and then select German (Germany) from the **Make Clone of** combo box. To make Swiss German a clone of German (Germany), do the same for the German (Switzerland) locale.
4. Repeat steps 2 and 3 of this instruction for all locales that you want to make clones.
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Note When a locale is made the clone of another locale, any form and string translations that exist for that locale are then deleted.

When a locale has been selected from which to clone forms and strings, the selected locale is then displayed as a subset of the locale of which it is a clone; as shown in the following image.

To specify that a locale is no longer to be a clone

1. Click the **Clone Locale Forms and Strings** sheet in the Locales dialog.
2. In the **Selected Locales** list box, select the locale that you no longer want to be a clone.
3. In the **Make Clone of** combo box, select **(Not Cloned)**.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Note When a locale is uncloned, its form and string translations are restored, by copying the form and string translations of the locale of which it was a clone.

Adding and Maintaining Formats

Use the **Formats** command from the Schema menu in the Schema Browser to define format strings for use with primitive types in your schema methods.

The **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the Jade initialization file enables handling of standard client and presentation client locale-sensitive formatting of date, time, and numeric values (that is, numbers and currency) to be standardized and the regional overrides set on the presentation client to be forwarded to the application server so that both nodes use a consistent set of locale settings for the application when running in Jade thin client mode.

To access the Format Browser

- Select the **Formats** command from the Schema menu.

The Format Browser is then displayed. Use the Format Browser to define or maintain format strings for use with primitive types in your schema methods.

The Formats menu, accessed from the Format Browser, contains the commands listed in the following table.

Command	Action	For details, see …
Add Short Date Format	Displays the Add Short Date Format dialog	Adding a Short Date Format
Add Long Date Format	Displays the Add Long Date Format dialog	Adding a Long Date Format
Add Time Format	Displays the Add Time Format dialog	Adding a Time Format
Add Numeric Format	Displays the Number Format dialog	Adding a Numeric Format
Add Currency Format	Displays the Add Currency Format dialog	Adding a Currency Format
Change	Displays the Format dialog for the selected format	Maintaining Formats
References	Displays the References window for the selected string	Viewing References to Formats
Remove Format	Deletes the selected format	Removing a Format

For example, if you define a short date format called **MyShortFormat** using the **Add Short Date Format** command, you could use that format in a Jade method, as follows.

```
write "The date today is " & date.userFormat($MyLongDate);
// Outputs The date today is Wednesday, 27 of November, 2002
```

Note When you use a defined format in a Jade method, you must prefix your user format names with a dollar sign (**$**); for example, **userFormat($MyDate)**.

The date would then be formatted in the manner defined in the Add Short Date Format dialog for **MyShortFormat**.

Note When you select a format in the Format Browser, an example of the selected format is displayed in the status line.

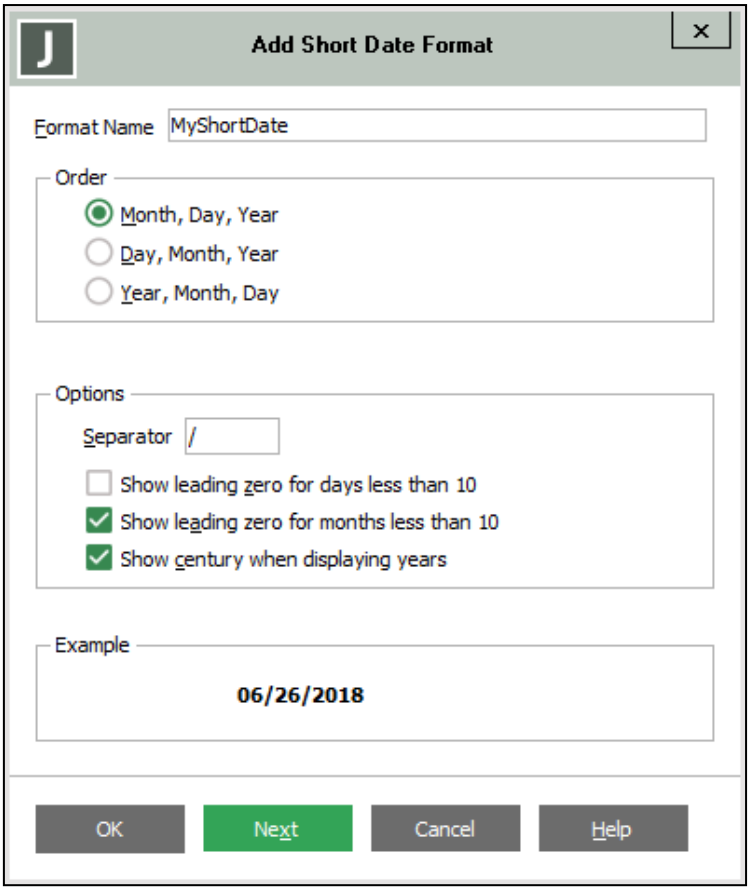
You can expand (open) and collapse (close) format sheet nodes in the Format Browser. For example, if you have a number of short date formats, double-click the **Short Date Formats** sheet to suppress the display of your short date formats.

Adding a Short Date Format

From the Format Browser, select the **Add Short Date Format** command from the Formats menu to add a short date format to the current schema. As you enter your format selections, an example of the date in that format is displayed at the bottom of the dialog.

To add a short date format

1. Select the **Add Short Date Format** command from the Formats menu. The Add Short Date Format dialog, shown in the following example, is then displayed.



2. In the **Format Name** text box, specify the name enter a name for your short date format. You must specify a value in this text box. An exception is raised if the specified name does not start with an uppercase letter or the format name is not unique; that is, your schema already contains a format with that name.
3. In the Order group box, select your required date order. Select the:
   - **Month, Day, Year** option button if you want your short date formatted in month, day, year order; for example, **12.31.02**.
   - **Day, Month, Year** option button, which is selected by default, if you want your short date formatted in day, month, year order; for example, **31.12.02**.
   - **Year, Month, Day** option button if you want your short date formatted in year, month, day order; for example, **02.12.31**.
4. In the **Separator** text box of the Options group box, if you do not want the default stroke character (/) used as the date separator, specify the type of separator that you require for your short date; for example, you can specify a space.
5. Check the **Show leading zero for days less than 10** check box of the Options group box if you want days of the month less than 10 (that is, the first nine days of any month) displayed with leading zeros; for example, **09/12/02**. By default, leading zeros are not displayed.
6. Check the **Show leading zero for months less than 10** check box of the Options group box if you want months of the year less than 10 (that is, the first nine months of any year) displayed with leading zeros; for example, **12/08/02**. By default, leading zeros are not displayed.
7. Check the **Show century when displaying years** check box of the Options group box if you want the century to be displayed in the year; for example, **14/12/2002**. By default, the century is not displayed.
8. Click the **OK** button when you have defined your new short date format. Alternatively, you can click the **Next** button to redisplay the Add Short Date Format dialog so that you can define another short date format, or you can click the **Cancel** button to abandon your selections.

When you click the **OK** button, the Format Browser is then displayed, with the defined short date format included in the Short Date Formats hierarchy and an example of a date in your specified format displayed in the status line.

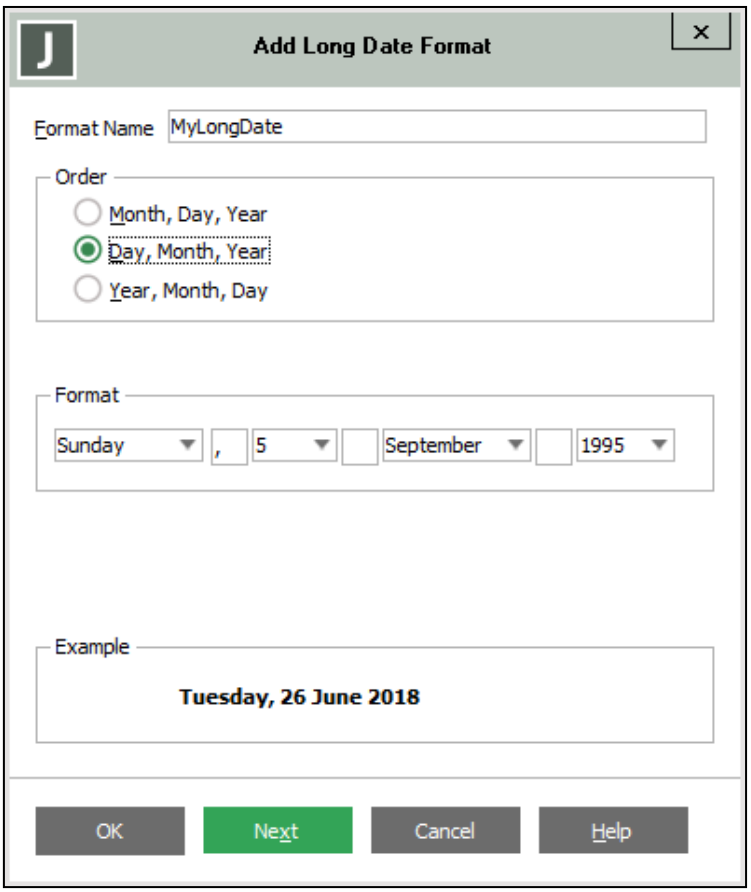
Adding a Long Date Format

From the Format Browser, select the **Add Long Date Format** command from the Formats menu to add a long date format to the current schema. As you enter your format selections, an example of the date in that format is displayed at the bottom of the dialog.

To add a long date format

1. Select the **Add Long Date Format** command from the Formats menu.

   The Add Long Date Format dialog, shown in the following image, is then displayed.



2. In the **Format Name** text box, specify a name for your long date format. You must specify a value in this text box. An exception is raised if the specified name does not start with an uppercase letter or the format name is not unique; that is, your schema already contains a format with that name.

3. In the Order group box, select the date order that you require. Select the:

   - **Month, Day, Year** option button if you want your long date formatted in month, day, year order; for example, **December 14, Saturday 2002**.
   - **Day, Month, Year** option button, which is selected by default, if you want your long date formatted in day, month, year order; for example, **Saturday, 14 December 2002**.
   - **Year, Month, Day** option button if you want your long date formatted in year, month, day order; for example, **2002, December 14 Saturday**.

4. In the combo boxes in the Format group box, select the required displays for your long date format; for example, in the first combo box you can select the day name to be displayed in full or abbreviated (that is, **Saturday** or **Sat**).

In the separator text boxes in the Format group box, specify the separators that you require between each part of your long format; for example, a hyphen (**-**) after the day name and a comma (**,**) between the day, month, and year.

As you define each part of your format, an example of your selected format is displayed beneath the Format group box.

5. Click the **OK** button when you have defined your new short date format. Alternatively, you can click the **Next** button to redisplay the Add Long Date Format dialog so that you can define another long date format or you can click the **Cancel** button to abandon your selections.

When you click the **OK** button, the Format Browser is then displayed, with the defined long date format displayed in the Long Date Formats hierarchy and an example of a date in the defined format displayed in the status line.

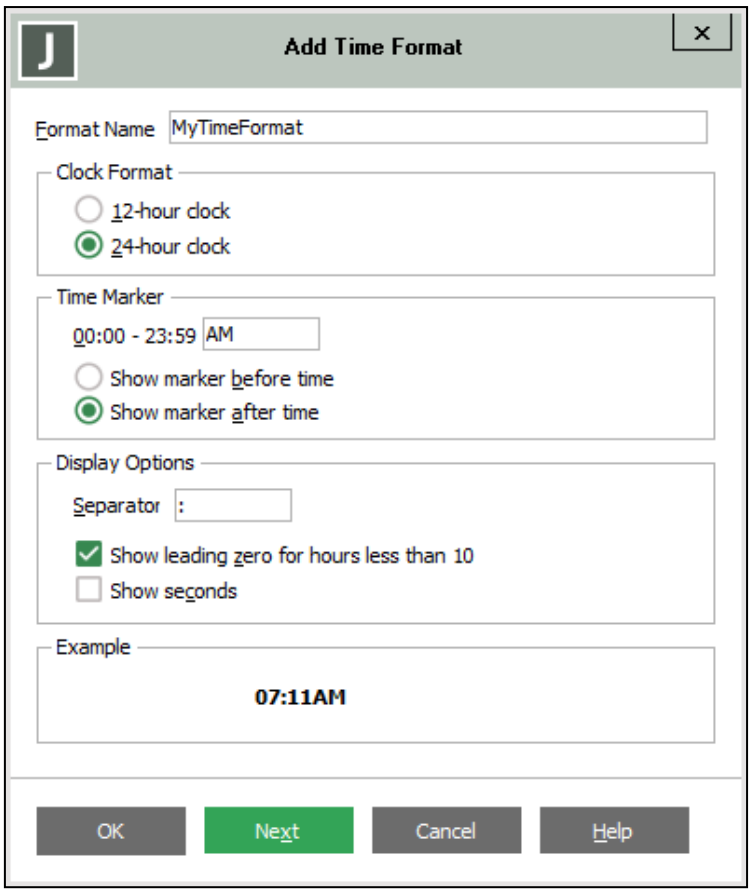
Adding a Time Format

From the Format Browser, select the **Add Time Format** command from the Formats menu to add a time format to the current schema. As you enter your format selections, an example of the time in that format is displayed at the bottom of the dialog.

To add a time format

1. Select the **Add Time Format** command from the Formats menu.

   The Add Time Format dialog, shown in the following image, is then displayed.



2. In the **Format Name** text box, specify a name for your time format. You must specify a value in this text box. An exception is raised if the specified name does not start with an uppercase letter or the format name is not unique; that is, your schema already contains a format with that name.

3. In the Clock Format group box, select the clock format that you require for your date; that is, the 12-hour or the 24-hour clock. By default, the date is formatted in the 24-hour format.

4. If you have selected the default 24-hour clock format option, in the **00:00 - 23:59** text box of the of the Time Marker group box, you can optionally specify words or symbols that you want displayed before or after the time format; for example, **EST** for Eastern Standard Time.

   If you have selected the 12-hour clock format option, in the **00:00 - 11:59** text box of the Time Marker group box you can optionally specify words or symbols that you want displayed in the time format before noon (for example, **A.M.** or **am**) and in the **12:00 - 23:59** text box, you can optionally specify words or symbols that you want displayed in the time format after noon (for example, **P.M.** or **pm**).

5. If you specified a time marker, select the **Show marker before time** option button if you want to display the time marker before the time, if required (for example, **EST 09:46:32**). By default, the marker is displayed after the time; that is, the **Show marker after time** option button is selected (for example, **09:46:32 A.M.**).

6. In the **Separator** text box in the Display Options group box, specify the separator that you want displayed between the hours, minutes, and seconds of your time format; for example, a stroke character (**/**). The default separator is a colon character (**:**).

7. Uncheck the **Show leading zero for hours less than 10** check box of the Display Options group box if you do not want hours less than **10** to be displayed with leading zeros; for example, **9:53:14**. By default, leading zeros are displayed.

8. Uncheck the **Show seconds** check box of the Display Options group box if you do not want seconds to be displayed in your time format; for example, **09:15:58 am**. By default, seconds are displayed.

9. Click the **OK** button when you have defined your new time format. Alternatively, you can click the **Next** button to redisplay the Add Time Format dialog so that you can define another time format or you can click the **Cancel** button to abandon your selections.

When you click the **OK** button, the Format Browser is then displayed, with the defined time format included in the Time Formats hierarchy and an example of the time in the format that you defined displayed in the status line.

Adding a Numeric Format

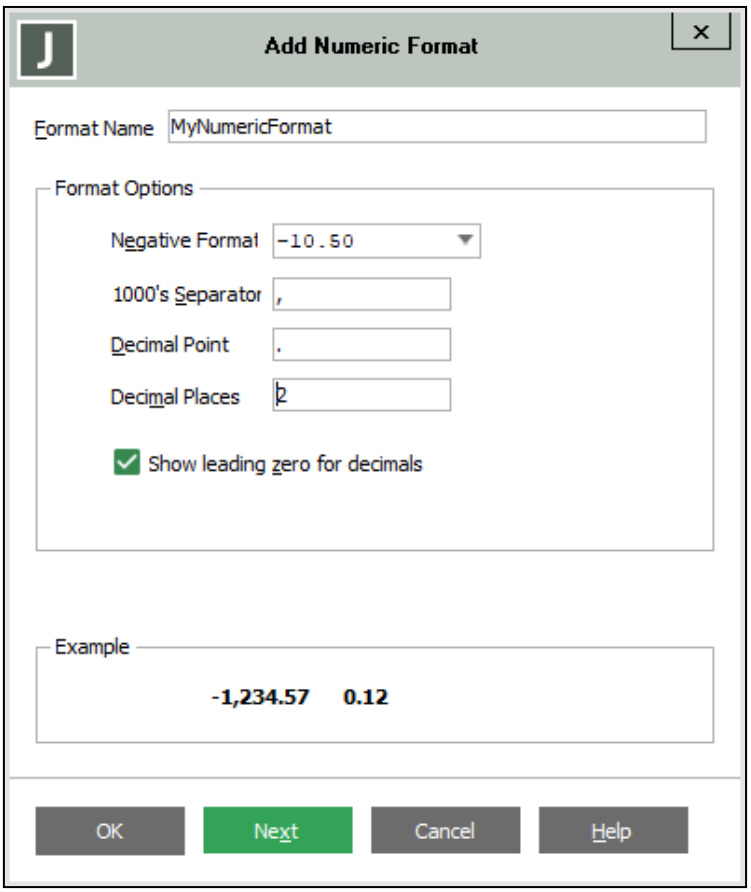
From the Format Browser, select the **Add Numeric Format** command from the Formats menu to add a numeric format to the current schema.

As you enter your format selections, an example of a number in that format is displayed at the bottom of the dialog.

To add a numeric format

1. Select the **Add Numeric Format** command from the Formats menu.

   The Add Numeric Format dialog, shown in the following image, is then displayed.



2. In the **Format Name** text box, specify a name for your numeric format. You must specify a value in this text box. An exception is raised if the specified name does not start with an uppercase letter or the format name is not unique; that is, your schema already contains a format with that name.

3. In the **Negative Format** combo box in the Format Options group box, select the required format that you want to use for negative currency values. The default value is parentheses enclosing the number; for example, **(10.50)**.

4. In the **1000's Separator** text box in the Format Options group box, specify the character you that want to use to separate thousands. The default value is a comma character (**,**).

5. In the **Decimal Point** text box in the Format Options group box, specify the character that you want to use to separate decimal digits from whole numbers. The default value is a period character (**.**).

6. In the **Decimal Places** text box in the Format Options group box, specify the number of digits that you want displayed to the right of the decimal separator. The default value is two decimal places; for example, **1,234.56**.

7. If you do not want to show a leading zero in front of numbers less than 1 (for example, **.75**), uncheck the **Show leading zero for decimals** check box in the Format Options group box. Alternatively, if you want to show leading zeros for decimals (for example, **0.75**) and the check box is unchecked, check the box.

The International - Number Format dialog from your Windows Control Panel determines the default setting of this check box; that is, if your workstation has the Windows **Leading Zero** option button selected, the **Show leading zero for decimals** check box is checked by default.

8. Click the **OK** button when you have defined your new numeric format. Alternatively, you can click the **Next** button to redisplay the Add Numeric Format dialog so that you can define another numeric format or you can click the **Cancel** button to abandon your selections.

When you click the **OK** button, the Format Browser is then displayed, with the defined numeric format included in the Numeric Formats hierarchy and an example of a number in the format that you defined displayed in the status line.

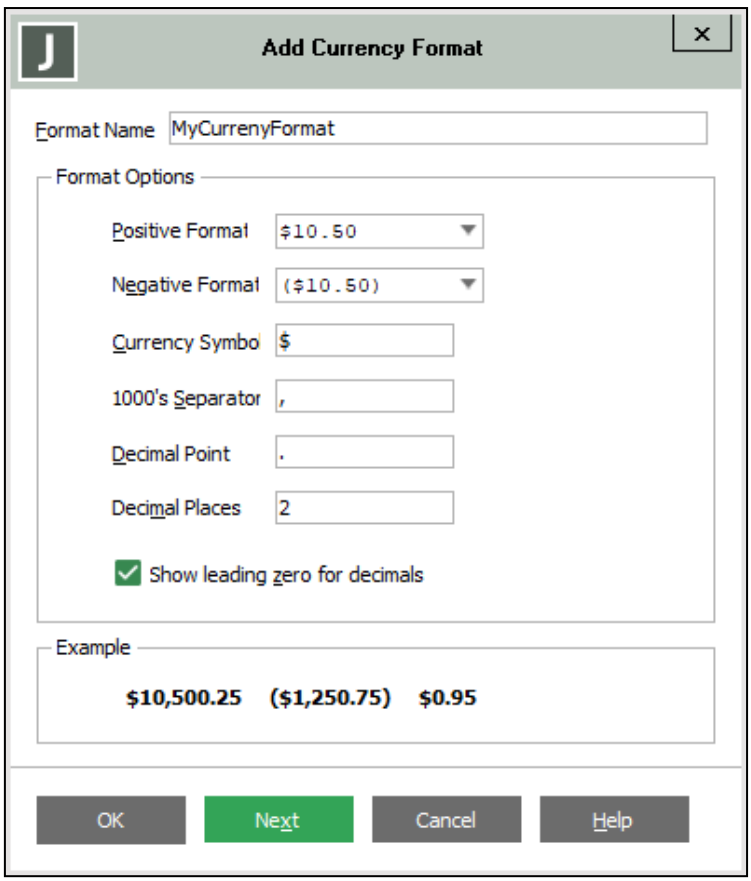
Adding a Currency Format

From the Format Browser, select the **Add Currency Format** command from the Formats menu to add a currency format to the current schema.

As you enter your format selections, an example of the currency in that format is displayed at the bottom of the dialog.

» To add a currency format

1. Select the **Add Currency Format** command from the Formats menu. The Add Currency Format dialog, shown in the following image, is then displayed.



2. In the **Format Name** text box, specify a name for your currency format. You must enter a value in this text box. An exception is raised if the specified name does not start with an uppercase letter or the format name is not unique; that is, your schema already contains a format with that name.

3. In the **Positive Format** combo box in the Format Options group box, select the required format you want to use for positive currency values. The default value is the positive format of your current Windows locale; for example, **$10.50**.

4. In the **Negative Format** combo box in the Format Options group box, select the required format you want to use for negative currency values. The default value is the negative format of your current Windows locale; for example, **($ 10.50)**.

5. In the **Currency Symbol** text box in the Format Options group box, specify the currency symbol that you want to use for the base locale of your current schema; for example, **$**. The default value is the currency symbol of your current Windows locale; for example, **$** if your current locale is United States.

6. In the **1000's Separator** text box in the Format Options group box, specify the character you want to use to separate thousands. The default value is a comma character (**,**).

7. In the **Decimal Point** text box in the in the Format Options group box, specify the character that you want to use to separate decimal digits from whole numbers. The default value is a period character (**.**).

8. In the **Decimal Places** text box in the of the Format Options group box, specify the number of digits you want to display to the right of the decimal separator. The default value is two decimal places; for example, **1,234.56**.

9. If you do not want to show a leading zero in front of numbers less than 1 (for example, **.75**), uncheck the **Show leading zero for decimals** check box in the Format Options group box. Alternatively, if you want to show leading zeros for decimals (for example, **0.75**) and the check box is unchecked, check the box.

   The International - Number Format dialog from your Windows Control Panel determines the default setting of this check box; that is, if your workstation has the Windows **Leading Zero** option button selected, the **Show leading zero for decimals** check box is checked by default.

10. Click the **OK** button when you have defined your new currency format. Alternatively, you can click the **Next** button to redisplay the Add Currency Format dialog so that you can enter another currency format or you can click the **Cancel** button to abandon your selections.

When you click the **OK** button, the Format Browser is then displayed, with the defined currency format included in the Currency Formats hierarchy and an example of currency in the format that you defined displayed in the status line.

Maintaining Formats

From the Format Browser, select the **Change** command from the Formats menu to edit an existing format. As you edit your format, an example of that format is displayed at the bottom of the dialog.

» To edit a format

1. In the Format Browser, select the format that you want to edit.

2. Select the **Change** command from the Formats menu.

   The Edit Format dialog for the type of format that you selected is then displayed, to enable you to amend your defined values to meet your requirements. For example, if you selected a short date format called **MyShortDateFormat** from the Format Browser, the Edit Short Date Format dialog is then displayed.

   The values that you defined for that format are then displayed in the appropriate combo boxes, text boxes, or option buttons.

3. Amend the displayed values, as required.

Note You can change any value other than the format name. If you want to alter the name of an existing format, you must add a new format with the appropriate name and values, and then delete the wrongly named format, if appropriate.

4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

The Format Browser is then redisplayed. The format that you edited is highlighted and an example of the amended format is then displayed in the status line.

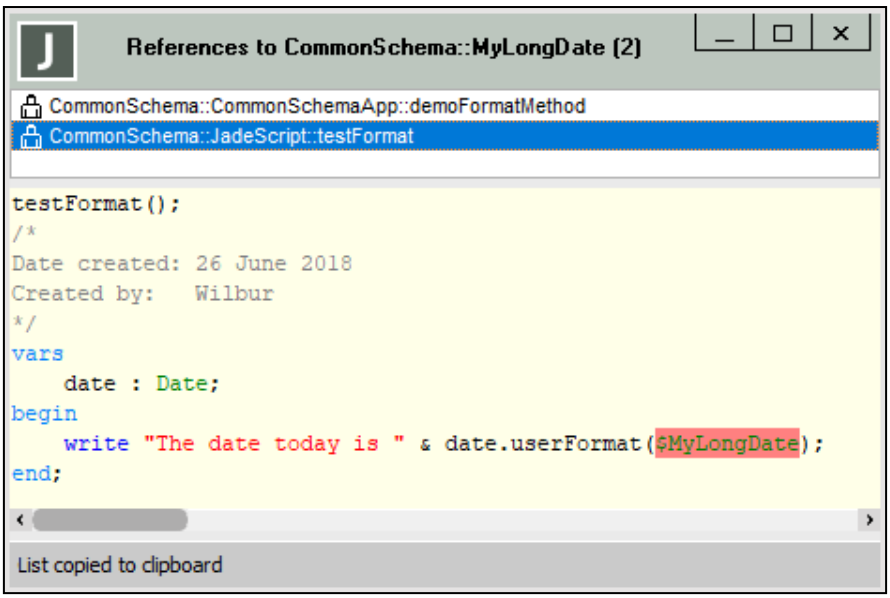
Viewing References to Formats

The format References window enables you to view all references to the format selected in the Format Browser.

To view references of the selected format

1. Select the **References** command in the Formats menu from the Format Browser. (This command is disabled if the selected format is not used in any method.)

   The References window for the selected format, shown in the following image, is then displayed.



   This window lists all classes and their methods that reference the selected format, and enables you to view specific references to that method.

2. In the Methods List window, select the method whose reference to the selected format you want to view.

The selected method is then displayed in the editor pane, with the format highlighted. You can maintain and compile methods displayed in the References window, if required. (For details, see "Compiling Methods", in Chapter 4.)

Removing a Format

From the Format Browser, select the **Remove Format** command from the Formats menu to delete an existing format.

To remove a format

1. In the Format Browser, select the format that you want to delete.
2. Select the **Remove Format** command from the Formats menu.

The selected format is then deleted and the Format Browser is updated so that the format has been removed from its hierarchy.

Translating Strings

You can access the String Browser that enables you to translate strings, by selecting:

- The **Strings** command from the Schema menu, to maintain strings for the locales supported by the schema.
- The **Jade Translator** program icon from your Jade program folder, to run the standalone Jade Translator utility when the value of the **Schema** class **formsManagement** property is set to the default value of **FormsMngmt_Multi_Multi (0)**. For more details, see "Jade Translator Utility", later in this chapter.

The **Add** and **Remove** buttons are not displayed on the String Browser when you access it from the **Jade Translator** program.

Notes Only one String Browser for the current schema can be open at any time. If a String Browser is already open for that schema, it is brought to the top when you select the **Strings** command from the Schema menu. You can have concurrent open String Browsers for different schemas in a development environment session.

For details about creating translatable strings in HTML document source, see "JADE_TAG Tag Notes", in Chapter 12.

When you delete a translatable string, you are warned if the translatable string that you want to delete is used by other translatable strings. For details about removing schema elements, see "Removing a Schema Element", in Chapter 3. For details about locating translatable strings that reference a specific translatable string, see "Searching for Strings that Reference the Current String", later in this chapter. See also "Compiling Translatable Strings", earlier in this chapter.

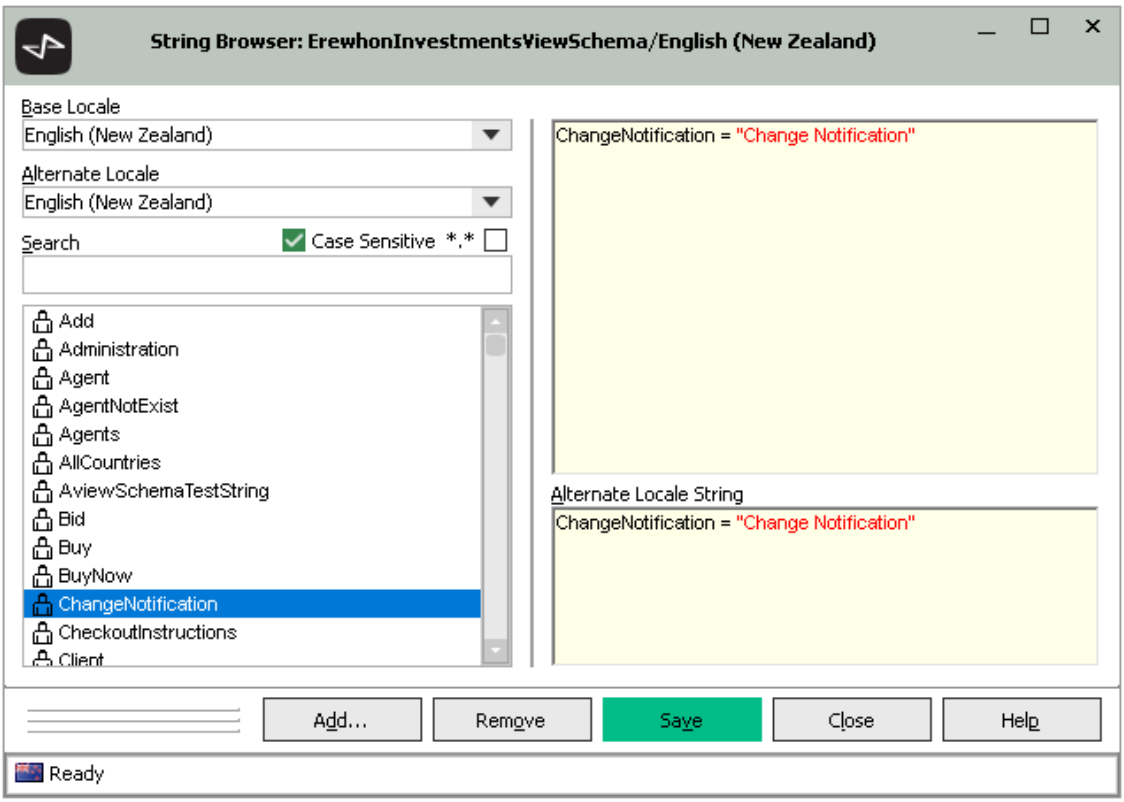
You can view the usages of a selected string in your Jade applications and add, edit, and remove strings for each locale.

The String Browser enables you to browse strings for your supported locales.

To access the String Browser from the Jade Platform development environment

- Select the **Strings** command from the Schema menu.

The String Browser for the current base locale is then displayed, as shown in the following image.



Use the **Preferences** command from the browser Options menu to set the base locale in the Jade Platform development environment or use the **Base Locale** combo box in the top left corner of the String Browser. You can change the translatable string value displayed in the pane at the upper left of the String Browser.

Using the String Browser

» To browse a string for a selected locale

1. Select the appropriate locale from the **Base Locale** combo box.
2. To maintain an existing string, select the required string from the list. The selected string is then displayed in the editor pane at the right of the String Browser, to enable you to view or modify it. Alternatively, specify the name of the string in the **Search** text box. The value specified in this text box acts as the starting key to locate an available string in the current schema.
3. If you want to perform a case-insensitive search for a translatable string, uncheck the **Case Sensitive** check box at the upper right of the **Search** combo box. This check box affects a standard prefix or a wildcard search.

   Notes For a standard prefix search, a non-case-sensitive search is not as efficient as a case-sensitive search because the existing (case-sensitive) translatable string dictionary must be searched in a linear fashion from the starting character prefix to locate matching strings. As a wildcard search must search the entire dictionary, efficiency is not affected.

   The translatable list box displays the translatable strings in case-sensitive order, regardless of the setting of the check box.

4. If you want to perform a wildcard search, check the ***.*** check box at the upper right of the **Search** text box and then press Enter. All translatable strings in the current schema that contain the text specified in the **Search** text box in its name are then displayed in the list box of translatable strings at the lower left of the String Browser.

   By default, the search is case-sensitive.

5. To display the value of the same translatable string in another locale, select the required locale from the **Alternate Locale** combo box. The selected string for the alternate locale is then displayed in the editor pane at the lower right of the String Browser.

Note To modify the value of a translatable string an alternate locale, you must select that base locale in the **Base Locale** combo box so that its definition and value are displayed in the updateable pane at the upper right of the String Browser.

The String Browser contains buttons that enable you to perform the actions listed in the following table.

Button	Action
Add	Displays the Add String dialog
Remove	Deletes the selected string from all locales, after prompting you to confirm that the selected string is to be deleted
Save	Saves changes made in the editor pane of the String Browser to the string
Close	Closes the String Browser
Help	Accesses the online help topic that describes using the String Browser

The **Add** and **Remove** buttons are not displayed on the String Browser when you access it from the **Jade Translator** program.

For details about translatable strings, see "Adding Strings", later in this chapter. See also "Removing a Schema Element", in Chapter 3 and "Running an Application from the String Browser", later in this chapter.

In addition, when the String Browser has focus, you can use the Search menu commands listed in the following table to perform search actions.

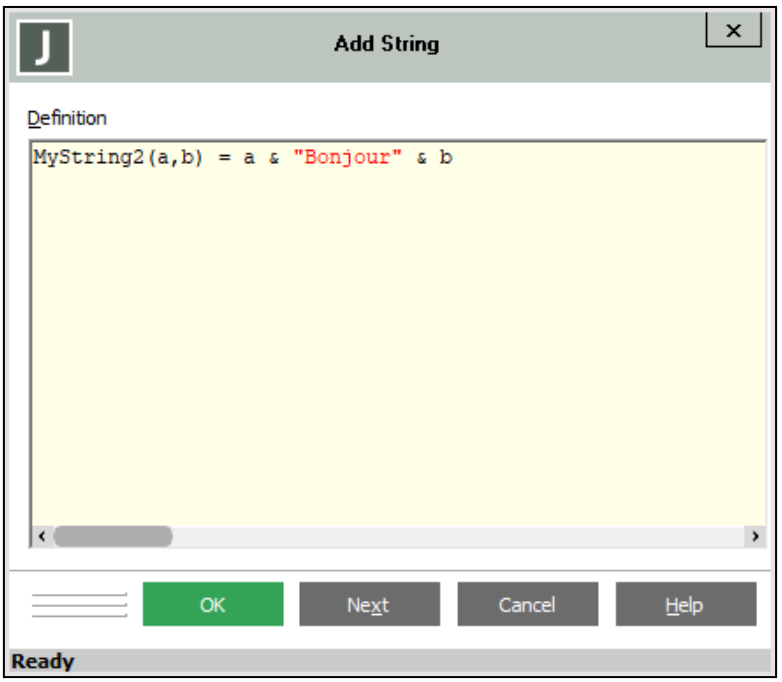
Command	Displays the…	For details, see …
Find Text	Search Translatable Strings dialog	Searching for Text in Existing Strings
All References	References Browser	Viewing References to a String
String References	String Usages window	Searching for Strings that Reference the Current String

Adding Strings

To add a string to the current schema

1. Click the **Add** button in the String Browser.

   The Add String dialog, shown in the following image, is then displayed.



   This form is also displayed when you select a property in the Translatable Property Browser in the Jade Painter. (For details, see "Using the Translatable Property Browser", in Chapter 5.)

2. In the Definition editor pane, specify the definition of the string.

Notes The string name must begin with an uppercase character.

You can search for an existing string, to enable you to locate an existing string instead of having to define another string which may be the same or very similar to an existing string in the current schema (for example, when you use strings for error messages or form labels in an application). For details, see "Searching for Text in Existing Strings", later in this chapter. See also "Searching for Strings that Reference the Current String", later in this chapter.

The string formats are:

```
string-name = "string-definition"

string-name(a,b) = a & "string-definition" & b
```

The following examples are strings defined in these formats.

```
MyString1 = "Bonjour"

MyString2(a,b) = a & "Bonjour" & b
```

Tip You can search for text in existing strings in the schema, and then copy text from a similar string that is located into the editor pane of the Add String dialog. (For details, see "Searching for Text in Existing Strings", later in this chapter.)

3. When you have defined your string, perform one of the following actions.
   - Click the **OK** button to return to the String Browser.
   - Click the **Next** button, to display an empty editor pane to enable you to define your next string.
   - Click the **Cancel** button, to abandon your string and return to the String Browser.

When a string is added, it is added to all locales supported by the schema. However, when a string has been added, it can be translated independently for each locale.

Searching for Text in Existing Strings

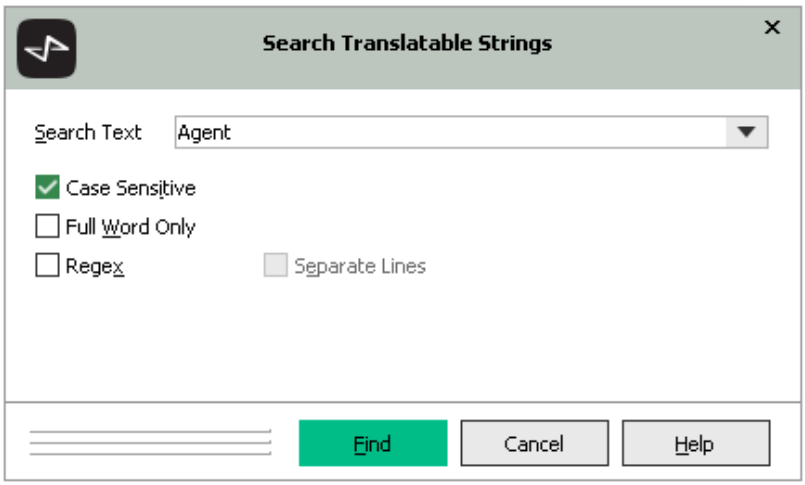
If you use strings for error messages or form labels in an application, your schema may contain a number of strings.

The String Browser enables you to search for specified text within strings in the same schema so that you do not have to define a new string when there may be an existing or similar string in the schema that you can use.

To search for text in a string

1. Select the **Find Text** command from the Search menu.

   The Search Translatable Strings dialog, shown in the following image, is then displayed.



2. In the **Search Text** combo box, specify the text that you want to locate in another string in the current schema.
3. If you want the exact match by case (where uppercase or lowercase is significant), check the **Case Sensitive** check box. A search is then performed in all strings in the current schema for text with the same capitalization as the text in the **Search Text** combo box. By default, searching is case-insensitive; that is, this check box is unchecked.
4. If you want to locate only a complete word, check the **Full Word Only** check box. A search is then performed only for full words in strings that match your specified text. By default, *any* text that matches the specified text is located; that is, this check box is unchecked.

5. If you want the search text to be interpreted as a regular expression pattern, check the **Regex** check box; for example, enter **[A-Za-z]+\d{4}** to search for a word followed by a four-digit number. By default, the **Regex** check box is unchecked. If you check the **Regex** check box, the **Separate Lines** check box is enabled.

   The following list shows examples of regular expressions that you can use in a Regex search. To search for:

   - At least a 4- or 5-digit number assignment ignoring whitespace; for example, an error code

     ```
     \s*:=\s*\d{4,5}
     ```

   - Transaction methods

     ```
     (abortTransaction|beginTransaction|commitTransaction)
     ```

   - **unloadForm** in an epilog

     ```
     epilog[\s\S]*unloadForm
     ```

   - A variable declaration named **userName**

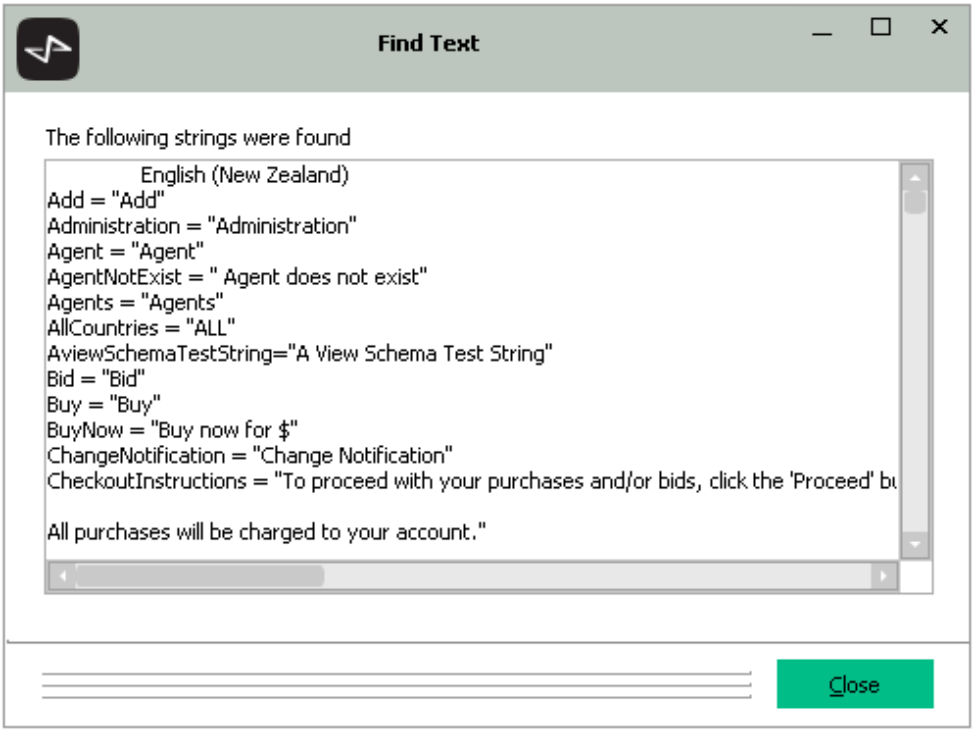
     ```
     vars[\s\S]*\buserName\b[\s\S]*begin
     ```

6. If you want to treat the search text as if it is broken up into separate lines, check the **Separate Lines** check box; for example, if you want to search for a specified string at the start or end of a line, check this check box. When this check box is checked, the search breaks the text into individual lines at each **CrLf** end-of-line sequence. By default, the **Separate Lines** check box is unchecked.

7. Click the **Find** button to initiate the search for the specified string in all strings in the current schema. Alternatively, click the **Cancel** button to abandon your search criteria.

If no occurrences of your specified text are located in any string in the schema, a message dialog is displayed, advising you that the search text was not found.

If occurrences of your specified text are located, the Find Text dialog, like that shown in the following example, is then displayed.



The Find Text dialog allows you only to view located strings that match your search criteria. Modify an existing string by making the changes that you require in the editor pane of the String Browser itself. (Alternatively, to delete an existing string, select the string in the list at the left of the String Browser and then click the **Delete** button.)

When you have viewed the located strings, click the **Close** button to exit from the Find Text dialog and return focus to the String Browser.

Tip If no located strings meet your requirements but a similar one has been located, you can select that string or text within a located string and then copy it into the editor pane of the Add String dialog. (For details, see "Adding Strings", earlier in this chapter.)

Searching for Strings that Reference the Current String

» To search for any strings in the current schema that reference a selected string

- Select the **String References** command from the Search menu.

The current schema is then searched for strings that reference the string selected in the list at the left of the String Browser.

If other strings that reference the selected string are located in the current schema, a message box is then displayed. A message box is displayed, advising you if no other strings reference the selected string.

When you have viewed the existing strings, click the **OK** button to close the message box and return focus to the String Browser.

Viewing References to a String

The References Browser enables you to view all references to the string selected in the String Browser.

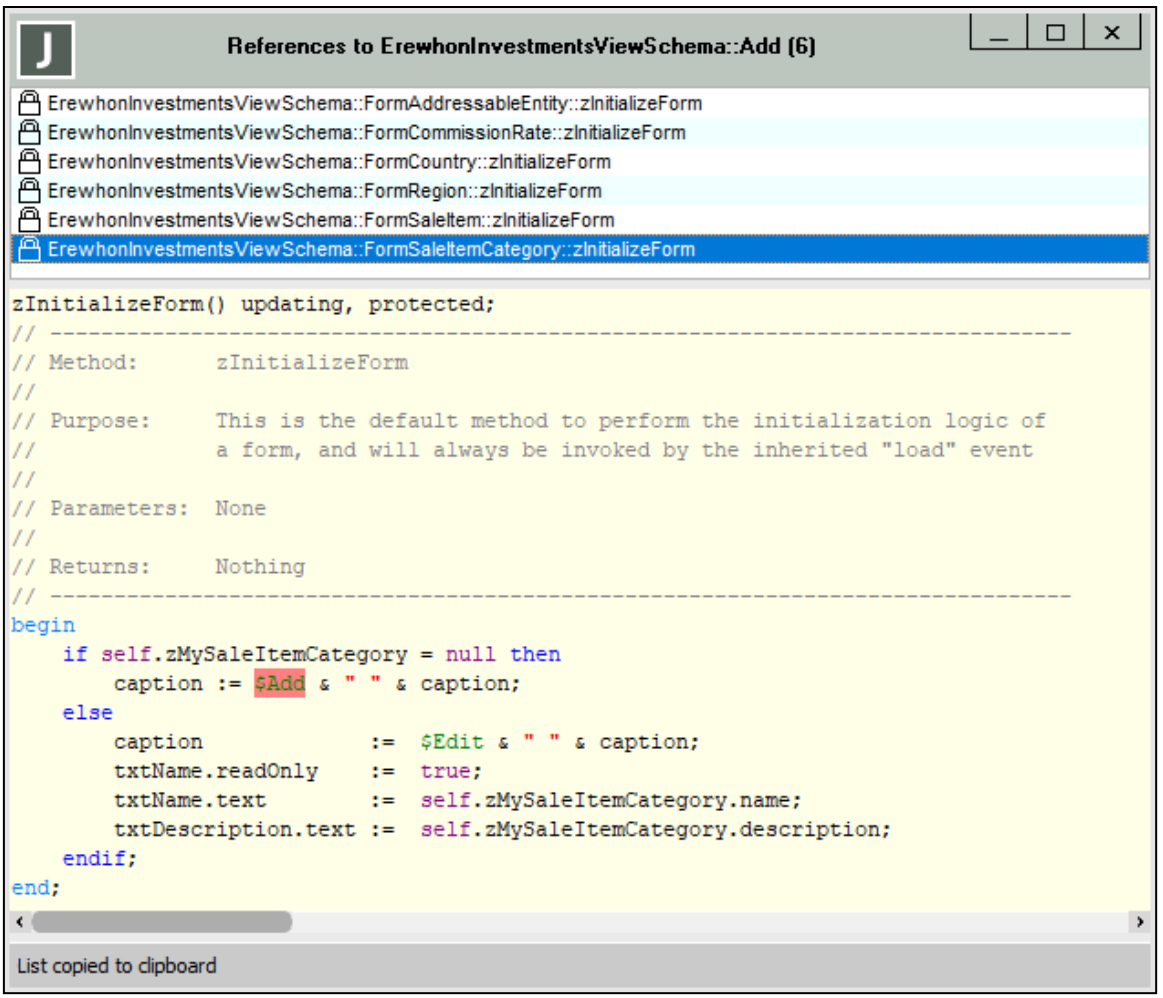
To view all references to the selected string

1. Select the **All References** command from the Search menu.

   If there are no references to the selected string in the current schema, the following message is displayed in the status line of the String Browser and in a warning message box.

   ```
   There are no references to <name-of-your-selected-string>
   ```

   If references to your selected string are located in the current schema, the References Browser for the selected string is then displayed, as shown in the following image.



2. In the Methods List at the top of the window, select the method whose reference to the selected string you want to view.

The selected method is then displayed in the editor pane, with the string highlighted. You can maintain and compile methods displayed in the References window, if required.

Running an Application from the String Browser

The String Browser enables you to run the current application for which you are translating strings.

» **To run an application, perform one of the following actions.**

- Select the **Run Application** command from the File menu.
- Press Ctrl+R.

The Run Application dialog is then displayed.

For details about running an application, see "Running an Application from the Jade Platform Development Environment", in Chapter 1 of the *Runtime Application Guide*.

Applies to Version: 2020.0.01 and higher

Handling Translatable Strings on Message Box Button Captions

The **Application** class **msgBox** method uses translatable strings for the button captions if they are available for non-English translations.

The construction of a message box is as follows.

- If an application form skin is not in use and no message box translated captions are available (or the current locale is English), the standard Windows message box is displayed.

  If an application form skin is in use or message box translated captions are available, a custom message box is displayed. If no message box translated captions are available and the locale is not English, the button captions from the resources in the Windows **user32.dll** are used. This means the captions will be displayed using the language that is currently installed on the user's workstation. If a button translation is available, the text of the translation is used. The custom message box is built so that its size does not exceed the displayable screen area. If the text displayed exceeds the displayable area, a vertical scroll bar is displayed so that the entire message can be accessed.

Message Box Translatable String Handling for Button Captions

When an application is initiated or the presentation client current locale is changed and the locale is not English, the following translatable strings are read, if available, and sent to the presentation client.

- MsgBox_Caption_OK
- MsgBox_Caption_Cancel
- MsgBox_Caption_Abort
- MsgBox_Caption_Retry
- MsgBox_Caption_Ignore
- MsgBox_Caption_Yes
- MsgBox_Caption_No
- MsgBox_Caption_TryAgain
- MsgBox_Caption_Continue

Note It is not necessary to define these translatable strings unless a message box needs to be shown in a locale other than the language installed on the user's workstation.

Programmatically Maintaining Translatable Strings

Jade provides methods that enable you to write your own extract and load routines for translatable strings and to add and modify translatable strings in a running deployed system.

An application can add translatable strings to a schema and change existing translatable strings. For details about these properties, which are summarized in the following table, see Volume 2 of the *Encyclopaedia of Classes*.

Class	Property	Contains a reference to the …
TranslatableString	formBuildDataRefs	Set of forms that contain the translatable string
TranslatableString	locale	Locale of the translatable string

The **TranslatableString** class inherits the properties summarized in the following table from its undocumented metaschema superclass.

Class	Property	Contains…
Constant	constantRefs	A reference to a set of translatable strings that use (reference) the translatable string
Constant	constantUsages	A reference to a collection of translatable strings that are used by (embedded by) the translatable string

For details about these methods, which are summarized in the following table, see Volume 2 of the *Encyclopaedia of Classes*.

Class	Method	Description
Schema	addCompileTranslatableString	Adds a translatable string to all base locales of the receiving schema.
Schema	getBaseLocalesLocal	Populates the specified collection with the base locales defined in the receiving schema.
Schema	getLocaleLocal	Returns a reference to the locale object with the specified name in the receiving schema. Note that this method returns both base and clone locales.
Script	getSource	Method that returns a string containing the current source of the translatable string.
Locale	getTranslatableStringLocal	Returns the specified translatable string object from the receiving locale. Note that if the receiver is a clone locale, this method always returns null.
Locale	getTranslatableStrings	Returns a reference to the translatable strings of the receiving locale. If the receiver is a clone, the collection is that of the associated base locale. The collection is in name order.

Class	Method	Description
Locale	getTranslatableStringsByNum	Returns a reference to the translatable strings of the receiving locale. If the receiver is a clone, the collection is that of the associated base locale. The collection is in number order.
Locale	isClone	Specifies whether the locale is a clone of another locale.
TranslatableString	updateCompile	Updates the existing translatable string.

See also "Adding a New Translatable String", "Updating an Existing Translatable String", "Extract Translatable Strings Method Example", and "Load Translatable Strings Method Example", in the following subsections.

Adding a New Translatable String

Use the **Schema** class **addCompileTranslatableString** method to add a translatable string to all base locales of the receiving schema. If the schema is versioned, the receiver must be the current version.

In the first parameter (**source**), specify the source for the translatable string. This must be in the standard Jade format, as follows.

```
string-name optional-parameter-list = translatable-string-expression
```

The optional parameter list contains one or more parameter name identifiers, separated by commas and enclosed in parentheses.

The translatable string expression consists of string literals, global constant names, and translatable string identifiers concatenated with the ampersand (**&**) operator. You can use white space characters (that is, space, tab, and CrLf) between tokens, as these are ignored. You can also include line comments (*//*) and stream comments (***/* */***).

Note If the source includes a translatable string identifier, the embedded string must also exist.

The process must be in transaction state before the **addCompileTranslatableString** method is called. If the method locates an error (for example, the name of the translatable string is already in use), the method aborts the transaction before returning. You can call the method multiple times in the same transaction.

The caller is responsible for committing successful updates.

This method returns **false** if the addition was successful and it returns **true** if an error was located and the transaction was aborted. The error details are returned in the second, third, and fourth parameters of the method (that is, **errorCode**, **errorOffset**, and **errorLength**, respectively). The token (if any) in the source string where the error was detected is found at the value of the **errorOffset** parameter for the value of the **errorLength** parameter.

You can use the **Process** class **getErrorText** method to obtain the description of the value of the **errorCode** parameter.

Updating an Existing Translatable String

Use the **TranslatableString** class **updateCompile** method to update an existing translatable string. If the translatable string is versioned, the receiver must be the current version.

Note Updates to the current version of a versioned translatable string are *not* carried forward to its latest version.

In the first parameter (**source**), specify the new source for the translatable string. This must be in the standard Jade format. For details about this format, see "Adding a New Translatable String", in the previous section.

The process must be in transaction state before this **updateCompile** method is called. If the method locates an error (for example, the name does not match), the method aborts the transaction before returning. You can call the method multiple times in the same transaction. The caller is responsible for committing successful updates.

This method returns **false** if the update was successful and it returns **true** if an error was located and the transaction was aborted. The error details are returned in the second, third, and fourth parameters of the method (that is, **errorCode**, **errorOffset**, and **errorLength**, respectively). The token (if any) in the source string where the error was detected is found at the value of the **errorOffset** parameter for the value of the **errorLength** parameter.

You can use the **Process** class **getErrorText** method to obtain the description of the value of the **errorCode** parameter.

Notes You cannot change the name of an existing translatable string.

If you want to change the number of parameters in a translatable string, you must first change the translatable string in the schema default locale, or the update will fail with exception 6331 (*Parameter count does not match default locale translation*).

Extract Translatable Strings Method Example

The following method example extracts translatable strings.

```
extractTStrings();
constants
    SchemaName       : String = "TestTstrings";
    LocaleName       : String = "5129";
    ExtractDirectory : String = "C:\Temp";
vars
    scm      : Schema;
    loc      : Locale;
    tstr     : TranslatableString;
    fylename : String;
    fyle     : File;
    count    : Integer;
    now      : TimeStamp;
    delim    : String;
    tsList   : ConstantNDict;
    const    : Constant;
    src      : String;
    chr      : Character;
begin
    scm := rootSchema.getSchema(SchemaName);
    if scm = null then
        write "**Unknown schema =>" & SchemaName;
        return;
    endif;
    loc := scm.getLocaleLocal(LocaleName);
    if loc = null then
        write "**Unknown locale =>" & scm.name & "::" & LocaleName;
        return;
    endif;
    if loc.isClone() then
        write "**Locale is a clone of =>" & scm.name & "::" & loc.name;
        return;
    endif;
    fylename := ExtractDirectory & "/" & scm.name & "-" & loc.name & ".txt";
```

```
    create fyle transient;
    fyle.kind := File.Kind_ANSI;
    fyle.mode := File.Mode_Output;
    fyle.fileName := fylename;
    if not fyle.tryOpen() then
        write "**Could not open file =>" & fylename;
        return;
    endif;
    fyle.writeLine("---Schema=" & scm.name);
    fyle.writeLine("---Locale=" & loc.name & "=" & loc.makeLocaleName());
    fyle.writeLine("---Extracted=" & now.display());
    delim := "-".makeString(60) & CrLf;
    fyle.writeString(delim);
    count := 0;
    foreach const in loc.getTranslatableStrings() do
        count := count + 1;
        tstr := const.TranslatableString;
        src := tstr.getSource().trimBlanks();
        // remove all trailing whitespace
        while src.length() > 0 do
            chr := src[src.length()];
            if chr = " " or chr = Lf or chr = Cr or chr = Tab then
                src := src[1:src.length()-1];
            else
                break;
            endif;
        endwhile;
        fyle.writeString(src & CrLf & delim);
    endforeach;
    write "Extracted " & count.String & " to " & fylename;
epilog
    delete fyle;
end;
```

Load Translatable Strings Method Example

The following method example reloads extracted translatable strings.

```
loadTStrings();
constants
    ExtractDirectory : String = "C:\Temp";
vars
    line        : String;
    token       : String;
    scm         : Schema;
    loc         : Locale;
    tstr        : TranslatableString;
    fylename    : String;
    fyle        : File;
    count       : Integer;
    offset      : Integer;
    linenum     : Integer;
    src         : String;
    errorCode   : Integer;
    errorOffset : Integer;
    errorLength : Integer;
```

```
    result      : Boolean;
begin
    write "";
    fylename := ExtractDirectory & "/" & "TestTstrings-5129.txt";
    create fyle transient;
    fyle.kind := File.Kind_ANSI;
    fyle.mode := File.Mode_Input;
    fyle.fileName := fylename;
    if not fyle.tryOpen() then
        write "**Could not open file =>" & fylename;
        return;
    endif;
    // First line is gives schema name
    line := fyle.readLine();
    linenum := 1;
    if line.length() < 11 or line[1:10] <> "---Schema=" then
        write "**Invalid file format =>" & fylename;
        return;
    endif;
    token := line[11:end];
    scm := rootSchema.getSchema(token);
    if scm = null then
        write "**Unknown schema =>" & token;
        return;
    endif;
    // Second line gives locale name
    line := fyle.readLine();
    linenum := linenum + 1;
    if line.length() < 11 or line[1:10] <> "---Locale=" then
        write "**Invalid file format =>" & fylename;
        return;
    endif;
    offset := 11;
    token := line.scanUntil("=",offset);
    loc := scm.getLocaleLocal(token);
    if loc = null then
        write "**Unknown locale =>" & scm.name & "::" & token;
        return;
    endif;
    if loc.isClone() then
        write "**Locale is a clone =>" & scm.name & "::" & loc.name;
        return;
    endif;
    //discard remaining header lines
    line := fyle.readLine();
    linenum := linenum + 1;
    while line.length() < 8 or line[1:8] <> "-".makeString(8) do
        if fyle.endOfFile() then
            write "**Unexpected end of file";
            return;
        endif;
        line := fyle.readLine();
        linenum := linenum + 1;
    endwhile;
    beginTransaction;
    while true do
```

```
            if fyle.endOfFile() then
                break;
            endif;
            src := "";
            line := fyle.readLine();
            linenum := linenum + 1;
            while line.length() < 8 or line[1:8] <> "-".makeString(8) do
                if src = "" then
                    src := line;
                else
                    src := src & CrLf & line;
                endif;
                if fyle.endOfFile() then
                    write "**Unexpected end of file at line " & linenum.String;
                    return;
                endif;
                line := fyle.readLine();
                linenum := linenum + 1;
            endwhile;
            src := src.trimBlanks();
            if src = "" then
                write "**Empty definition before line " & linenum.String;
                return;
            endif;
            // Get TString name
            offset := 1;
            token := src.scanWhile(
               "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_",
               offset);
            if token = "" or token.length > 30 or token[1] < "A" or token[1] >
               "Z" then
               write "**Invalid TString name before line " & linenum.String &
                     " =>" & src;
                return;
            endif;
            tstr := loc.getTranslatableStringLocal(token);
            if tstr = null then
                write "**Unknown TString name before line " & linenum.String &
             " =>" & src;
                return;
            endif;
            result := tstr.updateCompile( src, errorCode, errorOffset,
                   errorLength);
            if result then
                write "**Error in TString before line " & linenum.String &
                    " =>" & src;
                write "**Error " & errorCode.String & "=" &
                    process.getErrorText(errorCode) & "; offset=" &
                    errorOffset.String;
                return;
            endif;
            count := count + 1;
        endwhile;
        commitTransaction;
        write "Loaded " & count.String & " from " & fylename;
    epilog
```

```
        if process.isInTransactionState() then
            write "**aborting transaction";
            abortTransaction;
        endif;
        delete fyle;
    end;
```

Translating Forms

You can access the Form Browser that enables you to translate forms, by selecting:

- The **Translate Forms** command from the Schema menu in the Jade Platform development environment, to translate forms for the locales supported by the current schema.
- The **Jade Translator** program icon from your Jade program folder, to run the standalone Jade Translator utility when the **Schema** class **formsManagement** property is set to the default value of **FormsMngmt_Multi_Multi** (**0**). For more details, see "Jade Translator Utility", later in this chapter.

Notes When multiple translations of a form exist, the only changes that are propagated through all translations are the addition and deletion of controls. All other changes to a form and its controls (for example, size, captions, positions, security levels, and so on) are made to the current translation only.

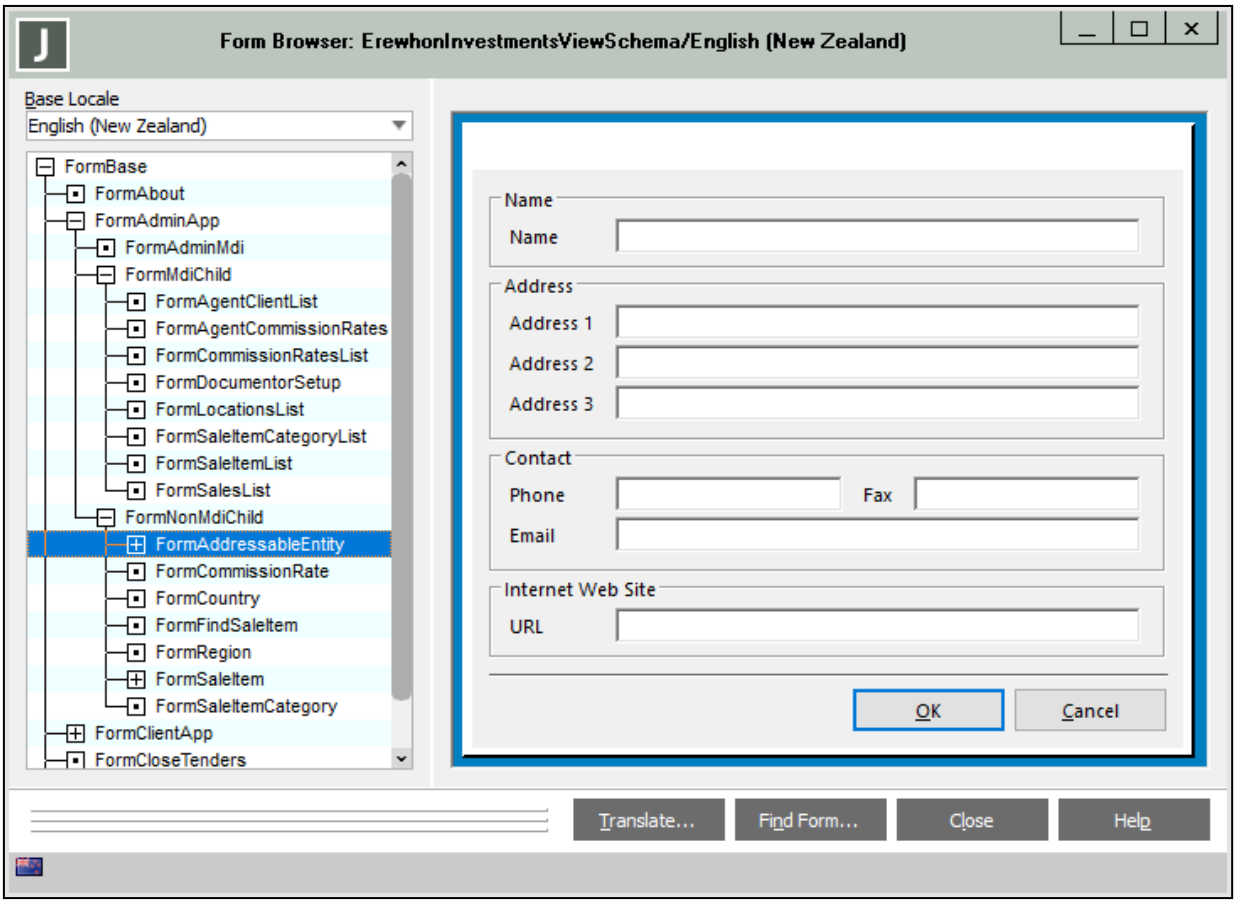
Only one Form Browser for the current schema can be open at any time. If a Form Browser is already open for that schema, it is brought to the top when you select the **Translate Forms** command from the Schema menu. However, you can have concurrent open Form Browser windows for different schemas in a development environment session.

The Form Browser enables you to browse forms for your supported locales.

» To access the Form Browser from the Jade Platform development environment

- Select the **Translate Forms** command from the Schema menu.

The Form Browser for the current locale is then displayed, as shown in the following image.



Using the Form Browser

To browse the forms for a selected locale

1. Select the appropriate locale from the **Base Locale** combo box.
2. From the form list, select the form that you want to translate.

By default, the selected form is then displayed at the right of the browser. (You can suppress the form preview, by selecting the **Do Not Show Form Previews** command from the browser Options menu. If previews are suppressed, you can enable them by selecting the **Show Form Previews** command from the browser Options menu.)

To translate the selected form

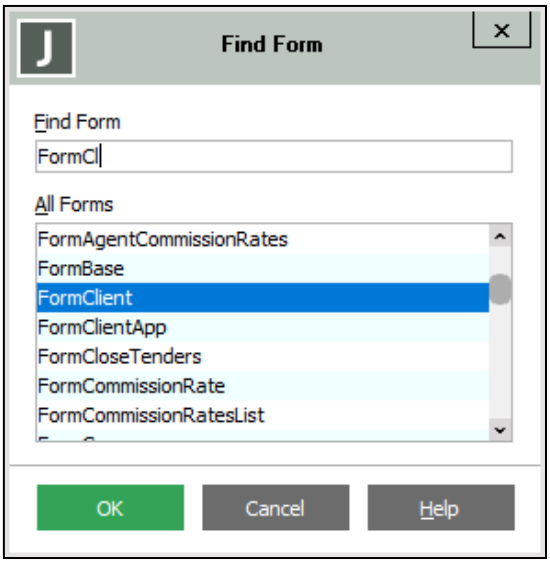
- Click the **Translate** button.

The Jade Translator is then activated. (For details, see "Using the Jade Translator", later in this chapter.)

To locate a specific form in the form list

- Click the **Find Form** button.

The Find Form dialog, shown in the following example, is then displayed.



Use the Find Form dialog to locate a specific form in your schema.

Using the Find Form Dialog

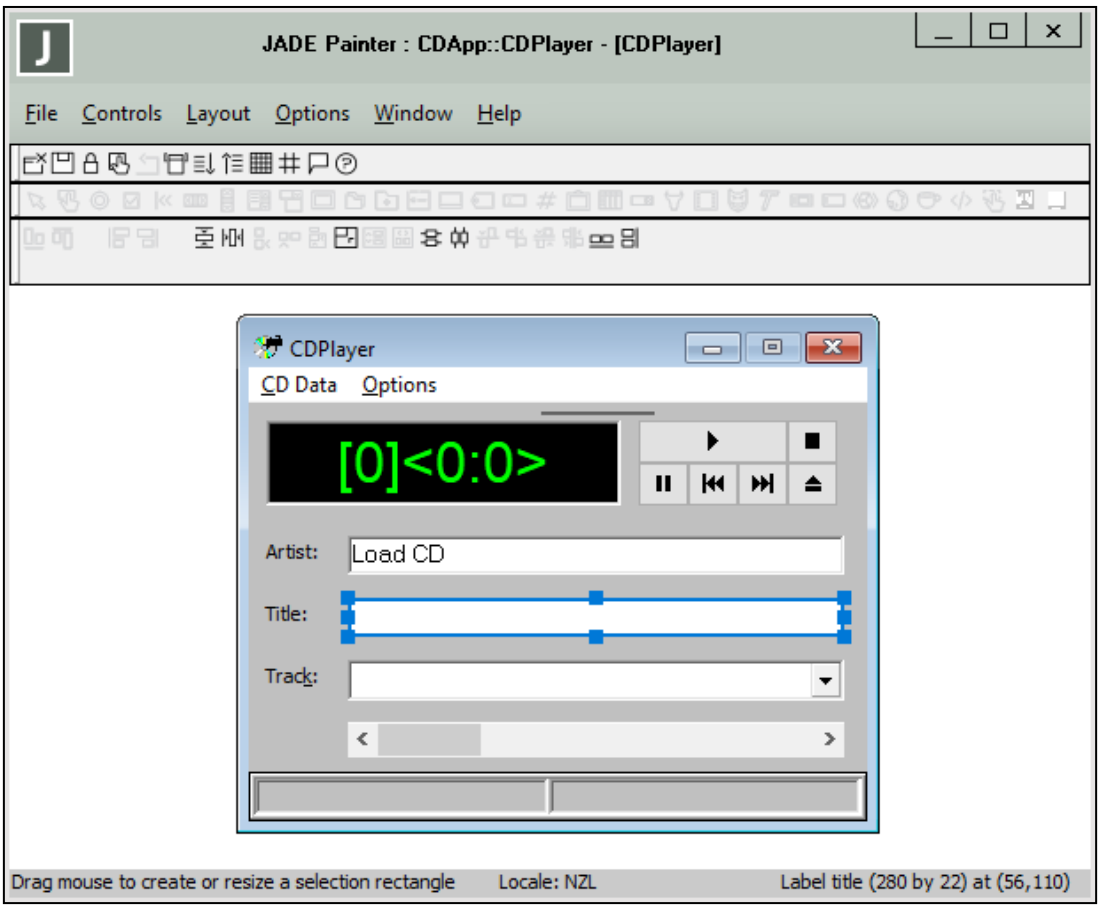
To locate a form

1. Perform one of the following actions.
   - Specify the first character or the first few characters of the form name in the **Find Form** text box. The first form matching your search criterion is selected in the **All Forms** list box.
   - Select the required form in the **All Forms** list box.
2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selection.

The selected form is then highlighted in the Form Browser form list and a form preview displayed at the right of the browser window.

Using the Jade Translator

The Jade Translator, shown in the following image, is activated when you click the **Translate** button in the Form Browser.



The Jade Translator is a subset of the Jade Painter, and enables you to translate the form selected from the Form Browser. For details, see Chapter 5, "Using the Painter and Form Wizard".

Notes You cannot add or delete forms, controls, or menu items by using the Jade Translator. In addition, you cannot make changes that would necessitate the recompilation of methods. For example, you can modify the visual attributes of forms, controls, or menu items (that is, the size, position, color, captions, and so on) but you cannot modify their names or event mappings.

If you have mapping logic on subclassed controls, other processes such as the Jade Painter, Translator utility, or the loading of schemas may also execute that logic. The logic therefore may need to perform checks to determine if it is running in the user application environment, to ensure that exceptions are not generated in these other situations.

The status line displays the three-letter abbreviated country code of the form being edited (for example, **USA** or **NZL**).

Jade Translator Utility

The Jade Translator utility is installed in your database (system) directory as part of the Jade installation process, to enable you to translate strings or forms as a separate application from the Jade Platform development environment.

Note We recommend that you do not use the Jade Translator utility to translate forms when the **Schema** class **formsManagement** property is set to **FormsMngmt_Single_Single** (**1**) or **FormsMngmt_Single_Multi** (**2**).

The default multiuser installation sets up the **Jade MultiUser \ Jade Translator** shortcut for the Jade Translator utility. The following table lists examples of the properties required to run the Jade Translator utility in multiuser mode.

Property	Example
Command line	d:\Jade\bin\jade.exe path=s:\jade\system app=JadeTranslator
Working directory	d:\jade\bin

The following table lists examples of the properties required to run the Jade Translator utility in single user mode.

Property	Example
Command line	jade.exe path=c:\jade\system app=JadeTranslator server=singleUser
Working directory	c:\jade\bin

Only one Jade Translator Schema/Locale Browser for the current schema can be open at any time. If a Jade Translator Schema/Locale Browser is already open for that schema, it is brought to the top when you select the **Jade Translator** program icon from your Jade program folder. However, you can have concurrent open Jade Translator Schema/Locale Browsers for different schemas in a development environment session.

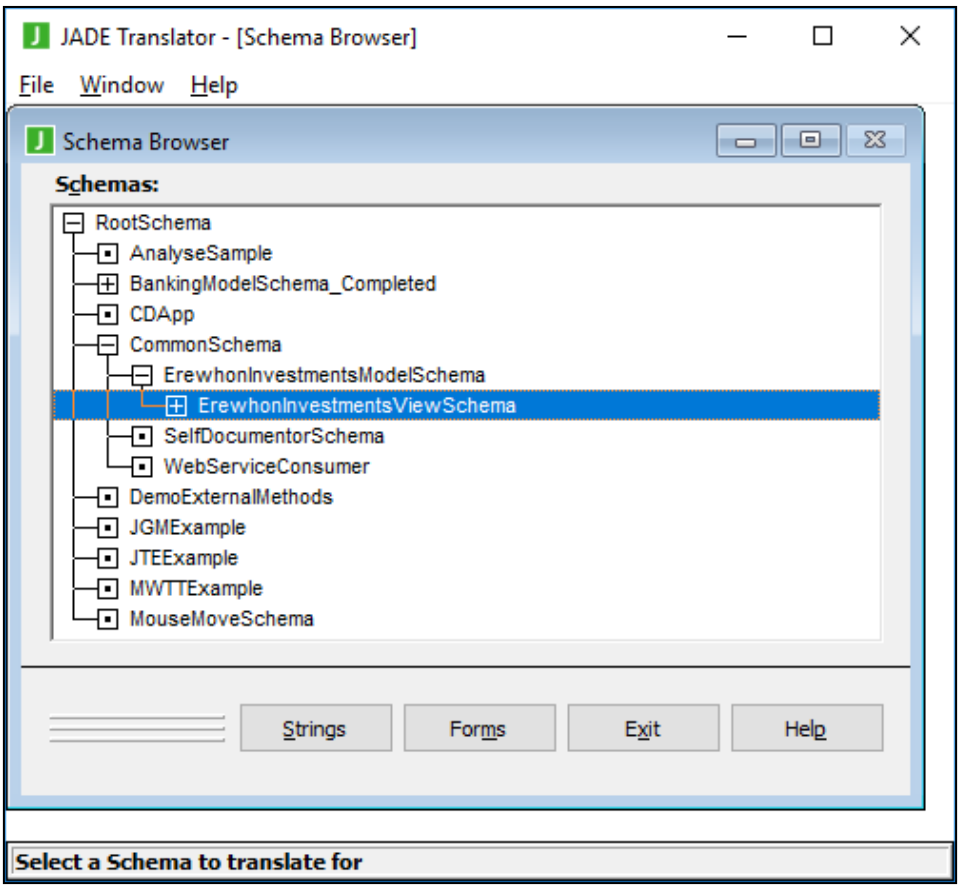
Note If you have mapping logic on subclassed controls, other processes such as the Jade Painter, Translator utility, or the loading of schemas may also execute that logic.

The logic therefore may need to perform checks to determine if it is running in the user application environment, to ensure that exceptions are not generated in these other situations.

To access the Form Browser from the Jade Translator utility

1. Select the **Jade Translator** program icon from your Jade program folder, to run the standalone Jade Translator utility. The Jade Translator sign-on dialog is then displayed.
2. Specify another user identifier in the **User Id** text box, if required.
3. Specify the name of your Jade password in the **Password** text box, if required. As this is a hidden text box for security reasons, each character is represented by an asterisk (*****).
4. Click the **OK** button. (Alternatively, click the **Exit** button to exit from Jade.)

If you have specified a valid user identifier and optional password, the Jade Translator Schema Browser window is then displayed, as shown in the following image.



5. In the **Schemas** list box, select the schema whose strings or forms you want to translate.

6. Click the **Strings** button if you want to translate strings. If there are strings available for translation in the selected schema, the String Browser for the selected schema and locale is then displayed. For more details, see "Translating Strings", earlier in this chapter.

   Alternatively, click the **Forms** button if you want to translate forms. The Form Browser for the selected schema and locale is then displayed. For more details, see "Translating Forms", earlier in this chapter.

Jade Translator Menus

The Jade Translator menu bar contains the following menus.

- File
- Edit (when translating strings only)
- Options (when translating forms only)
- Window
- Help

These menu options are described in the following sections.

File Menu

Use the File menu in the Jade Translator to exit from the utility. (You can also exit from the Jade Translator by clicking the **Exit** button.)

Exiting from the Jade Translator Utility

Use the **Exit** command from the File menu to exit from the Jade Translator utility.

To exit from the Jade Translator utility

- Click the **Exit** command on the File menu.

The Jade Translator utility is then terminated.

Edit Menu

The Edit menu is displayed only when a String Browser window is active. The String Browser child window lists all strings defined for the selected schema.

Using the Undo Command

You can undo the last action in the definition editor pane, if required, or you can perform multiple undo operations.

To undo your last action

- Select the **Undo** command in the Edit menu in the Jade Translator window.

The last action (for example, a keystroke) that you actioned in the editor pane is then undone.

This command is disabled when there is no action that can be undone.

Using the Cut Command

You can cut a selected portion of text in the String Browser definition editor pane to the clipboard, if required.

To cut selected text

- Select the **Cut** command in the Edit menu in the Jade Translator window.

  Alternatively, press Shift+Delete.

The selected text is then cut (logically deleted) from the definition editor pane, and moved to the clipboard.

This command is disabled when there is no text selected in the editor pane.

Using the Copy Command

You can copy text selected in the definition editor pane to the clipboard, if required. This text can then be pasted at another position in the definition editor pane, or in another editor; for example, a Jade method or a text editor such as Notepad.

To copy selected text to the clipboard

- Select the **Copy** command in the Edit menu in the Jade Translator window.

  Alternatively, press Ctrl+Insert.

The selected text is then copied to the clipboard.

This command is disabled when there is no text selected in the editor pane.

Using the Paste Command

You can paste text from the clipboard into the current string translation, if required. The pasted text can be a selection copied or cut from the current string translation, or from another editor.

To paste text at the current caret position

- Select the **Paste** command in the Edit menu in the Jade Translator window.

  Alternatively, press Shift+Insert.

The text is then copied from the clipboard, starting at the current caret position.

This command is disabled when there is no selected text in the clipboard.

Options Menu

The Options menu is displayed only when the **Forms** button in the Jade Translator window is pressed.

Use the **Do Not Show Form Previews** command in the Options menu from the Form Browser to suppress the display of form previews in the Form Browser.

Form previews are displayed by default, to enable you to ensure that the form that you select for translation to another locale is the correct form. (A form may not always be obvious from its form name to those wanting to translate or maintain a translated form.)

If you have previously selected this command so that form previews are not displayed in the Form Browser, this command is displayed as the **Show Form Previews** command in the Options menu from the Form Browser.

Using the Do Not Show Form Previews Command

You can suppress the display of form previews in the Form Browser, if required.

To turn off the display of form previews

- Select the **Do Not Show Form Previews** command in the Options menu from the Form Browser.

The Forms Browser then contains only the **Base Locale** combo box and the forms list; the form selected in the form list is not displayed.

Using the Show Form Previews Command

You can display form previews that you had previously suppressed.

To show form previews

- Select the **Show Form Previews** command in the Options menu from the Form Browser.

The Forms Browser then displays the preview of the form selected in the form list for the current locale.

Window Menu

The Jade Translator Window menu provides standard facilities to select tiling or cascading of windows. It also displays a list of the currently enabled windows.

The lower portion of the Window menu lists all currently open windows (after these commands) in the order in which they were opened. The current, or active, window is indicated by a check mark to the left of the window number and name.

Tip To access an open window, select the appropriate window from the list in the Window menu. Alternatively, you can use the Ctrl+F6 shortcut keys to cycle through all open MDI child windows.

The commands available on the Jade Translator Window menu are described in the following subsections.

Arrange Icons

When you minimize a window, it becomes an icon. You can use the mouse to drag icons individually, or you can use the **Arrange Icons** command from the Windows menu to arrange all minimized icons in an orderly manner across the bottom of the screen.

To arrange minimized icons

- Select the **Arrange Icons** command from the Window menu

All minimized window icons are then arranged in an orderly manner across the bottom of the screen.

Cascade

Use the **Cascade** command from the Window menu to arrange open windows in an overlapping pattern so the title bar of each window remains visible. (This is the default Jade Translator window option.)

To cascade open windows

- Select the **Cascade** command from the Window menu

All open windows are then overlapped so that the title of each window is visible.

New Window

Use the **New Window** command from the Window menu to open another window of the current window; for example, a new Schema/Locale Browser window is opened if the Schema/Locale Browser is the current window or a new Form Browser window is opened if the Form Browser is the current window.

To open a new Jade Translator window

- Select the **New Window** command from the Window menu

A new copy of the current window is then displayed.

Tile Horizontal

Use the **Tile Horizontal** command from the Window menu to resize and arrange windows without overlap so that each window is wider than it is long so that all windows are visible.

To tile open windows horizontally

- Select the **Tile Horizontal** command from the Window menu

All open windows are then sized so that they are arranged horizontally.

Tile Vertical

Use the **Tile Vertical** command from the Window menu to resize and arrange windows without overlap so that each window is longer than it is wide so that all windows are visible.

To tile open windows vertically

- Select the **Tile Vertical** command from the Window menu

All open windows are then sized so that they are arranged vertically.

Help Menu

Use the commands in the Jade Translator Help menu to access the standard CUA help options.

The commands available on the Jade Translator Help menu are described in the following subsections.

Index

Use the Jade Translator Help menu **Index** command to open the Jade online help.

To access the online help, perform one of the following actions

- Select the **Index** command from the Help menu
- Press F1

The first page of the Jade online help directory document is then displayed, which provides a summary of and hyperlinks to all documents in the Jade Platform product information library.

The Jade Platform Product Information Library document contains the titles, file names, and contents summary of the Jade Platform documentation.

To navigate to the first page of any document to which you have access

- Click the hyperlink in the first column of the table on the page that is first displayed.

For a summary of the information contained in a document

- Click the hyperlink in the second column of the table on the page that is first displayed

Note PDF documentation files are not part of the installation process, and must be downloaded and installed from the Jade website or the release medium separately, if required, into the **documentation** folder of your Jade installation directory.

For details about using Jade help files, see "Jade HTML5 Online Help" or "Jade Platform Product Information Library in Portable Document Format", in Chapter 2.

About Jade Translator

Use the Jade Translator Help menu **About Jade Translator** command to access information about the current release of the Jade Translator. This information includes the Jade release version and copyright information.

To access the Jade Translator version information

- Select the **About Jade Translator** command from the Help menu

The About Jade Translator dialog is then displayed. This dialog is for display purposes only.

Translating Messages

All Jade Object Manager display text and error messages are stored in the **jadmsgs.eng** file.

Note These are the default English settings for all text.

To translate the messages and errors, use any text editor (for example, Notepad). Change only the text enclosed in quotation marks (**""**).

Do *not* change the numbers or the position of the numbers. All numbers must start in column 2 and can be preceded by a hyphen.

The two stroke characters (*//*) in columns 1 and 2 are treated as comments, which are ignored when the Jade Object Manager is running.

To create several copies of the language file, save the file in the **jadmsgs.*xxx*** format. The ***xxx*** value can be any three characters representing the language. For example, to save the copy as a French translation file, you might save the file as **jadmsgs.fre**.

To specify a language file for the **jadmsgs.eng** file when running Jade, specify the following in the [JadeClient] section of your Jade initialization file.

```
Language=xxx
```

The *xxx* value is the file extension that you used when you saved the translated language file; for example, **eng** or **fre**. (The default value of the **Language** parameter is **<default>**.)

If you specify a parameter value of more than three characters in the [JadeClient] section of the Jade initialization file, only the first three characters are used. For example, if you specify **Language=english**, the Jade Object Manager interprets this as **Language=eng**.

Chapter 12 Adding and Maintaining HTML Documents

This chapter covers the following topics.

- Overview
- Accessing the HTML Document Browser
- Using the HTML Wizard to Add an HTML Document
  - Naming your HTML Document and its Source
  - Specifying the Class Name and Its Property Names
- Changing an HTML Document
- Editing an HTML Document
- Extracting and Loading an HTML Document
  - Extracting a Specific HTML Document
- Saving an HTML Document as a File
- Reloading an HTML Document
- Removing an HTML Document
- Reimplementing Methods to Interrupt the Processing Cycle
- HTML Document Implementation
  - Method Handling
  - Generating Data for JADE_TAG Tags Only

Overview

Jade simplifies the use of HyperText Markup Language (HTML) pages from an external source by providing the following facilities.

- Importing HTML pages using a wizard or in batch mode
- Creating the classes and properties for imported HTML pages
- Methods for updating an HTML page by using the class and properties associated with the page and for receiving the updated instance of a class as a result of a web request
- User exits
- Administration tools for maintaining HTML pages
- Load and extract facilities

The HTML process flow is as follows.

1. A request is received from the web browser.
2. The web application receives the request and determines that it is an HTML page request.
3. If the application determines that it is the first request, it finds the home page and returns it to the web browser.
4. If it is not the first request, the application creates an instance of the class associated with the page and updates its properties.
5. Jade logic accesses this instance and performs the required processing.
6. Jade logic sends a response back to the web browser, which could be the same page updated or a completely new page.
7. Repeat the process, starting from step 1.

Note You can break into any phase of this cycle by using method reimplementation. For details, see "Reimplementing Methods to Interrupt the Processing Cycle", later in this chapter.

In this chapter, an HTML file is referred to as an HTML *document* and the presentation of that file on a web browser at run time is referred to as an HTML *page*.

Accessing the HTML Document Browser

The HTML Document Browser enables you to add and maintain HTML documents in the current schema. Each schema in the Jade database can have a collection of HTML documents.

» To open an HTML Document Browser, perform one of the following actions

- Select the **HTML Documents** command from the Browse menu in the Class or Schema Browser
- Press Ctrl+U

An HTML Document Browser window is then opened and the HTML menu is displayed in the menu bar. If you have not yet defined an HTML document, nothing is displayed in the HTML Document Browser.

The HTML menu contains the commands listed in the following table.

Command	For details, see…	Action
Add	Using the HTML Wizard to Add an HTML Document	Displays the HTML Wizard
Change	Changing an HTML Document	Displays the HTML Wizard
Edit HTML	Editing an HTML Document	Displays the HTML editor window
Extract	Extracting a Specific HTML Document	Displays the Extract dialog
Save HTML	Saving an HTML Document as a File	Displays the common Save As dialog
Reload	Reloading an HTML Document	Displays the Reload Document dialog
Remove	Removing an HTML Document	Deletes the selected HTML document

When the class associated with an HTML document is in use by an application:

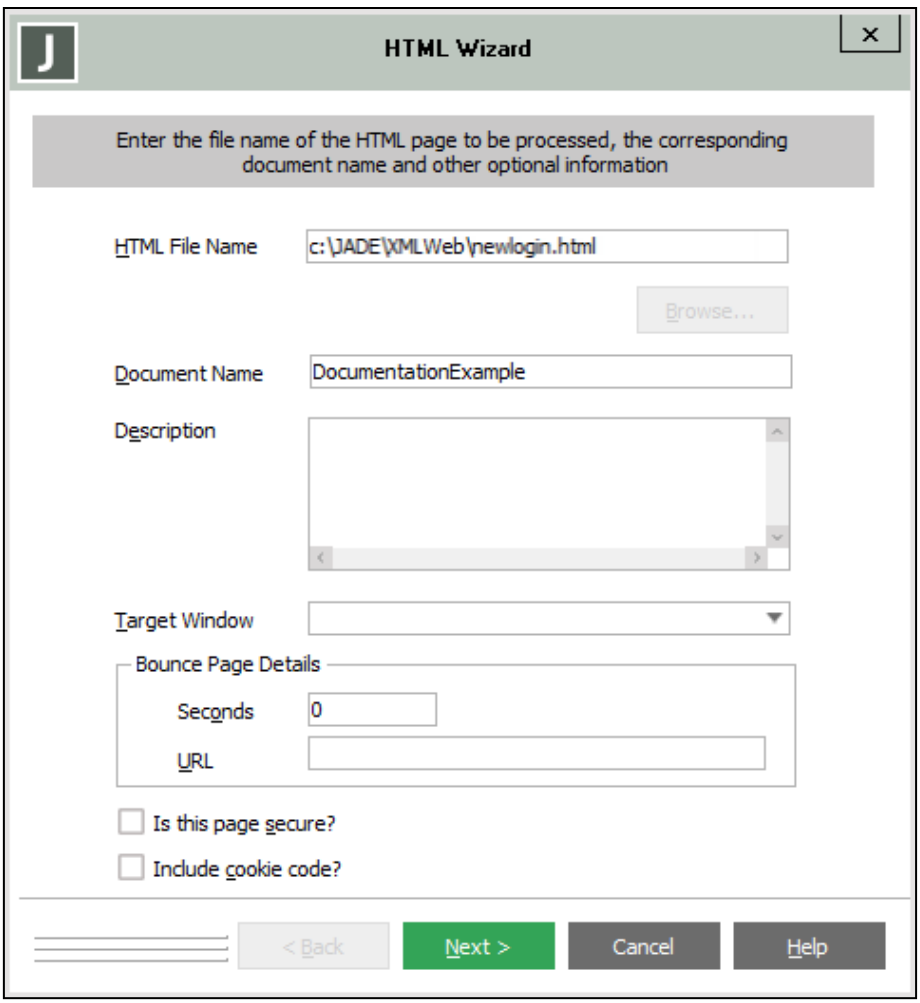
- The HTML menu **Change**, **Reload**, and **Remove** commands are disabled.
- The HTML text for a document may be displayed but you cannot update the HTML.

Using the HTML Wizard to Add an HTML Document

To add an HTML document to the current schema

- Select the **Add** command from the HTML menu.

The first page of the HTML Wizard, shown in the following image, is then displayed.



The HTML Wizard:

- Accepts an HTML file (that is, an HTML document) as input and processes the file to create a set of persistent objects that represent the file contents (that is, an HTML page).
- Creates a class and properties corresponding to the **Input**, **Select**, and **JADE_TAG** tags that are present in the HTML.

- Enables you to change the default class and property names and to maintain a mapping of the HTML property with the Jade property.

When using the HTML Wizard, click the:

- **Next >** button to continue defining the HTML document.
- **< Back** button to return to the previous page. You can then change your selections, if required.
- **Cancel** button to close the HTML Wizard.

You can use the **< Back** and **Next >** buttons to go through the HTML Wizard pages, making the appropriate selections, as required.

When adding an HTML document:

- You cannot use **JADE_TAG** tags for a name. As Jade needs to create properties based on the name, the name in the HTML file must be a literal value.
- You can use duplicate names.
- When processing the HTML document, as Jade looks for specific tags and does not parse the document for syntactical correctness, invalid constructs such as a missing closing angle bracket (>) can cause unexpected results.
- Attributes for input tags (including **select**) can be set without a **JADE_TAG** having to be included in the definition. For example, to disable a button called **addsave**, if you were to code **setAttributes('addsave', 'disabled', '');** in the **updateValues** method, this would cause the button to be disabled.
- You can use the **JADE_TAG** tags to define translatable strings in the HTML document source. (For details, see "JADE_TAG Tag Notes" and "Generating Data for JADE_TAG Tags Only", later in this chapter.)

Naming your HTML Document and its Source

The first page of the HTML Wizard enables you to define the name of your HTML document and its source.

To specify a name for your HTML document

1. In the **HTML File Name** text box, specify the name and location of the file from which to create the HTML document. Although this can be any valid file type, it is most often an HTML file (that is, .**html** or .**htm**). Alternatively, click the **Browse** button. The common File dialog is then displayed, to enable you to select the source file from the appropriate location.
2. In the **Document Name** text box, specify the name that you require for your HTML document. You must enter a document name, which can contain underscore characters but it cannot contain spaces. The document name length is restricted to **100** characters.
3. In the **Description** text box, enter the free-format text that describes your HTML document, if required.
4. In the **Target Window** drop-down list box, select the target attribute of the HTML document, if required. You can select one of the HTML target attributes listed in the following table.

Target	Loads the linked document into …
_blank	A new blank window, which is not named
_media	The Media Bar (available in Internet Explorer 6 or later)

Target	Loads the linked document into …
_parent	The immediate parent of the document containing the link
_search	The browser's search pane (available in Internet Explorer 5 or later)
_self	The window in which the link was clicked (the active window), which is the default target attribute
_top	The topmost window

5. In the Bounce Page Details group box, optionally perform the following actions.

   a. In the **Seconds** text box, enter the number of seconds after which the web page specified in the **URL** text box is displayed if there has been no user action.

   b. In the **URL** text box, specify the URL of the web page that is displayed when there has been no user action for the number of seconds specified in the **Seconds** text box.

6. If the HTML page is secure, check the **Is this page secure?** check box. The form action and all hyperlinks are then prefixed with **https**. By default, the HTML page is not secure (that is, this check box is unchecked).

   To dynamically change the page security status, set the Boolean value of the **securePage** property in the **JadeHTMLClass** class to the appropriate value.

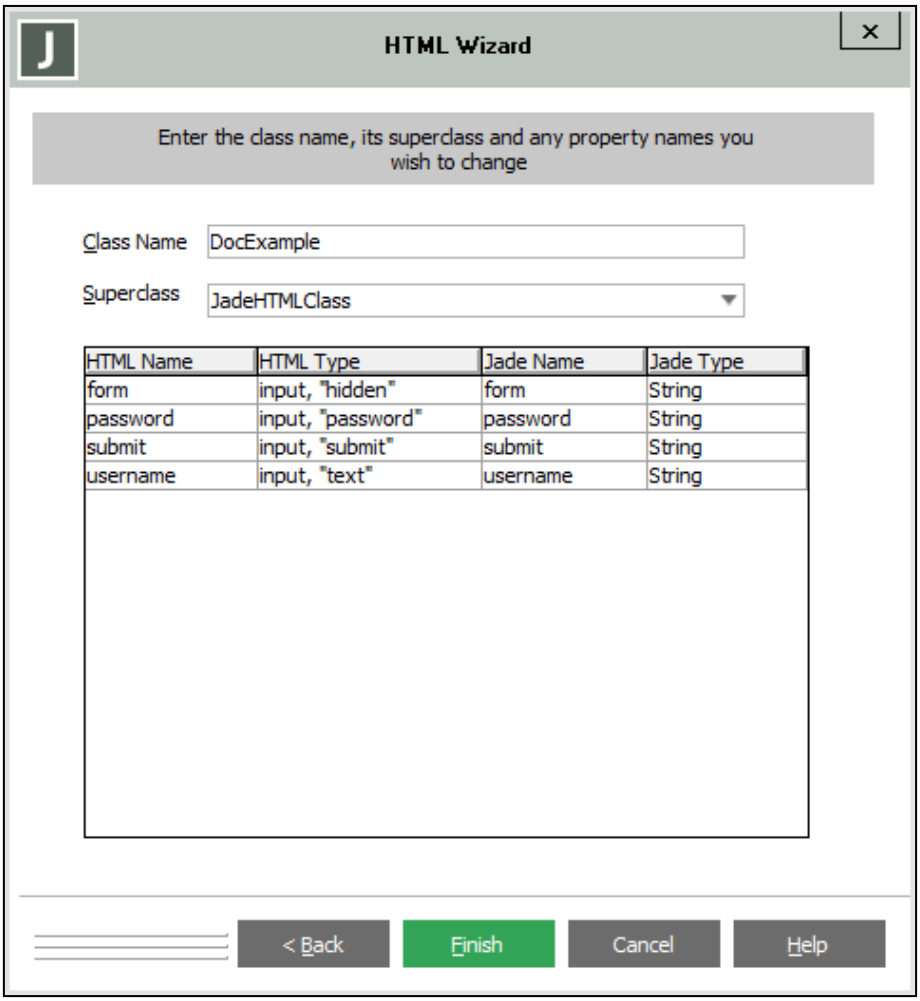
7. If you want to include cookie code in the HTML document, check the **Include cookie code?** check box. By default, cookie code is not included (that is, this check box is unchecked).

8. Click the **Next >** button. Alternatively, click the **Cancel** button.

Specifying the Class Name and Its Property Names

When you have defined the name of your HTML document, the next page of the HTML Wizard enables you to specify the class name, its superclass, and to change any Jade property names.

An example of the second page of the HTML Wizard is shown in the following image.



To specify your HTML document class and property name options

1. In the **Class Name** text box, specify the Jade class name that you require for the HTML document.
2. In the **Superclass** list box, select the **JadeHTMLClass** or one of its subclasses that you require for the superclass of your HTML document class. (You can add your HTML document class only as a subclass of class of **JadeHTMLClass** or one of its subclasses.)

   If you are defining the first HTML document for the schema, only the **JadeHTMLClass** class is displayed in this list box.
3. In the **Jade Name** column, change the name of each of the Jade properties that you require, by clicking in the appropriate cell and specifying the valid Jade property name that meets your requirements.

   You can change values only in the **Jade Name** column.

The following table lists the HTML tags that cause a Jade property to be created and the property types to which the Jade types are converted.

HTML Type	Jade Type
Input, radio (**true**, if set)	Boolean
Input, checkbox (**true**, if set)	Boolean
Input, submit	String
Input, image	String
Input, reset	String
Input, button	String
Input, text	String
Input, password	String
Input, hidden	String
Input, file	String
Select	String
Select, multiple	StringArray
TextArea	String
JADE_TAG	String / JadeHTMLClass

Each HTML document can include any number of **JADE_TAG** tags but each tag must have a **name** attribute. If the name is not unique, the same value is set for the tags. **JADE_TAG** tags can have one of the following formats.

- <*JADE_TAG name*=headerTag>

  This form of the **JADE_TAG** tag creates a property of primitive type **String**. At generation time, this tag is replaced by the value of the property (that is, by the **headerTag** value). Note that if this value is an HTML string that contains **Input** type tags, there is no equivalent property for the class and it is your responsibility to process any information that is returned for these tags.

- <*JADE_TAG name*=includeHeader *value*=Header>

  This form of the **JADE_TAG** tag creates a property of type **JadeHTMLClass**. The **value** attribute should resolve to a subclass of the **JadeHTMLClass** class.

- <*JADE_TAG name=nameLabel* _translate=*translatableStringName*>

  This form of the **JADE_TAG** tag enables you to define translatable strings in the HTML document source.

For more details, see "JADE_TAG Tag Notes" and "Generating Data for JADE_TAG Tags Only", later in this section. See also "Using the HTML Wizard to Add an HTML Document", earlier in this section.

You can add additional attributes to the tags at run time, by calling the **JadeHTMLClass** class **setAttributes** method on the transient class. For details, see "Reimplementing Methods to Interrupt the Processing Cycle", later in this chapter.

4. Click the **Finish** button when you have finished specifying your HTML document. Alternatively, click the **< Back** button to redisplay the previous page or the **Cancel** button to abandon your selections.

When you click the **Finish** button, the class and all of its associated properties are created. The class is created as a subclass of **JadeHTMLClass** in the schema to which you added the HTML document.

Notes You can delete neither the created class nor any of its properties. An HTML document class is deleted only when its corresponding HTML document is deleted. (For details, see "Removing an HTML Document", later in this chapter.) However, you can add additional properties to the class.

These properties can be changed and deleted by using the standard change and delete operations.

To set up mutually exclusive radio (option) buttons, the **name** attribute must be the same and the **value** attributes must be different. (Jade generates properties based on the **value** attribute.) This applies only to radio buttons. For example, the following HTML fragment generates two **Boolean** properties, one called **positive** and one called **negative**. While it is possible to set these two properties in code, the web browser displays only one as being selected (that is, the second property).

```
<input type=radio name=status value=positive>
<input type=radio name=status value=negative>
```

JADE_TAG Tag Notes

The following applies when using **JADE_TAG** tags on your HTML pages.

- The **<*JADE_TAG name*=headerTag>** tag creates a property of primitive type **String**.
- The **<*JADE_TAG name*=includeHeader *value*=Header>** tag creates a property of type **JadeHTMLClass**. Use this mechanism to insert one HTML page into another.

  For example, if your system has a common header, create an HTML file containing only this header information and import this file into Jade. You can then insert this document into another imported document by using the **<*JADE_TAG name*=includeHeader *value*=Header>** syntax. At run time, the HTML will be inserted into the main page.
- The **<*JADE_TAG name=nameLabel* _translate=***translatableStringName*>** tag creates translatable strings in the HTML document source.

  When the HTML is generated for a document containing the **_translate** attribute, a lookup of the specified ***translatableStringName*** string is performed, using the locale defined for the session. If the locale does not exist, it is set to null (**""**), or if the string does not exist in the list of translatable strings, the **<JADE_TAG>** tag is replaced by the ***translatableStringName*** string itself. If the string does exist, the **<JADE_TAG>** tag is replaced by the value of the translatable string for that locale.

  A selective extract of a **JadeHTMLClass** subclass extracts the translatable string definitions of all translatable strings that are defined in the HTML document.

  Note In the case of a patch extract, if the translatable string is not defined in the patch, extracting a patch extract extracts only what is defined in the patch so it is not extracted.
- If a **JADE_TAG** tag is to be used as an attribute of another tag, the **JADE_TAG** tag must be enclosed in single quote characters (**'**) or in double quote characters (**"**). For example, when using this tag with the **<A>** tag, you could specify the following.

  ```
  <A href = "<JADE_TAG name=updateAccount>"> Edit </A>
  ```

  At run time, the value of the **updateAccount** property is used to generate the HTML code; for example, the following is generated if **updateAccount** := **http://www.jadeworld.com";** in your Jade code.

  ```
  <A href = "http://www.jadeworld.com"> Edit </A>
  ```

- Although you can insert **JADE_TAG** tags within **<select>** and **</select>** tags, the value of the property associated with the **JADE_TAG** should contain only **<option>** and **</option>** tags. Inserting any other tag may cause unpredictable behavior (browser-dependent).

See also "Generating Data for JADE_TAG Tags Only", later in this chapter.

Changing an HTML Document

You can modify the name and characteristics of an existing HTML document in the current schema.

When you modify an HTML document, you can change the document name but you cannot change the class name on the second page of the HTML Wizard.

To change an HTML document

1. In the HTML Document Browser, select the page whose values you want to modify (for example, the document class name or one or more property names).
2. Select the **Change** command from the HTML menu.

The HTML Wizard is then displayed. For details about using the HTML Wizard, see "Using the HTML Wizard to Add an HTML Document", earlier in this chapter.

Editing an HTML Document

You can modify an existing HTML document in the current schema. When you edit HTML, you edit the HTML code in the Jade database.

To edit an HTML document

1. In the HTML Document Browser, select the document whose HTML code you want to modify.
2. Select the **Edit HTML** command from the HTML menu. The HTML editor window is then displayed.
3. Edit the HTML code to suit your requirements.

Tip Right-click in the editor pane to access the popup (context) menu, which provides standard edit and search functions. For details, see "Performing Edit Actions", in Chapter 2.

When you close the HTML editor window, a message box prompts you to confirm that you want to save your changes to the HTML.

To save your changes and update the Jade database, click the **Yes** button. You can also press F2 or click the **Save icon** in the toolbar to save your changes. Alternatively, click the **No** button to abandon your changes or the **Cancel** button to continue editing the HTML code until it meets your requirements.

Extracting and Loading an HTML Document

You can extract HTML documents individually (that is, a partial schema definition) or as part of a complete schema extract into schema metadata (.**scm**) and form and data (.**ddb** or .**ddx**) definition files.

For details about extracting an individual HTML document, see "Extracting a Specific HTML Document", in the following subsection.

When loading a file that contains HTML documents, the internal structures are rebuilt by using the extracted information that you load. Existing HTML documents are replaced by incoming documents in the load file that have the same name.

Extracting a Specific HTML Document

From the HTML Document Browser, the **Extract** command from the HTML menu enables you to extract an individual HTML document.

A selective extract of a **JadeHTMLClass** subclass extracts the translatable string definitions of all translatable strings that are defined in the HTML document.

Note As a patch extract extracts only what is defined in the patch, if a translatable string defined in an HTML document is not defined in a patch, it is not extracted with that patch.

» To extract an HTML document

1. In the HTML Document Browser, select the HTML document that you want to extract.
2. Select the **Extract** command from the HTML menu.

The Extract dialog is then displayed, to enable you to select the options that your require. For details about using the Extract dialog, see "Extracting Your Schema", in Chapter 10.

A partial definition schema metadata (.**scm**) file and form and data definition (.**ddb** or .**ddx**) file are then extracted to the directory that contains the file from which the HTML document was created if you do not specify an absolute path.

Saving an HTML Document as a File

Although the HTML document that you add, change, or edit is saved in the Jade database itself, you can save the HTML document outside the Jade database.

» To save the HTML document externally

1. In the HTML Document Browser, select the HTML document that you want to save.
2. Select the **Save HTML** command from the HTML menu.

The common Save As dialog is then displayed, to enable you to specify the name and location of your external HTML file.

The file name defaults to the name of the current HTML document, with an .**htm** suffix. The location defaults to the directory that contains the file from which the HTML document was created; for example:

```
s:\jade\HTMLTest\JADEorder.htm
```

If you did name the HTML file the same as the name of the file from which it was created, a message box advises you that the file already exists and prompts you to confirm that you want to replace it.

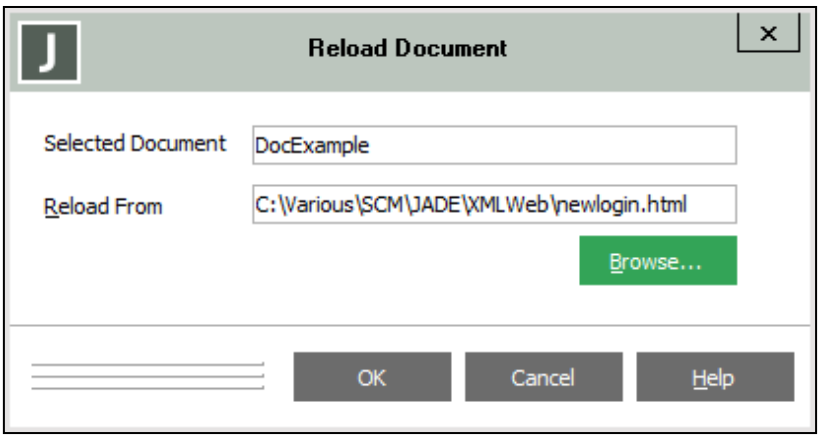
Reloading an HTML Document

You can update an existing HTML document by reloading it from an external HTML file. This replaces the document, retaining common information and removing information that does not exist in the incoming file. (For details about processing multiple HTML files, see the **Schema** class **loadHTMLDocuments** method in Chapter 1 of the *Encyclopaedia of Classes*.)

Note Properties can therefore get deleted. If there are method references to deleted properties, the methods will be marked as being in error.

To reload an HTML document

1. In the HTML Document Browser, select the HTML document that you want to reload.
2. Select the **Reload** command from the HTML menu. The Reload Document dialog, shown in the following image, is then displayed.



   The name of the HTML document selected in the HTML Document Browser is displayed in the **Selected Document** text box. (As this is a read-only text box, you cannot change this value or select another HTML document from this dialog.)
3. In the **Reload From** text box, specify the name and location of the HTML file that you want to reload into your Jade database. If you are unsure of your file directory, click the adjacent **Browse** button to access the Specify HTML File to Reload dialog (the common File dialog) is then displayed, to enable you to select the file and location that you require.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Tip If your application uses the same schema but with different HTML source, you can store the HTML in another object before loading a schema and then replacing it following the schema load process.

The **Schema** class **getHtmlDocumentSource** and **setHtmlDocumentSource** methods enable you to get the HTML source of a document before loading a schema and then set the HTML source to the required value following the schema load process.

Removing an HTML Document

From the HTML Document Browser, the **Remove** command from the HTML menu enables you to remove (delete) the HTML document that is currently selected.

Note You can delete neither the created class nor any of its properties. An HTML document class is deleted only when you delete its corresponding HTML document from the HTML Document Browser.

To remove an HTML document

1. In the HTML Document Browser, select the HTML document that you want to remove.
2. Select the **Remove** command from the HTML menu. Alternatively, press the Delete key. A message box is then displayed, to enable you to confirm that you want to remove the specified HTML page.
3. Click the **Yes** button in the Confirm Delete message box to confirm that you want to delete the HTML document. Alternatively, click the **No** button to abandon the deletion. The Delete Class message box then prompts you to confirm that you want the associated subclass of the **JadeHTMLClass** deleted as well.
4. Click the **Yes** button to confirm that the associated HTML class is also deleted. Alternatively, click the **No** button to abandon the deletion.

The HTML Document Browser is then updated to reflect the removal of the selected HTML document and the subclass for that HTML document is deleted from the Class Browser.

Reimplementing Methods to Interrupt the Processing Cycle

When you require little reimplementation, you simply reimplement the **JadeHTMLClass** class **processRequest** method to process the incoming data. Set up the **goToPage** property in this method to respond with a different page.

When you want full control of HTML pages in your application, for every **JadeHTMLClass** subclass, reimplement the **generateHTMLString**, **processRequest**, **reply**, and **updateValues** methods.

HTML Document Implementation

The **JadeHttp.dll** library supports TCP/IP and Internet named pipe connections.

A web-enabled application containing HTML documents opens a pipe and listens for input. When input is received, it is processed and a reply is sent to the web browser. The application then continues to listen for input.

Note If a start-up form is defined for a web-enabled application, it is assumed that the application will use Jade forms. If no start-up form is defined for a web-enabled application, an HTML page is used.

When a request is received by an HTML document-based application, the **processRequest** method is not called if it is the initial request where the home page is invoked (it is called when additional request is made on that document; for example, a link is clicked) or if the reference tag in the received data is **_jadeReferencePage**. If the reference tag in the received message is **_jadeReferenceClass**, the **processRequest** method is called. Alternatively, you can reimplement the **updateValues** method to set up the required property values for the HTML page.

If no HTML home page is defined for the application, the processing is as follows.

1. When the application is started up, an Internet TCP/IP or named pipe connection is created and the application then listens asynchronously on that connection.
2. When the input is received, it determines if it is the initial request. If it is the initial request, it determines the default home page.

   If the default home page is not specified in the application, an error message is formulated by using the **Application**::**htmlPageNotFoundMessage** method and this error is sent as the reply.

If a home page is found, the following processing occurs.

1. A transient instance of the corresponding class for this page is created.
2. The message **reply** is sent to the instance. This message calls the **updateValues** method.

   The **reply** method returns a string, which is then sent as the response to the request. The default implementation of this method calls the **JadeHTMLClass** class **generateHTMLString** method, which generates the HTML with the updated values from the transient instance.

   It also includes the **_ jadeReferenceClass** class name and the **_ jadeDocumentName** HTML document name as hidden fields in the string. In addition, the session id of the **currentSession** system variable is also included as a hidden field. If the page was defined as secure, all URLs on the HTML page are prefixed with **https**.
3. The transient instance is deleted and the application goes back to waiting for input.
4. If it is not the initial request, the **_ jadeReferenceClass** and **_ jadeDocumentName** values are extracted from the incoming string and a transient instance of this class is created.

The rest of the processing for HTML pages is the same as that for the home page, except that the **JadeHTMLClass**::**processRequest** method updates the property values of the instance from the request string.

Method Handling

The incoming request is processed as follows.

- If the message contains an **_executeMethod** attribute, the method name associated with the attribute is executed and the value returned from the **readPipeCallback** method is returned to the web browser, as shown in the following example.

  ```
  http://ourtest2a/jade/jadehttp.dll?WebTimeSheet&_
  executeMethod=MyApp::generateTimeSheetXML&employee=WilburWinstanley
  ```

- If the message contains an **_ jadeReferencePage** attribute, the HTML document name associated with this attribute is used to get the HTML document, the HTML is generated, and the response is returned to the web browser, as shown in the following example.

  ```
  http://ourtest2a/jade/jadehttp.dll?WebTimeSheet&_
  JadeReferencePage=TimeSheetHelp
  ```

- If the message contains a **_ jadeReferenceClass** attribute, a transient instance of the specified class is created and the **JadeHTMLClass** class **processRequest**, **updateValues**, and **reply** methods are sent to it. The response from the **reply** message is returned to the web browser.
- The appropriate session is retrieved or created and the **currentSession** system variable is set to this value.
- If the incoming request contains none of the above attributes:
  - If there is a start-up form for the application, it is assumed to be a standard web-enabled application and processed as such.
  - If there is no startup form and there is a home page defined for the application, this document is retrieved, the HTML is generated, and the response is returned to the web browser.
  - If none of the above conditions is met, the **Application**::**htmlPageNotFoundMessage** results, which returns a string to be sent back to the web browser.

For details about the methods defined in the JadeHTMLClass class, see Chapter 1 of the *Encyclopaedia of Classes*.

Generating Data for JADE_TAG Tags Only

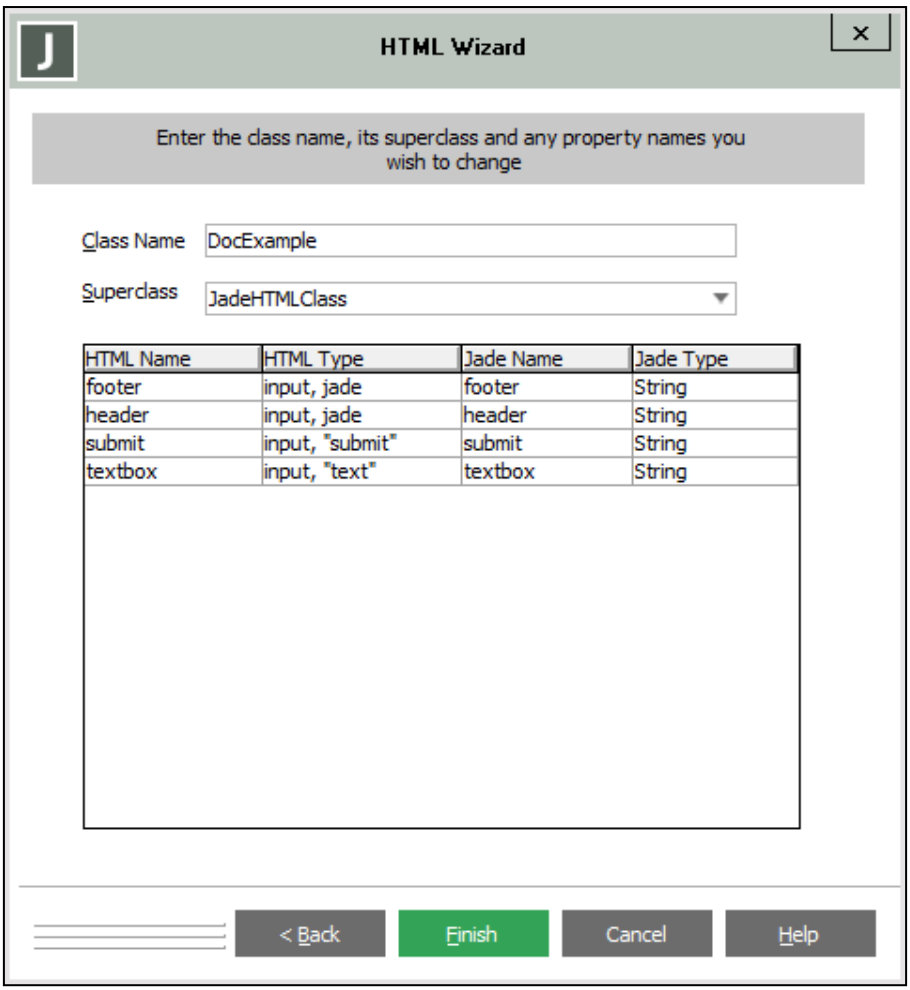
If you want to process only the **JADE_TAG** tags when HTML is generated for a page, set the **JadeHTMLClass** class **generateHTMLForJadeTagsOnly** property to **true**, in which case all other tags will not be altered in any way. The default for this property is **false**, which will process all tags. See also "JADE_TAG Tag Notes", earlier in this chapter.

The steps in the following example illustrate the behavior of HTML document processing when HTML is generated for **JADE_TAG** tags only.

1. Use the HTML Wizard to import the following HTML.

```
<html>
<body>
<jade_tag name="header">
<form method="POST" action="http://localhost/jade/jadehttp.dll?S2">
<input type=text name=textbox value="">
<input type=submit name="submit" value="OK">
</form>
<jade_tag name="footer">
</body>
</html>
```

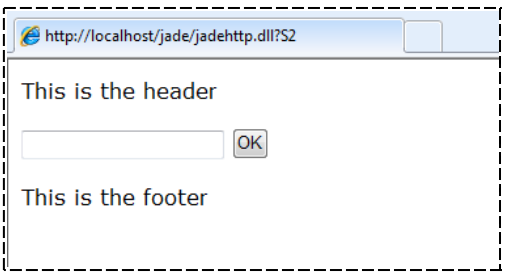
2. The HTML Wizard is then displayed as is shown in the following image.



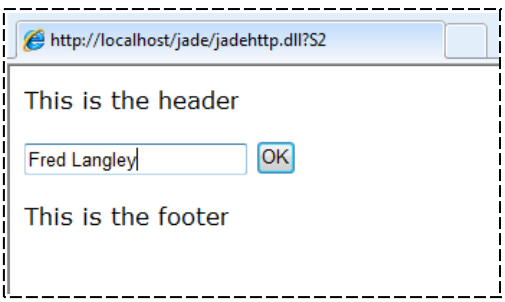
3. Add the following Jade code in the **updateValues** method for the **DocExample** class.

```
vars
begin
    header := "This is the header";
    footer := "This is the footer";
    return inheritMethod();
end;
```

4. The following image shows the web browser when this page is opened.



5. Enter some data in the text box, as shown in the following image, and then click the **OK** button.



6. The page is now processed by web application framework and the text box has the value **Fred Langley** displayed in it.

7. The web application framework then calls the **updateValues** method again and generates the HTML for all tags including the **<input type=text …>** tag, which is sent back to the web browser.

   Note Because the **<input type=text …>** is also processed, the generated HTML contains the value of the corresponding property for this tag.

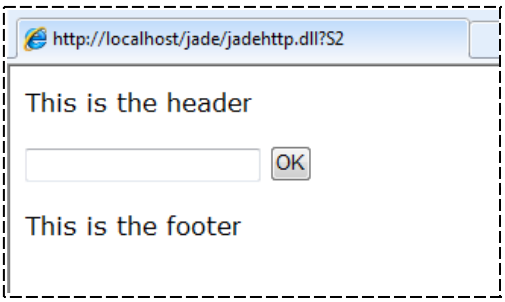
   The web browser is the same as that shown in step 5 of this example.

8. Add a **create** method with the following code to the **Test** class.

```
create() updating;
begin
    generateHTMLForJadeTagsOnly := true;
end;
```

9. Repeat steps 4 through 7 of this example.

The web browser then looks like the following image.



The text box is now empty instead of having the text **Fred Langley** displayed in it, because now only the **JADE_TAG** tags are processed so the **<input type=text …>** tag takes on its default value (which in this case is **""**, as can be seen in the HTML shown in step 1 of this example).

10. One way to make this simple HTML work as before but still processing only **<jade_tag>** tags is to alter the HTML, as follows.

```
<html>
<body>
<jade_tag name="header">
<form method="POST" action="http://localhost/jade/jadehttp.dll?S2">
<input type=text name=textbox value="<jade_tag name=textBoxValue>">
<input type=submit name="submit" value="OK">
</form>
<jade_tag name="footer">
</body>
</html>
```

In this HTML, another **<jade_tag>** to represent the value of the text box has been added. Saving this text adds another String property **textBoxValue**, to represent the **JADE_TAG** tag.

11. Change the **updateValues** method to populate the **textBoxValue** with the value of **textbox**, as shown in the following method.

```
updateValues(): Boolean updating;
vars
begin
    header := "This is the header";
    footer := "This is the footer";
    textBoxValue := textbox;
    return inheritMethod();
end;
```

12. Running the application now has the same behavior as before the **generateHTMLForJadeTagsOnly** property was set to **true**, while generating HTML only for the **JADE_TAG** tags.

In general, any tag that is input-output is populated only with its initial value when the HTML message is generated, regardless of whether these values are in the message or changed in code. You therefore need to consider an approach like the one shown in this example if you need this information to be sent back to the web browser.

Chapter 13 Using Method Views to Bookmark Workflows

This chapter contains the following topics.

- Overview
- Adding a Method View
- Displaying All Method Views that You Created
- Adding Methods to a Method View
- Removing a Method from a Method View
- Using the Methods View Browser
  - Opening a Method View
  - Changing a Method View
  - Copying a Method View
  - Removing a Method View
  - Extracting a Method View
  - Showing Method Views Created by All Developers

Overview

The Jade Platform development environment enables you to add or maintain method views, which group methods from any class in any schema into a named workspace that can assist you in book-marking workflows (for example, all methods and events that relate to a specific feature, with the method view description advising other developers about the functionality provided by the methods).

Method views are useful for encapsulating all methods that relate to a specific area or feature on which you are working to separate them out from the Schema Browser and the Class Browser of each schema. You can include methods from any class in any schema in a method view by dragging and dropping them into the Methods View Browser from any other source window, including another Methods View Browser.

You can browse all method views created by you or by another developer, and add methods to or maintain methods in an existing method view. As a method updated in a method view is updated in the original (source) location, any change is immediately reflected in any source window for that method when the lock is released after compilation. When you extract a method view, all methods are extracted to a partial schema file.

Note As a method view is persistent, it is available across work sessions for you or any other developer to view or maintain by adding or removing methods or updating methods in the view.

The middle section of the **Methods Viewer** submenu (accessed from the Browser menu) displays the ten most recent method views that you accessed, so that you can quickly access the one that you require.

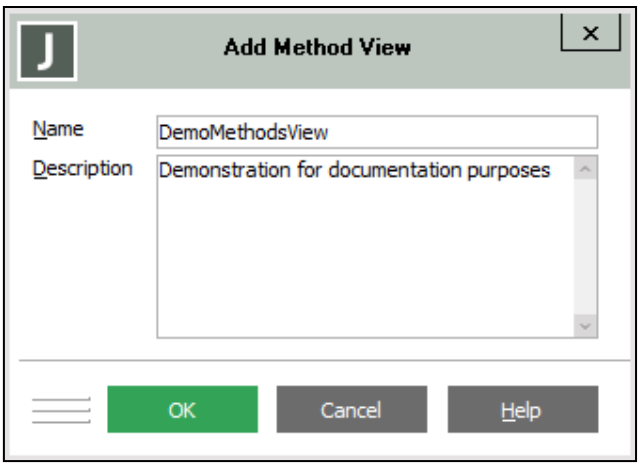
Adding a Method View

You can add a new method view from any browser other than the Primitive Types Browser.

To add a new method view

1. Select the **Methods Viewer** command from the Browser menu. A submenu is then displayed.
2. Perform one of the following actions.
   - Select the **New View** command from the Methods Viewer submenu.
   - Select the **Add** command from the **Methods Viewer** menu (accessed when the Methods View Browser has focus) or from the popup menu in the Methods View Browser.

   The Add Method View dialog, shown in the following image, is then displayed.



3. In the **Name** text box, specify the name that you require for the method view.

   The method view name, which must start with a letter, can be a maximum of 100 characters. It can include numbers and underscore characters, but cannot include punctuation or spaces. The first letter of the name is converted to an uppercase character, if it is lowercase. The name must be unique to the Jade database.
4. In the **Description** text box, specify the free-format text that describes your method view, if required (for example, to indicate to other developers the purpose of the method view).
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon the method view maintenance.

The Methods View Browser is then displayed. For details, see "Adding Methods to a Method View", in the following subsection. See also "Removing a Method from a Method View" and "Using the Methods Viewer Browser", later in this chapter.

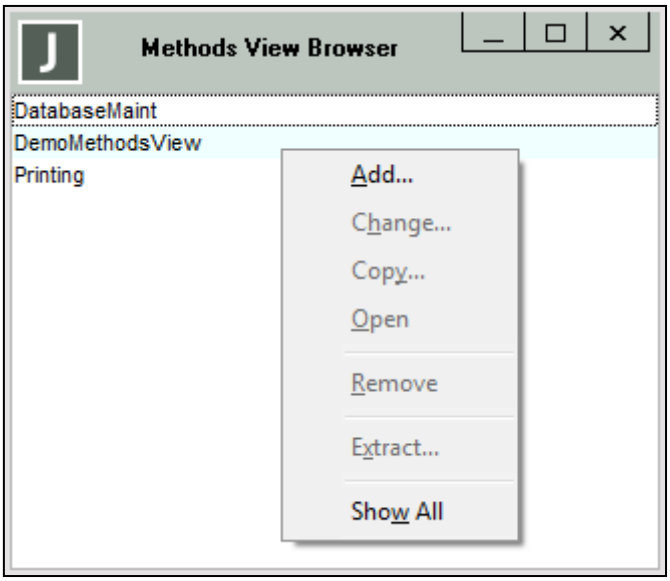
Displaying All Method Views that You Created

By default, the Methods View Browser displays only the method views that you created.

To display all method views that you created in the Methods View Browser

- Select the **Browse All** command from the Browse menu **Methods Viewer** command submenu.

The Methods View Browser, shown in the following image, is then displayed, listing all method views created by you.



For details about displaying method views created by all developers, see "Showing Method Views Created by All Developers", later in this chapter. See also "Using the Methods View Browser", later in this chapter.

Adding Methods to a Method View

The Methods Viewer window enables you or any other developer to add a method defined in any primitive type or in any class in any schema to a method view at any time. As you can copy existing methods to a method view, you cannot define a new method by using the Methods Viewer window. For details about defining a method, see "Adding Jade Methods to Classes or Primitive Types", in Chapter 4. For details about quickly accessing a specified class or method, see "Using the Quick Navigation Facility", in Chapter 2.

Tips The easiest way to add a method to a method view is to have an open source window (for example, a Class Browser) and an open Methods Viewer window positioned beside it. You then simply drag and drop the methods that you want to include in your method view from the source window to the Methods Viewer window.

You can drag and drop a specific method or all methods in a specific class or primitive type. (If a specific class has one or more subclasses, only the methods in the selected class are copied to the Methods Viewer window; that is, subclasses and their methods are not copied.)

For details about accessing the Methods Viewer window, see "Adding a Method View", in the previous section. Alternatively, perform one of the following actions.

- Select the appropriate method view from the middle section of the submenu that is displayed when you select the **Methods Viewer** command from the Browse menu. (See also "Displaying All Method Views that You Created", in the previous section.)
- Double-click on a method view listed in the Methods View Browser. (For details about this browser, see "Using the Methods View Browser", later in this chapter.)

The Methods Viewer window is then displayed.

To add methods to the Methods Viewer window, perform one of the following actions

- While holding down the Ctrl or the Shift key, select the method from the Methods List in the source window and then drag the method to the list at the left of the Methods Viewer window.

  Repeat this action for each primitive type or subclass of the selected class whose methods you want to include in the method view.

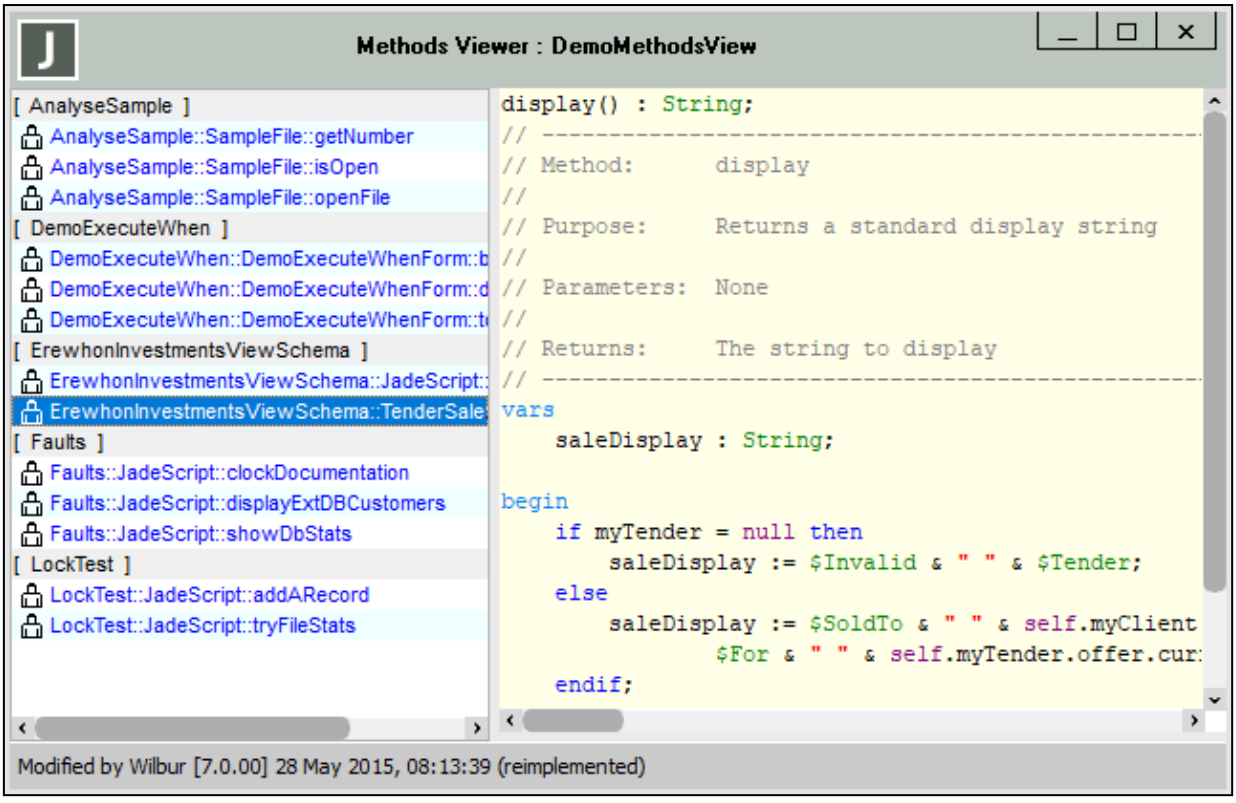
- While holding down the Ctrl or the Shift key, select the class in the Class List of the Class Browser or the primitive type in the Primitive Types Browser and then drag the class or primitive type to the list at the left of the Methods Viewer window. All methods in the selected class or primitive type are then copied to the Methods Viewer window, prefixed with the class or primitive type name (for example, **JadeScript::loadData**).

  As only methods in the selected class or primitive type are copied, you must repeat this action for each subclass of the selected class whose methods you want to include in the method view.

The position to which you drag the class, primitive type, or method is irrelevant, as all methods are grouped in the schema in which they are defined.

Note You can drag an external key dictionary class only if there are no items in the dictionary.

Schemas are displayed with a light gray background and are enclosed in bracket symbols (**[ ]**) symbols; for example, **[LockTest]**, as shown in the following image.



The popup (context) menu, displayed when you right-click in the editor pane, provides standard edit and search functions. When you update a method in a method view, the method is updated in the original (source) location and any change is immediately reflected in any source window for that method when the lock is released after compilation.

For details about maintaining methods, see "Defining and Compiling Jade Methods and Conditions", in Chapter 4. See also "Locating Text on Which the Caret is Positioned" and "Searching for an Element in all Classes in the Current Schema", in Chapter 3.

Tip To open a standalone editor pane for a method, double-click on the method in the list at the left of the Methods Viewer window.

For details about removing a method from the Methods Viewer window or extracting a method view, see "Removing a Method from a Method View" or "Extracting a Method View", later in this chapter.

Removing a Method from a Method View

As a method removed from a method view is removed only from the method view, it is *not* deleted from the class in which it is defined and it remains in the Methods List of the Class Browser. (For details about removing a method from the Jade database, see "Removing a Schema Element", in Chapter 3.)

You can remove a method from a method view that you created or that was created by another developer.

To remove a method from a method view

1. In the Methods Viewer window, perform one of the following actions.
   - If you want to remove a single method from the method view, select the method from the list at the left of the window.

     Repeat this action for all methods that you want to remove from the method view.
   - As the methods list enables multiple-selection, use the Shift or Ctrl key to select a group or range of methods.
2. Right-click on the selected method or methods in the list at the left of the Methods Viewer window and then select the **Remove From List** command from the popup menu that is displayed.

   Alternatively, select the **Remove From List** command from the Methods menu.

The selected method is then removed from the method view and it is no longer displayed in the Methods Viewer window (but it remains in the source window; for example, the Methods List of the Class Browser).

For details about removing a method view, see "Removing a Method View", later in this chapter.

Using the Methods View Browser

To access the Methods View Browser

- Select the **Browse All** command from the Browse menu **Methods Viewer** command submenu.

The Methods View Browser is then displayed, listing only the method views that you created.

The Methods View Browser enables you to perform the actions listed in the following table.

Action	For details, see…
Add	Adding a Method View
Change	Changing a Method View

Action	For details, see…
Copy	Copying a Method View
Open	Opening a Method View
Remove	Removing a Method View
Extract	Extracting a Method View
Show All	Showing Method Views Created by All Developers

To perform a maintenance action on a method view created by another developer, you may first have to display the method views created by all developers. For details, see "Showing Method Views Created by All Developers", later in this chapter.

For details about highlighting a method in the methods list (for example, as a reminder that more work on that method is required), see "Highlighting Methods in Lists of Methods", in Chapter 4.

Opening a Method View

To open the selected methods view, perform one of the following actions

- Double-click on the method view in the Methods View Browser.
- In the Methods View Browser, select the method view that you want to open and then select the **Open** command from the Methods View menu.
- In the Methods View Browser, right-click on the method view that you want to copy and then select the **Open** command from the popup menu that is displayed.

The methods view is then displayed in the Methods Viewer.

Changing a Method View

You can change a method view that you created or one that was created by another developer.

To change an existing method view

1. Perform one of the following actions.
   - In the Methods View Browser, select the method view that you want to change and then select the **Change** command from the Methods View menu.
   - In the Methods View Browser, right-click on the method view that you want to change and then select the **Change** command from the popup menu that is displayed.

   The Change Method View dialog is then displayed.
2. In the **Name** text box, change the name of the method view, if required.
3. In the **Description** text box, change the free-format text that describes the method view, if required (for example, to indicate to other developers the purpose of the method view).
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon the method view maintenance.

The Methods View Browser is then updated to reflect the change action.

Copying a Method View

You can copy an existing method view to a new method view that is similar to the method workflow that you require and then add methods to or remove methods from the copied and renamed method view.

To copy an existing method view created by you or another developer

1. Perform one of the following actions.
   - In the Methods View Browser, select the method view that you want to copy and then select the **Copy** command from the Methods View menu.
   - In the Methods View Browser, right-click on the method view that you want to copy and then select the **Copy** command from the popup menu that is displayed.

   The Copy Method View As dialog is then displayed.
2. In the **Name** text box, specify the name that you require for the copied method view. The name must be unique to the Jade database.
3. In the **Description** text box, change the free-format text that describes the copied method view, if required.
4. Click the **OK** button. Alternatively, click the **Cancel** button to abandon the method view copy action.

The method view copy is then displayed Methods View Browser. For details about adding methods to or moving methods from the copied method view, see "Adding Methods to a Method View" or "Removing a Method from a Method View", earlier in this chapter.

Removing a Method View

You can remove a method view that you created or one that was created by another developer.

To remove an existing method view

1. Perform one of the following actions.
   - In the Methods View Browser, select the method view that you want to remove and then select the **Remove** command from the Methods View menu.
   - In the Methods View Browser, right-click on the method view that you want to remove and then select the **Remove** command from the popup menu that is displayed.
2. If the method view was created by another developer, the Delete Confirmation message box is then displayed, prompting you to confirm that you want to remove the method view created by the specified developer.

   Click the **OK** button to confirm the deletion or the **Cancel** button to cancel the deletion.

The method view is then removed from the Jade database and is no longer displayed in the Methods View browser. (The methods that were contained in the deleted method view remain in the class in which they were defined, however.)

Extracting a Method View

Although a method view is not extracted as part of a standard schema extraction, you can extract methods added to a method view that you created or one that was created by another developer.

Note A method view schema extract file contains only the methods added to that view. As no information regarding the method view itself is extracted, loading a method view schema extract file into a new Jade database does not recreate the method view itself but loads the methods that were added to the originating method view.

To extract methods in a method view

1. Perform one of the following actions.

   - In the Methods View Browser, select the method view whose methods you want to extract and then select the **Extract** command from the Methods View menu.
   - In the Methods View Browser, right-click on the method view whose methods you want to extract and then select the **Extract** command from the popup menu that is displayed.

   The Extract dialog is then displayed, with all options other than the selected **Multiple Schemas** option button disabled.

   The file name defaults to the name of the current schema, with a .**mul** suffix.

   The syntax of the multiple schema file for methods extracted from a method view is as follows.

   ```
   #MULTIPLE_SCHEMA_EXTRACT
   schema-name_method-view-name.scm schema-name_method-view-name.ddb|ddx
   schema-name_method-view-name.scm schema-name_method-view-name.ddb|ddx
   schema-name_method-view-name.scm schema-name_method-view-name.ddb|ddx
   ...
   ```

   The following is an example of a method view multiple schema extract file.

   ```
   #MULTIPLE_SCHEMA_EXTRACT
   NoteTest_MyMethodView.scm NoteTest_MyMethodView.ddb
   PrintTest_MyMethodView.scm PrintTest_MyMethodView.ddb
   SortTest_MyMethodView.scm SortTest_MyMethodView.ddb
   TestSchema_MyMethodView.scm TestSchema_MyMethodView.ddb
   ```

   The methods in each schema in your methods view are extracted to a separate pair of .**scm** and .**ddb** or .**ddx**, depending on the value of the **Extract as DDX (xml format) instead of DDB** check box on the **Source Management** sheet of the Preferences dialog, for each schema. The multiple schemas extract file itself contains merely a list of these file names. For details about multiple schema extract files, see "Extracting Multiple Schemas", in Chapter 10.

2. In the **Multi Extract File Name** text box, specify the name and location of the methods view multiple schemas file you want to extract.

   If you do not specify a location, the file is extracted to your Jade working directory (for example, **c:\jade\bin\MyMethodView.mul**).

   If you want to extract the methods in the methods view to an existing multiple schemas file or you are unsure of existing file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   When you extract a method view, all methods in that view are extracted to a partial schema file. You cannot encrypt the saved methods view in the multiple extract file.

   Note If the multiple extract file does not have the same absolute path as the schema and forms definition files containing the class or classes in which the methods are defined, a load error is encountered when you attempt to load the extracted methods view into your Jade database.

3. Click the **OK** button to continue with the method view extraction process. Alternatively, click the **Cancel** button to abandon the extraction.

Showing Method Views Created by All Developers

By default, the Methods View Browser displays only the method views that you created.

To display the method views created by you and all other developers

- Perform one of the following actions.
  - In the Methods View Browser, select the **Show All** command from the Methods View menu.
  - Right-click in the Methods View Browser and then select the **Show All** command from the popup menu that is displayed.

All method views created by all developers are then displayed in the Method View Browser and a check mark symbol (✓) is displayed to the left of the command in the Methods View menu, indicating that all method views are currently displayed.

Use the **Show All** command to toggle the display of your own method views or all method views to meet your requirements. Alternatively, use the **Browse All** command from the Browse menu **Methods Viewer** command submenu to display only the method views created by you.

For details, see "Displaying All Method Views that You Created", earlier in this chapter.

Chapter 14 Adding and Maintaining Interfaces

This chapter covers the following topics.

- Overview
  - Jade Interface Implementation
  - Benefits of Using an Interface
- Accessing the Interface Browser
- Defining an Interface
- Maintaining an Interface
  - Adding Interface Methods
  - Compiling an Interface Method
  - Specifying Text for an Interface
  - Viewing the Implementation of an Interface Method
  - Displaying the History of an Interface
  - Viewing Current Interface Mappings
  - Viewing Interface Method Mappings
- Locating an Interface
- Displaying Interface Mappings
- Displaying all References to the Selected Interface
- Implementing an Interface
  - Updating an Existing Interface Method Implementation
  - Stopping the Implementation of an Interface
- Using Interfaces in your Applications
  - Code Re-Factoring
  - Frameworks and Packages
  - New System Development
  - Interface-Related Methods and Properties
- Printing an Interface

- Extracting an Interface
- Removing an Interface

Overview

An *interface* is a mechanism that provides a set of methods that are guaranteed to be available on any implementing class. When a class implements an interface, it agrees to implement *all* of the methods as defined by the interface. With this contract in place, the implementing classes can participate in useful type-safe callback mechanisms.

Exposing a class via its interface effectively hides the implementation details of the class, as the user is forced to communicate *only* through the public interface methods.

Classes that implement an interface can be grouped by that interface type; for example, a collection can have a membership of a specified interface.

Using interfaces, non-related classes can be grouped to capture their similarities, without the need to artificially force a class relationship. Think of this as allowing a class to perform multiple roles outside of the role dictated by its class hierarchy.

When a class is selected in the Class List of the Hierarchy Browser, the editor pane displays any inherited interfaces within the schema and superschemas and any interfaces implemented in subschemas. If the class does not implement any interfaces, none of the following sections are displayed.

- Implements the following interfaces:

  schema-name::*interface-name*

- Inherits the following interfaces:

  schema-name::*interface-name* from *schema-name*::*class-name*

- Subschema implemented interfaces:

  schema-name::*class-name* implements *schema-name*::*interface-name*

- Superschema implemented interfaces:

  schema-name::*class-name* implements *schema-name*::*interface-name*

Jade Interface Implementation

Jade achieves this by the following mechanism.

- Creates an interface, which defines a set of constant and method definitions. The defined interface methods act as a communication protocol for accessing implementing classes.
- A class implements an interface, by agreeing to implement all of the interfaces methods. The implementing class is not required to belong to any predefined class hierarchy.
- To implement an interface, a class makes an agreement to implement *all* of the methods defined by the interface (thereby establishing a communication protocol that can be used to interact with all implementing classes).

Benefits of Using an Interface

The benefits of using an interface in your application are as follows.

- The ability to define behavior that can be implemented by a group of unrelated classes without forcing them to share a common class hierarchy
- By implementing an interface, a class can perform in a role that is different from that dictated by its class hierarchy
- As a class is not limited to the number of interfaces that it can implement, it can therefore participate in a wide variety of roles
- A class can be exposed through its interface, effectively hiding its implementation details by forcing all communication to be through its public interface methods
- The class hierarchy can be simplified when developing with interfaces, which typically reduces overloading
- You can easily modify existing systems to provide new functionality within their current class hierarchy

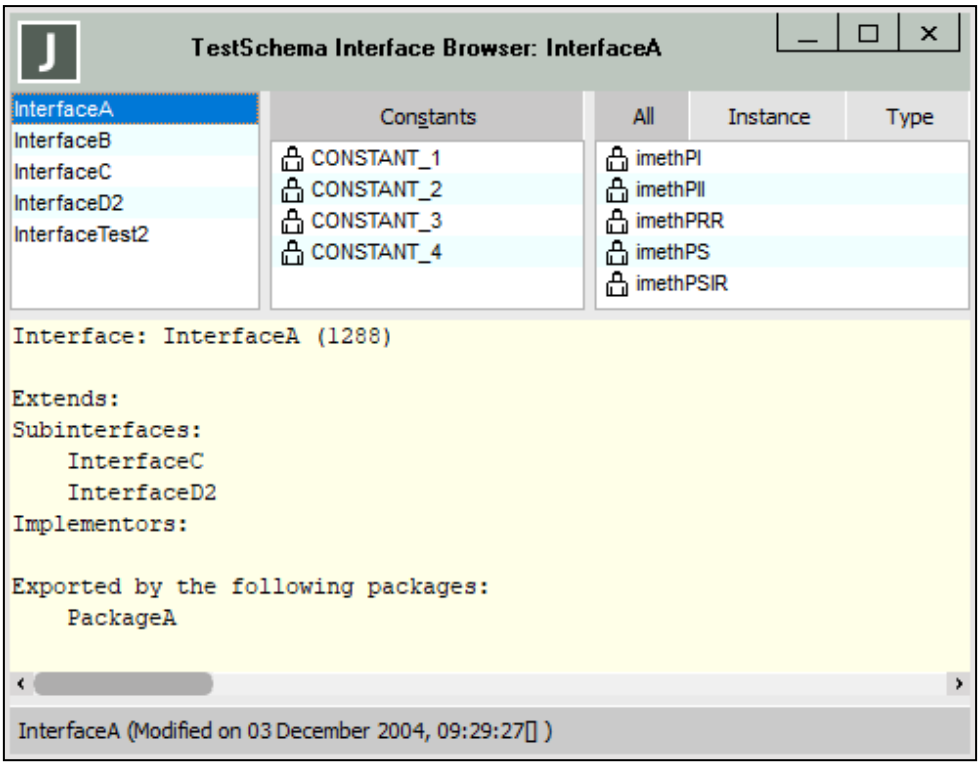
Accessing the Interface Browser

The Interface Browser enables you to add and maintain interfaces in the current schema. Each schema in the Jade database can have a collection of interfaces.

» To open an Interface Browser, perform one of the following actions

- Click the **Browse Interfaces** button on the Browser toolbar
- Select the **Interfaces** command from the Browse menu

An Interface Browser window is then opened and the Interfaces menu is displayed in the menu bar, as shown in the following image.



If you have not yet defined an interface in the current schema, nothing is displayed in the Interface Browser. When you select an interface, the editor pane displays all implementors in the current schema and all of its subschemas.

By default, the displayed interfaces are those defined in this schema and interfaces that are implemented in this schema. To view interfaces in superschemas, select the **Superschemas** command from the View menu.

The Interfaces menu contains the commands listed in the following table.

Command	For details, see…	Action
Add	Defining an Interface	Displays the Define Interface dialog
Extract	Extracting an Interface	Displays the standard Save As dialog
Compile	Compiling an Interface Method	Compiles the current method
Methods Browser	Using the Quick Navigation Facility, in Chapter 2	Displays the Methods Browser for the selected interface
References	Displaying all References to the Selected Interface	Displays the References Browser
Show History	Displaying the History of an Interface	Displays the Summary of Patches dialog
Change	Maintaining an Interface	Displays the Define Interface dialog
Find	Locating an Interface	Displays the Find Type dialog
Remove	Removing an Interface	Removes the selected interface
Text	Specifying Text for an Interface	Displays the Text window

The Interface Browser is similar to the Class Browser, with the following differences.

- An interface can define only methods and constants
- An interface method does not contain a body
- Interfaces are always displayed in alphabetical order
- An interface can extend multiple other interfaces
- The View menu **Show Inherited** command displays details of methods and constants from any interface defined in a superschema of the current schema

For details about the Class Browser, see "Opening a Class, Primitive Types, or Interface Browser", in Chapter 3.

Tip Use the context (popup) menu to access menu items that relate to interfaces; for example, the Delta menu that enables you to check a selected method out of or into a delta, compare method sources, and undo a method check-out.

Toggling the Display of Entities in Imported Interfaces

When the current window is the Interface Browser, the View menu contains the command listed in the following table.

Command	Description
Show Imported Interfaces	Toggles the display of interfaces (and their constants and methods, when applicable) imported into the current schema

A check mark is displayed to the left of the command in the View menu when imported interfaces are displayed, which is the default display.

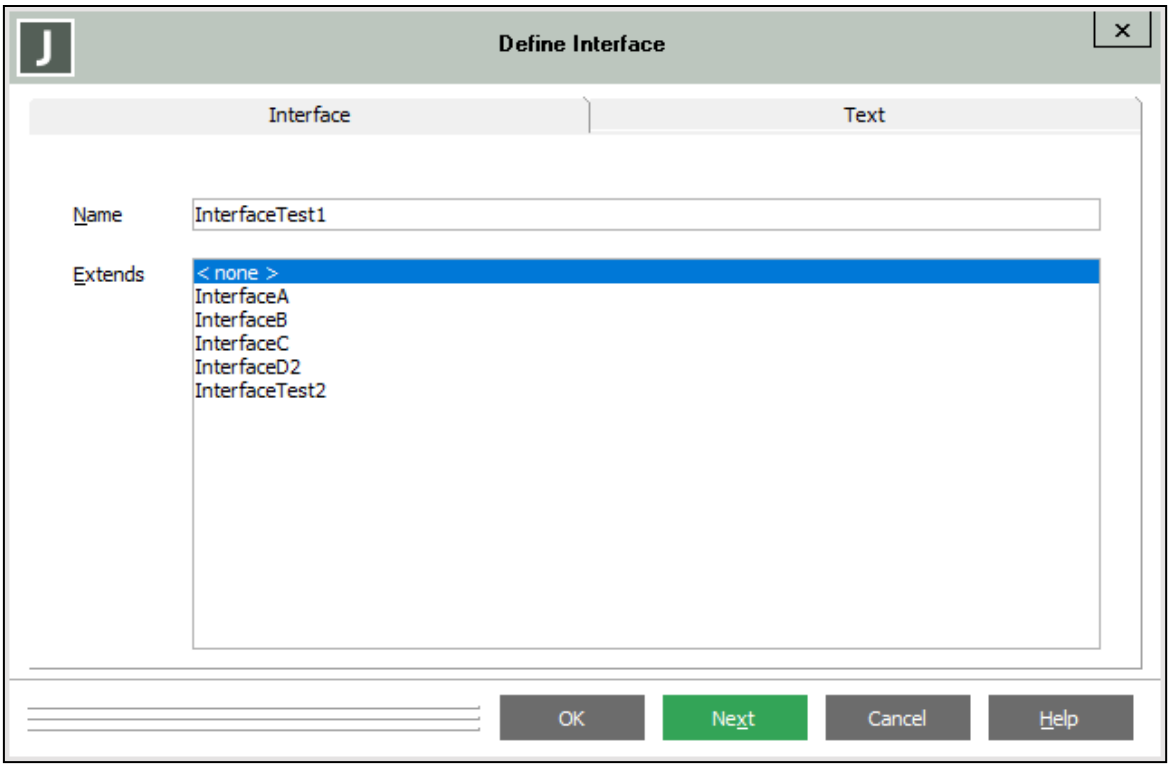
Interfaces in imported packages are displayed in the Interface List by default.

Defining an Interface

To add a new interface to the current schema

1. Select the **Add** command from the Interfaces menu.

   The Define Interface dialog, shown in the following example, is then displayed.



2. In the **Name** text box, specify the name of the interface that you want to define. The name must be unique within the schema to which it is being added.

3. If you want the interface to extend any existing interface that exists in the current schema or superschema, select one or more interfaces that you want to extend from the list displayed in the **Extends** list box. As no interface is extended by default, the **< none >** value is selected by default.

   As an interface can extend multiple other interfaces, you can select multiple interfaces in the **Extends** list, if required, by using the Ctrl key and clicking each interface that you want to extend.

4. If you want to enter free-format descriptive text for the interface, select the **Text** sheet. The Define Interface dialog **Text** sheet is then displayed.

5. In the editor pane, enter the descriptive text for your interface.

Note You can also enter supplementary text, which is for display only, by selecting the **Text** command from the Interfaces menu. The **Text** command accesses a freestanding window rather than a separate sheet of the Define Interface dialog. For more details about entering interface text by using the freestanding editor window, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3. See also "Specifying Text for a Schema Element", in Chapter 3.

The editor determines the behavior of the text editor window. Text is *not* automatically wrapped to fit the current window size. When you want text to start in the next line, press Ctrl+Enter. (If you want text to wrap in text and source windows, see "Maintaining Editor Options", in Chapter 2.)

6. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your class selections or the **Next** button to define another interface.

The specified interface is then displayed in the Interface Browser and the descriptive text, if any, is displayed at the end of the interface details in the integrated Interface Browser editor pane when the interface is selected in the Interface List. It is also printed beneath the interface name when the interface is printed (by selecting the **Print Selected** command from the File menu) and the **Text** check box in the Print dialog is checked.

You can now add constants and methods to an Interface in the same manner that a constant or method is added to a class or primitive type. For details, see "Defining Class, Primitive Type, and Interface Constants" and "Adding Jade Methods to Classes, Primitive Types, or Interfaces", in Chapter 4. See also "Adding Interface Methods", later in this chapter. For details about defining and using interface constants, see "Adding a Class, Primitive Type, or Interface Constant" and "Using Class, Primitive Type, or Interface Constants in Your Methods", in Chapter 3.

Maintaining an Interface

This section contains the following topics.

- Adding Interface Methods
- Compiling an Interface Method
- Specifying Text for an Interface
- Viewing the Implementation of an Interface Method
- Displaying the History of an Interface
- Viewing Current Interface Mappings
- Viewing Interface Method Mappings

For details, see the following subsections.

Adding Interface Methods

An interface method consists of a signature only (that is, it contains no method body). When a class implements an interface, the interface methods are implemented by the class as if they were normal Jade methods, although it is your responsibility to provide the required method sources that make the method implementations useful.

When adding methods to an interface:

- If a new method is added to a currently implemented interface, a new method stub is automatically generated in the implementing classes, as shown in the following image. (For details, see "Implementing an Interface", later in this chapter.)

If you did not specify that you require a reminder that generated methods are purely a shell without any code and that you have not yet provided the appropriate method implementation details, the new class method is generated with the text block at the beginning of the stub method body commented out, as shown in the following image.

```
imethNew();

    /*
        this method stub has been automatically generated by Jade to
        satisfy interface implementation requirements for the InterfaceB interface
    */

vars

begin

end;
```

If you want a compile-time error generated when the stub method is created (that is, it is marked in error in the Methods List of the Class Browser), check the check box in the Interface Options group box on the **Miscellaneous** sheet of the Preferences dialog. (For details, see "Maintaining Miscellaneous Options", in Chapter 2.) Stub methods are then automatically generated with body details that you must comment out or remove before the method compiles, as a way of highlighting that you have yet to provide the required method body. For example, you can locate these empty methods by displaying methods in error, to find the methods that you are likely to want to flesh out to make the implementation useful.

To compile the method, delete or comment out the text line.

- You can view current mappings of a selected method. For details, see "Viewing Current Interface Mappings", later in this chapter.
- Interface methods do not provide any implementation details. A compile-time error is raised if the method contains body details.

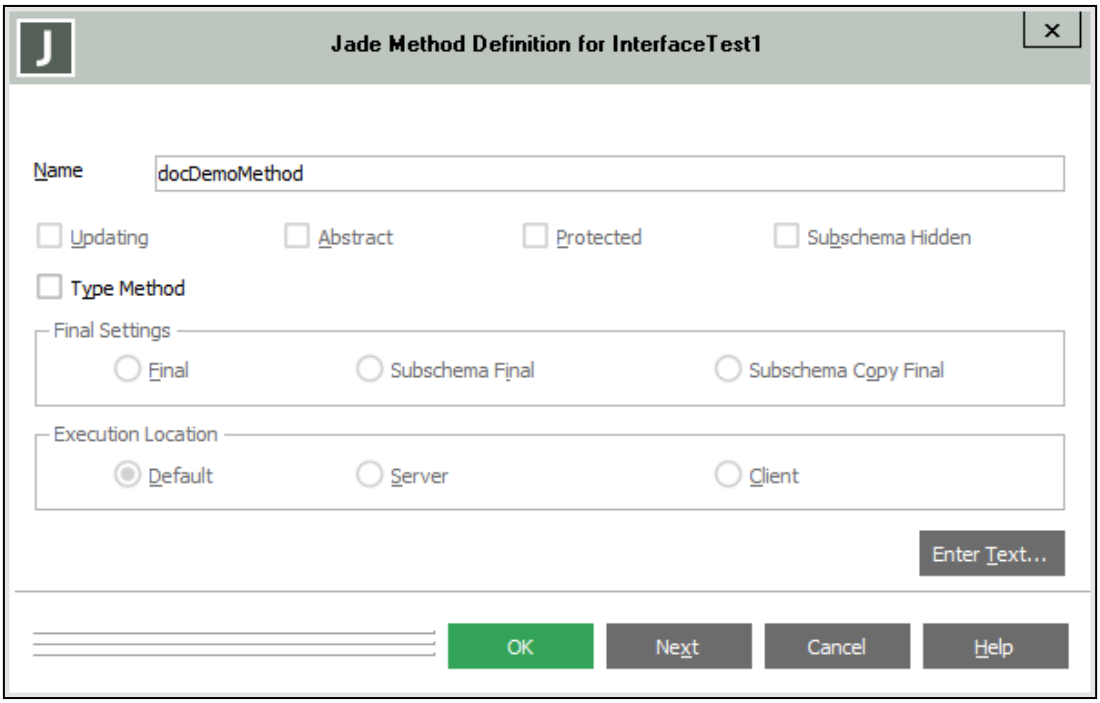
For details about obtaining bubble help (for example, to locate and insert the required constant or property for the interface), see "Using Bubble Help in the Editor Pane", in Chapter 4.

Note Before you define a method, you must select the interface to which the method is to be added.

To add a method to an interface

1. In the Interface List of the Interface Browser, click on the interface for which the method is to be added.
2. Select the **New Jade Method** command from the Methods menu.

The Jade Method Definition dialog, shown in the following image, is then displayed.



Note As interface methods can contain only a method signature, controls that do not apply to interface methods are disabled.

3. Specify the name of your new method in the **Name** text box. The name value must start with a lowercase character and the name must be unique to the interface to which it is being added.

4. Click the **Enter Text** button if you want to specify or maintain descriptive text for the Jade method as part of the definition or maintenance of the method. For details, see "Specifying Text for a Schema Element", in Chapter 3.

   Tip You can also specify descriptive text for the Jade method at any time, by selecting the **Text** command from the Methods menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

5. Click the **OK** button or the **Next** button.

The method signature is then displayed in the editor pane of the Interface Browser.

Compiling an Interface Method

You can change interface method signatures at any time. If the interface is currently implemented, the mapping method for the implementing class is not compatible and therefore has a compilation error. See also "Using the Method Status List Browser", in Chapter 4.

When you change the signature of an interface method that has one or more mappings, the Compile Method message box is then displayed, stating that the method signature has changed, one or more references must be compiled, and prompting you to confirm that you want to compile now.

Click the **Yes** button to recompile the mapped method or methods immediately. Click the **No** button if you do not want the implementing method or methods recompiled immediately.

You must then change the signature of affected methods and then recompile them (by pressing F8 in the editor pane of each method implementor in the Class Browser or Status List Browser or by selecting the **Compile** command from the Methods menu).

If the interface method signature change results in a compile error in the implementing class, a message box is then displayed, advising you that the compile process has one or more errors, and prompting you to specify that you want to view the error list.

Click the **Yes** button to display the methods in implementing classes whose signatures need updating before recompilation. Click the **No** button if you do not want to view and update the methods in error now.

Tip To automatically update the class method signature in the method in error (from the Class Browser) so that it matches that of its interface method, position the caret after the **(** opening parenthesis symbol and then press Ctrl+1, Ctrl+2, or Ctrl+3.

The correct signature for the mapped method is then inserted.

For more details about compiling methods, see "Compiling All Methods Defined in a Class, Primitive Type, or Interface", in Chapter 3, and "Compiling Methods", in Chapter 4.

Specifying Text for an Interface

Although you can add or maintain text for an interface at the time that you add or maintain it (by using the **Text** sheet on the Define Interface dialog), you can also specify descriptive text for the interface at any time, by selecting the **Text** command from the Interfaces menu. For details, see "Using the Free-Standing Editor Window to Define Text", in Chapter 3.

For details about specifying text during the definition of an interface, see "Defining an Interface", earlier in this chapter.

The specified or updated text is then displayed at the end of the interface details in the integrated Interface Browser editor pane when the interface is selected in the Interface List. It is also printed beneath the interface name when the interface is printed (by selecting the **Print Selected** command from the File menu) and the **Text** check box in the Print dialog is checked (the default value).

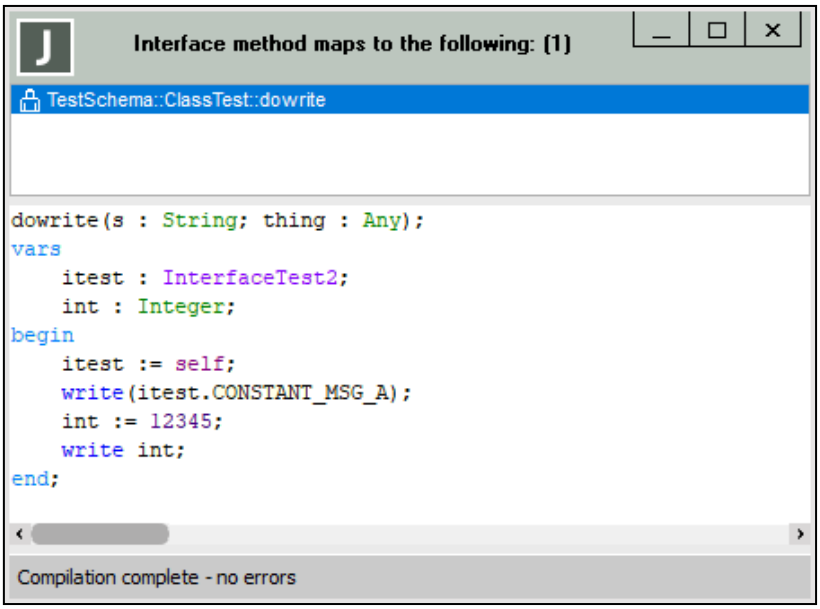
Viewing the Implementation of an Interface Method

You can view the implementation of an interface method selected in the Interface Browser.

» To display the implementation of an interface method

1. In the Interface Browser, select the interface in which the required method is defined.
2. Select the method in the Methods List for that interface.
3. Select the **Method Implemented By** command from the Methods menu.

The browser shown in the following image is then displayed.



The title of the browser window states in parentheses the number of class methods to which the interface method maps.

You can amend and compile a selected method, if required; for example, to automatically update the method signature when the signature of the interface method has changed. (For details, see "Compiling an Interface Method", earlier in this chapter.)

To display the interface methods to which a class method is mapped, see "Displaying Interface Methods to which the Selected Method is Mapped", in Chapter 4.

Displaying the History of an Interface

You can display a summary of the changes made to the interface selected in the Interface Browser.

Note You cannot display a patch history summary for an interface unless patch versioning is enabled for your current schema or for the Jade Platform development environment.

For details, see "Enabling or Disabling Patch Versioning", in Chapter 3 of the *Development Environment Administration Guide*.

To display a summary of patch version changes to the current interface

- Select the **Show History** command from the Interfaces menu.

The Summary of Patches window is then displayed, showing all changes to the current interface in all Jade system and patch versions (sorted in descending order of time, by default).

For details, see "Displaying a Patch History for a Selected Class, Type, Interface, or Method" and "Using the Summary of Patches Window", in Chapter 3 of the *Development Environment Administration Guide*.

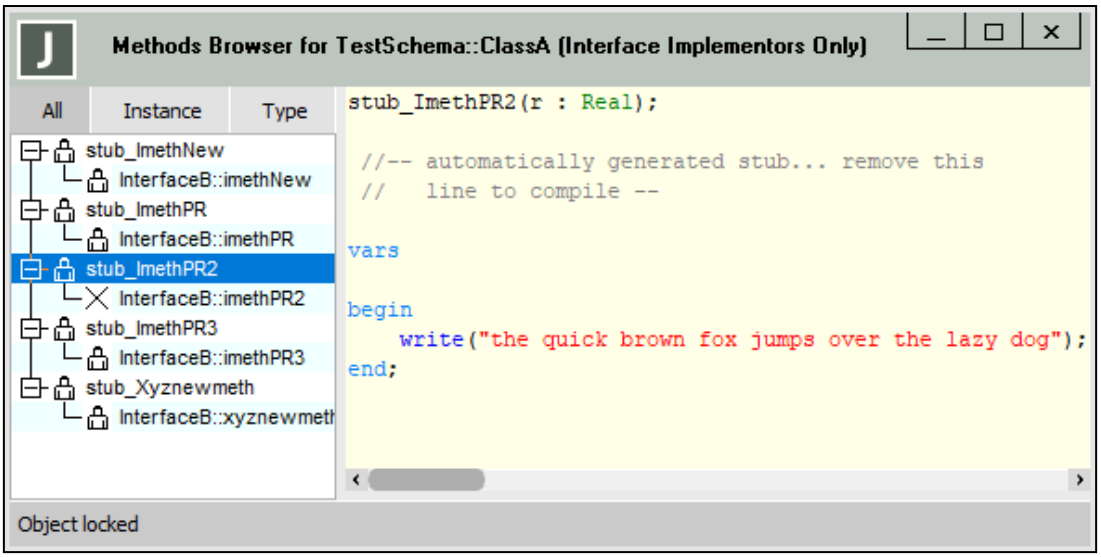
Viewing Current Interface Mappings

You can map methods defined on classes to multiple interface methods from the same interface or from different interfaces. You can view the scope of mapping from the Methods Browser, by displaying only the methods that are involved in an interface mapping.

To display the scope of local method mappings

1. In the Class Browser, select the class whose interface mappings you want to view.
2. Select the **Methods Browser** command from the Classes menu. The Methods Browser for the selected class is then displayed.
3. Select the **Show Interface Implementors Only** command from the View menu.

The Methods List at the left of the Methods Browser then displays the scope of the local method mappings in the current schema, as shown in the following image.



Although you can change a class method in the editor pane of this browser, to change an interface method displayed in the Methods List of the Methods Browser for interface implementors only, you can do so only from the Interface Browser.

See also "Displaying Interface Mappings", elsewhere in this chapter.

Viewing Interface Method Mappings

Although all interface methods implemented by a class selected in the Class List of the Class Browser are displayed in the Methods List of the Class Browser, you can display the implemented methods grouped by the interface or interfaces in which they are defined.

This can be useful if your class contains many methods and implements a number of interfaces, particularly when the same implemented method can be defined in one or more interfaces.

By default, implemented interface methods are not grouped by interface.

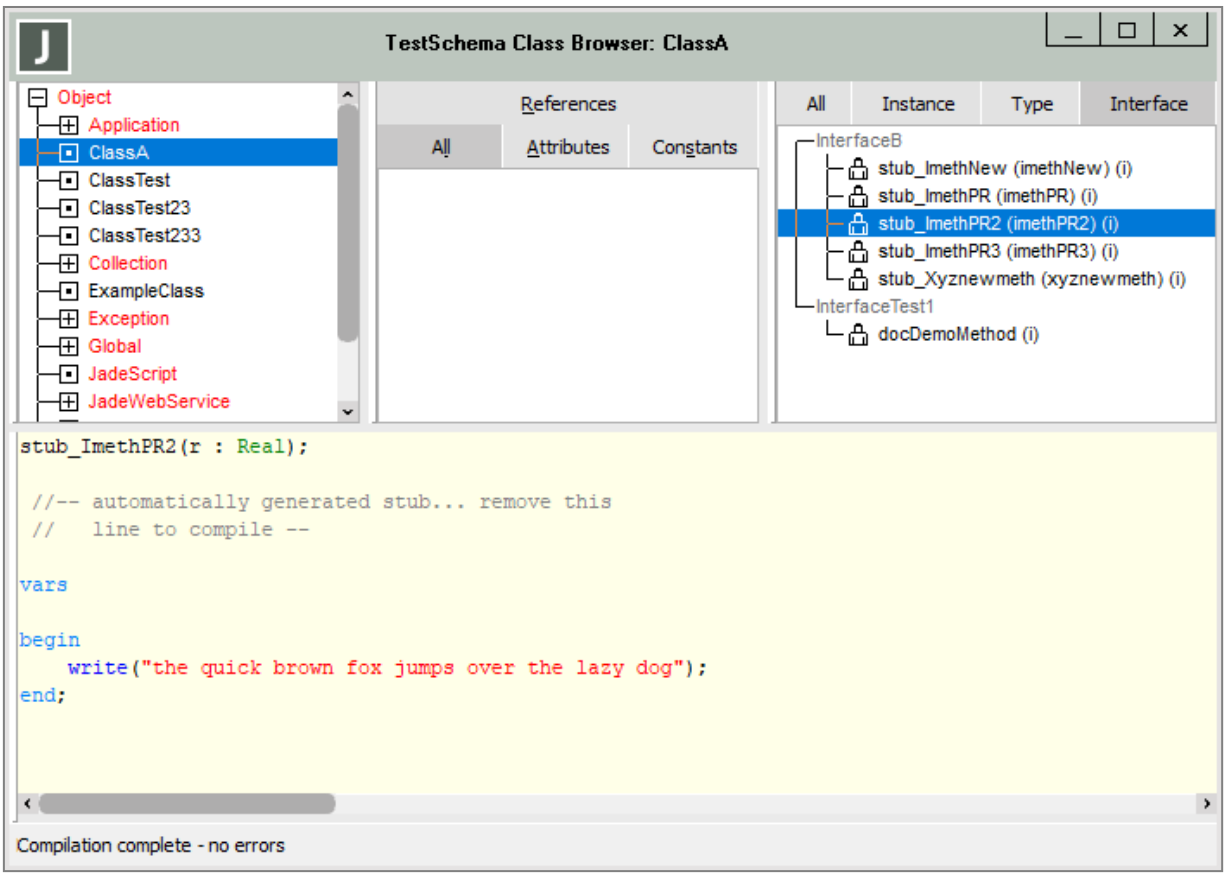
To view all implemented interface mappings

- Select the **Interface** folder in the Methods List window in the hierarchy Class Browser.

The Methods list then displays all implemented interfaces and mapped methods as a two-level hierarchy. The top level displays the names of all interfaces implemented on the selected class and the second level displays all of their implemented methods.

If a method has multiple interface mappings, it displays the method under each of the mapped interfaces. If the implemented method name differs from the interface method name, the interface method name is displayed in parentheses after the implemented methods name.

The following image shows the implemented methods listed below the interface or interfaces in which they are defined.



To group implemented methods into interfaces in which they are defined for all Methods List folders

- Select the **Show Interface Method Mappings** command from the View menu.

The Methods List then displays all implemented methods and methods defined in a selected class, followed by each interface name and their implemented methods. A check mark is displayed at the left of the command in the View menu to indicate that interface methods mappings are shown.

Note Selecting the **Show Interface Method Mappings** command from the View menu does not change what is displayed when viewing the **Interface** method list folder, as it already displays this information.

For details about implementing an interface for a class, see "Implementing an Interface", later in this chapter. See also "Displaying Interface Mappings", later in this chapter.

Locating an Interface

The **Find** command is useful when you:

- Have more interfaces in a schema than are displayed in the Interface List window of the Interface Browser. (The search automatically locates the specified interface.)
- Want to locate an interface provided by a superschema; for example, the **JadeMultiWorkerTcpTransportIF** class of the **RootSchema**.

» To locate an interface

- Perform one of the following actions from the Interface Browser.
  - Press F4.
  - Select the **Find** command from the Interfaces menu.

The Find Type dialog is then displayed. Interfaces listed in green indicate a class or interface imported into the schema in a package. For details, see Chapter 8, "Using Packages", in the *Developer's Reference*.

For details about using the Find Type dialog to locate the interface that you require, see "Finding a Schema, Class, Interface, or Primitive Type", in Chapter 3.

Displaying Interface Mappings

If you navigate through a series of methods (for example, using Shift+F11) and you reach a method defined on an interface, you can get a list of the classes and methods that implement that interface.

» To display interface mappings, select the

- **Interface Mappings** menu command in the Methods menu of a hierarchy browser to display all interface methods that the method implements and the entire list of methods that implement each interface method.

  This shows each interface method at level 1 in the list box and all methods that implement the interface underneath at level 2 (sorted by name) with drawn tree levels and icons; for example:

```
RootSchema::JadeMultiWorkerTcpTransportIF::connectionEvent (interface method
and the following methods that implement it)
     RootSchema::JadeOdbcServer::mwtt_ConnectionEvent
     RootSchema::JadeWebAppExtension::mwtt_connectionEvent
```

- **Method Implemented by** command in the Methods menu from the Interfaces Browser to display the same information as that in the previous item of this list.
- **Interface Mappings** command from the Methods menu for all Method windows, including all Shift+F11 method lists.

  Note This command is enabled only when the currently selected method implements an interface or it is a an interface method.

See also "Viewing Current Interface Mappings" and "Viewing Interface Method Mappings", earlier in this chapter.

Applies to Version: 2020.0.01 and higher

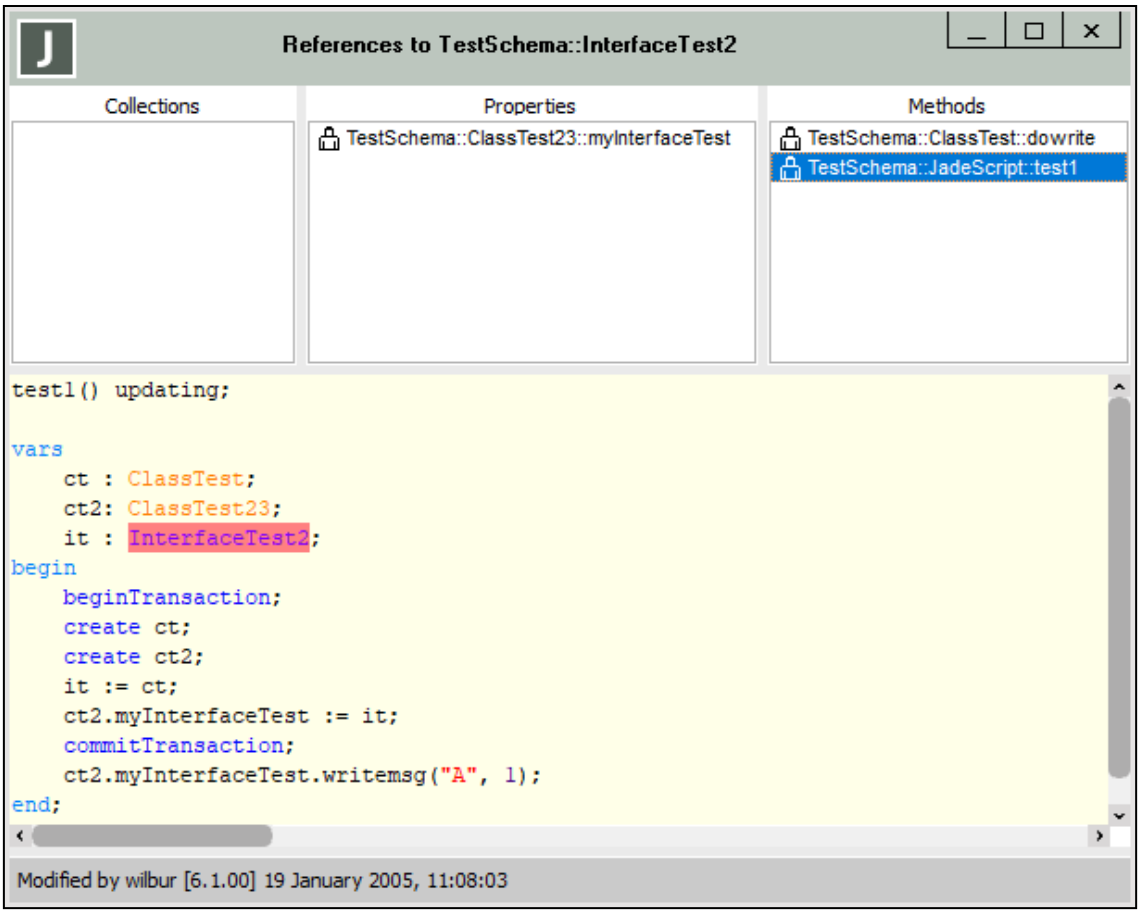
Displaying all References to the Selected Interface

The References window for a selected interface lists all collections, properties, and methods that reference the selected interface, and enables you to view specific references to the interface.

To display references for the interface selected in the Interface Browser

- Select the **References** command from the Interfaces menu.

All references to the interfaces selected in the Interface List of the Interface Browser are then displayed, as shown in the following image.



This window lists all collections, properties, and methods that reference the selected interface, and enables you to view specific references to that interface.

To view a reference to an interface

- In the References window, select the collection, property, or method whose reference to the selected interface you want to view.

The details of the selected collection, property, or method are then displayed in the editor pane.

You can maintain a method displayed in the editor pane of the References window.

Implementing an Interface

For a class to implement an interface, it must:

1. Indicate that it wants to use a specific interface
2. Satisfy the interface requirements by implementing *all* of the interfaces methods

Note For a class to implement an interface, it enters into a contract to provide a mapping between each method in the interface and a method from the class. This does not have to be a one-to-one relationship, as one class method can map to multiple interface methods for a specific interface or to one or more methods in multiple interfaces.

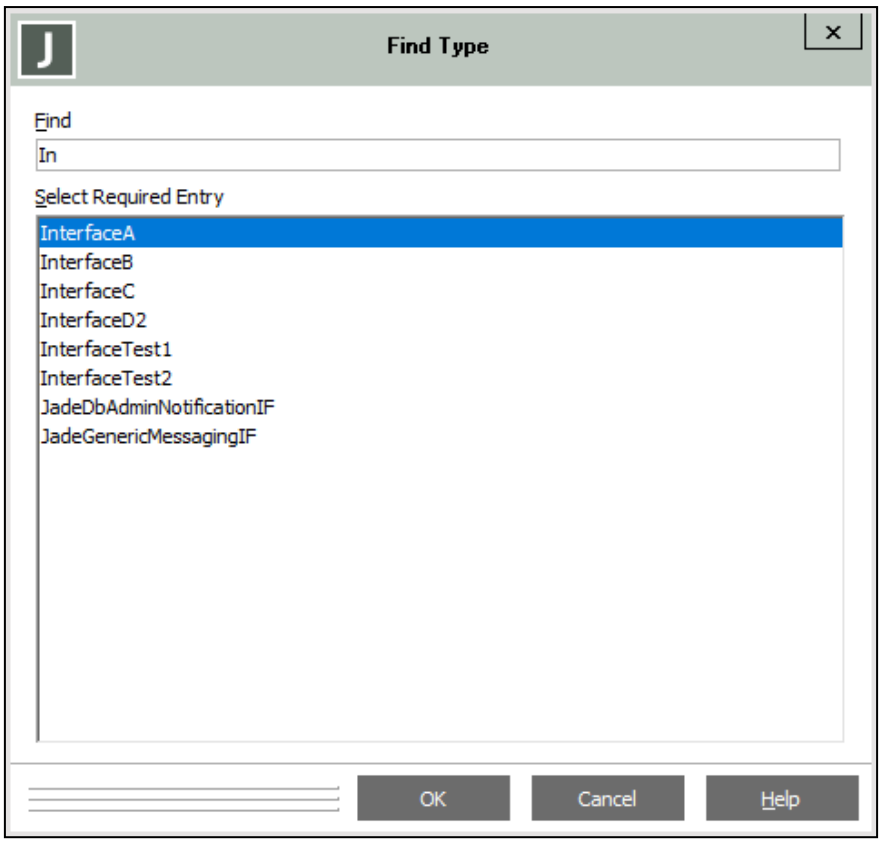
The only requirement is that the method signatures are compatible.

To implement an interface

1. In the Class List of the Class Browser, select the class that is to use the interface.
2. Select the **Interface Mapping** command from the Classes menu.

   The Interface Implementation Mapper dialog is then displayed.
3. In the **Available Interfaces** list box, select the interface that you want to implement. Alternatively, if you want to search for an interface, press F4.

   The Find Type dialog is then displayed, listing all available interfaces for classes in the schema from which you invoked the Interface Implementation Mapper dialog.

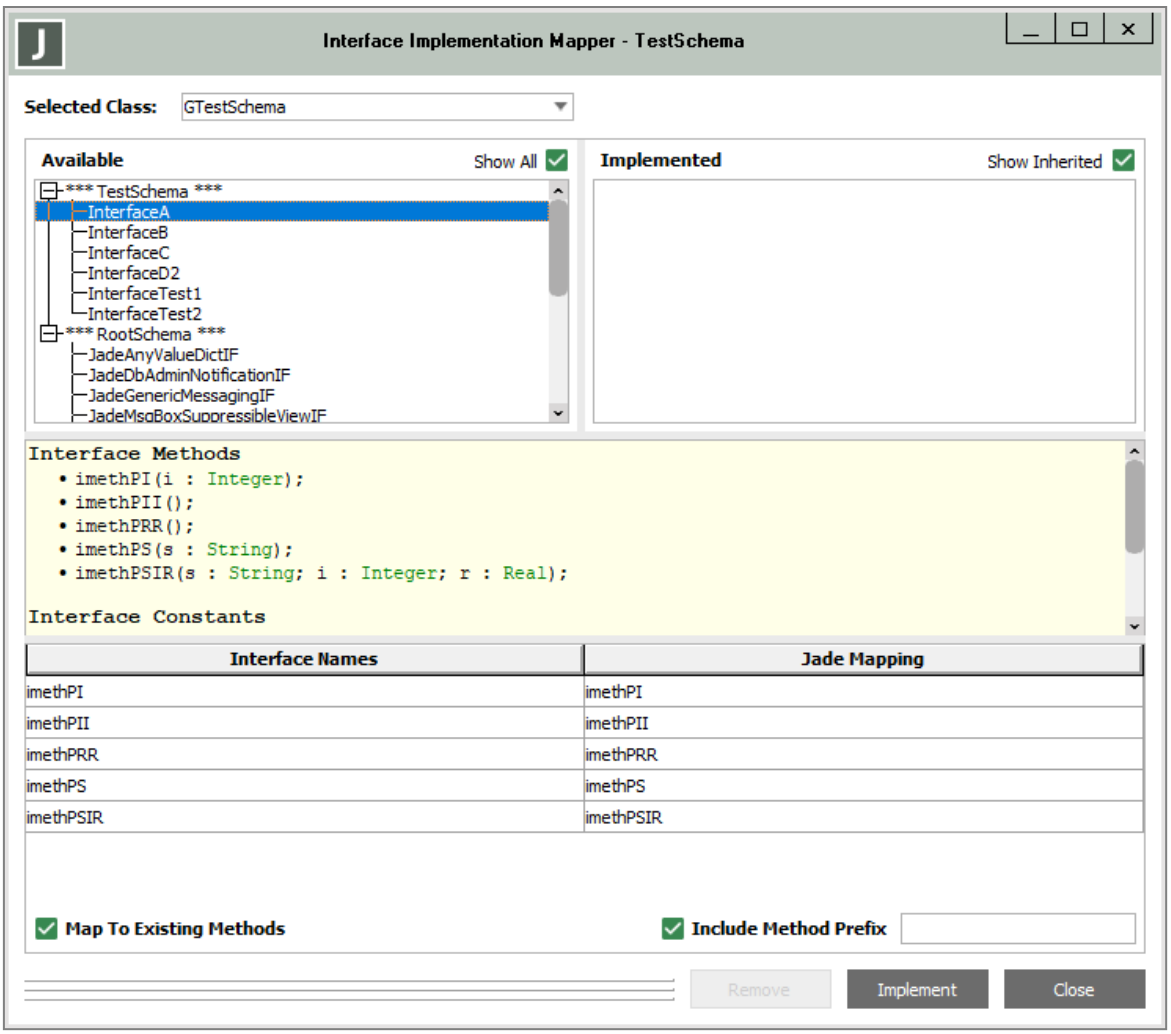


When you select an interface in the **Available Interfaces** list box of the Interface Implementation Mapper dialog (or in the **Find** text box of the Find Type dialog and you then click the **OK** button), the methods defined in the selected interface that the class must implement to satisfy the interface are then displayed in the first column of the table towards the bottom of the dialog.

An interface cannot be implemented by a class if it is already implemented by that class in a superschema or a subschema. These interfaces are disabled and cannot be selected in the available interfaces list box.

The Jade method mapping to the selected interface (that is, either a stub method that is automatically be generated in the selected class when the interface is implemented, or if you check the **Map To Existing Methods** check box, an existing method defined on the class) is displayed in the second column on the table.

In addition, a summary of the interface hierarchy, all methods, and constants defined in the selected interface are displayed in the read-only pane in the middle of the dialog, as shown in the following image.



The interface summary text box includes information about any super interfaces belonging to the selected interface. If the selected class does not implement one or more of these prerequisite super interfaces, they are displayed in red with the expression (not implemented) beside their name. In addition, if a super interface implementations is missing, the **Implement** button is disabled. If the super interfaces are implemented or inherited by the selected class, they are displayed in green.

4. Use the **Selected Class** drop-down combo box at the top of the dialog to quickly transition to different classes without closing and reopening the dialog every time.

5. If you want to list all interfaces for the current schema and its superschemas in the **Available Interfaces** list box, check the **Show All** check box. By default, only interfaces of the current schema view are listed (for example, if the owning Class Browser is viewing the current schema and superschema classes, by default interfaces in both those schemas are listed).

6. Check the **Show Inherited** check box above the implemented interfaces list box if you want to display the interfaces implemented on any superclasses of the currently selected interface in the interfaces list box. The inherited interfaces also display the schema and superclass in which they are implemented.

7. You can enter any valid method name in the combo box of a cell in the second column, to create a new class method (to which normal method naming rules apply) or you can select an existing class method (by using the drop-down list box portion of the combo box) from the list of any existing class method that has a complementary method signature.

8. Below the interface mapping table is the **Map To Existing Methods** check box, which is checked by default when the dialog opens so that the dialog attempts to map the interface methods to existing class methods with an identical name and compatible method signature.

   Note This overrides the method prefix behavior; that is, if the **Map To Existing Methods** check box is checked and a compatible method exists, the dialog attempts to map to this method regardless of the **Include Method Prefix** check box state.

9. If you want add a method prefix to *all* mapped method names, perform the following actions.

   a. In the text box at the right of the **Include Method Prefix** check box, specify the value (for example, **test_**) that you want to apply to all Jade methods that are mapped to the selected interface.

   b. Check the **Include Method Prefix** check box.

      Note The default value of this check box and the method prefix text box value are determined by your user profile settings; that is, the **Include Method Prefix** check box is checked when the **Generate stub methods without prefix** check box in the Interface Options group box on the **Miscellaneous** sheet of the Preferences dialog is unchecked.

10. To remove a method prefix from all mapped methods, perform the following actions.

    a. Specify the value in the text box at the right of the **Include Method Prefix** check box that you want to remove from all Jade methods that are mapped to the selected interface.

    b. Disable the **Include Method Prefix** check box.

       Tip If you want to remove a method prefix from all Jade methods in the target class that are mapped to the interface, perform the actions in this step before you click the Implement button.

       As the modification of a mapped method name (both adding and removing a prefix) creates a corresponding method, removing a prefix does *not* remove the method from the target class. For more details, see "Stopping the Implementation of an Interface", later in this chapter.

11. Click the **Implement** button.

    The selected class is updated to include any new methods that are required to satisfy the interface. These methods contain only the method signature and template. A mapping is then set up between the interface methods and the corresponding Jade methods that were selected for mapping in the dialog. (These can be new or existing methods, depending what was selected in the mapping table.)

12. Click the **Close** button.

    If you click the **Close** button or select another class or interface before you have clicked the **Implement** button, a message box is displayed, prompting you to confirm that you want to save pending changes. If you click the **No** button in this message box, your mappings to that interface are not saved.

Generated stub methods in classes implementing interfaces provide only the method signature and template; that is, they do not provide any implementation details. It is *your* responsibility to define the appropriate source for each method in the implementing class so that it performs the required action or actions.

In the Interface Options group box on the **Miscellaneous** sheet of the Preferences or Jade Installation Preferences dialog, if you want:

- A compile-time error generated when the stub method is created (that is, it is marked in error in the Methods List of the Class Browser), check the **When implementing an interface, do not compile automatically generated stub methods** check box.
- New interface method names to be generated with the same name as the original interface method name, with no added prefix, check the **Generate stub methods without prefix** check box.

  When you check the **Generate stub methods without prefix** check box, the **Generated stub method prefix** text box is disabled and the generated stub methods have the same name as the interface method (unless there is a conflict with an existing name, in which case a number is appended to the name).
- To specify the method name prefix that is generated for all new implemented interface methods, specify the prefix that you require in the **Generated stub method prefix** text box.

  When a method is added to the interface, the prefix specified in the user preferences is used unless you check the **Generate stub methods without prefix** check box, in which case no prefix is used.

Note These options apply to new implemented interface methods only; that is, they do not affect existing implemented interface method names. For details, see "Maintaining Miscellaneous Options", in Chapter 2.

Stub methods are then automatically generated with body details that you must comment out or remove before the method compiles, as a way of highlighting that you have yet to provide the required method body. For example, you can locate these empty methods by displaying methods in error, to find the methods that you are likely to want to flesh out to make the implementation useful.

To display the interface methods to which a class method is mapped, see "Displaying Interface Methods to which the Selected Method is Mapped", in Chapter 4. See also "Viewing Interface Method Mappings", earlier in this chapter, for details about displaying the implemented methods grouped by the interface or interfaces in which they are defined in the Methods List of the Class Browser.

Updating an Existing Interface Method Implementation

The Interface Implementation Mapper dialog enables you to remap an existing interface method mapping, by:

- Selecting and viewing the required interface.
- Entering a new method name or selecting one of the currently compatible methods for each method that you want to remap.
- Updating the existing interface mapping implementation.

To update an existing interface method implementation

1. In the Class List of the Class Browser, select the class whose interface mapping you want to change.
2. To quickly transition to a different class without closing and reopening the dialog every time, select the required class in the **Selected Class** drop-down combo box at the top of the dialog.
3. Select the **Interface Mapping** command from the Classes menu. The Interface Implementation Mapper dialog, containing the current mapping details, is then displayed.
4. In the **Implemented Interfaces** list box, select the interface whose mapping methods you want to change.

   Note If the **Show Inherited** check box is checked, the Implemented Interfaces list box also include interfaces implemented on superclasses of the selected class. These mappings cannot be updated. You can update only mappings belonging to the selected class. The inherited interfaces also display the schema and superclass in which they are implemented.

5. Change the required mapping method or methods in the second column of the table by entering a new method name or by selecting a complementary method from the drop-down list in the combo box.
6. If you want to make bulk changes to all method names, perform the following actions.
   a. In the text box to the right of the **Include Method Prefix** check box, specify the value that you want to remove from all Jade methods that are mapped to the selected interface.
   b. Check (enable) the **Include Method Prefix** check box.
7. To remove a method prefix from all mapped methods, perform the following actions.
   a. In the text box at the right of the **Include Method Prefix** check box, specify the value that you want to remove from all Jade methods that are mapped to the selected interface.
   b. Uncheck (disable) the **Include Method Prefix** check box.

   Tip If you want to remove a method prefix from all Jade methods in the target class that are mapped to the interface, perform the actions in this step before you click the Implement button.

   As the modification of a mapped method name (both adding and removing a prefix) creates a corresponding method, removing a prefix does *not* remove the method from the target class. For more details, see "Stopping the Implementation of an Interface", in the following section.

8. Click the **Update** button, to update the existing interface mapping implementation.

   Notes The target class is updated to include any new methods that are required to satisfy the interface. Generated stub methods in classes implementing interfaces provide only the method signature and template; that is, they do not provide any implementation details. (For details about using the **Miscellaneous** sheet of the Preferences dialog to specify that do not cause a compile error, see "Maintaining Miscellaneous Options", in Chapter 2.) To compile the method or methods in the target class, delete or comment out the text line. It is *your* responsibility to define the appropriate source for each method in the implementing class so that it performs the required action or actions.

   A message box is displayed if you are attempting to implement an interface on the current version of a class when it is already implemented on the latest version, prompting you to discontinue the interface implementation.

9. Click the **Close** button.

If you click the **Close** button before you have clicked the **Update** button, a message box is displayed, prompting you to confirm that you want to save pending changes before closing the dialog. If you click the **No** button in the Interface Implementation Mapper message box, your mappings to that interface are not updated.

Stopping the Implementation of an Interface

You cannot delete a class method that is currently participating in a mapping to an interface method. Removing the method would mean that the class did not correctly implement the interface (that is, it would be an illegal operation).

To remove the method, the current interface mapping must be transferred to another method (maintaining the contract that a class implements *all* methods supplied by an interface) or the class must cease implementing the interface. You can perform these actions by using the Interface Implementation Mapper dialog.

» To stop the implementation of an interface

1. In the Class List of the Class Browser, select the class whose interface mapping implementation you want to stop.
2. Select the **Interface Mapping** command from the Classes menu. The Interface Implementation Mapper dialog, containing the current mapping details, is then displayed.
3. In the **Implemented Interfaces** list box, select the interface whose implementation you want to stop.
4. To transfer an interface mapping to another method, change the required mapping method in the second column of the table by entering a new method name or by selecting a complementary method from the drop-down list in the combo box.

   Perform steps 5 through 8 under "Updating an Existing Interface Method Implementation", in the previous section, to change the method mapping implementation.
5. To remove an existing interface implementation from the class, click the **Remove** button. (This button is enabled only when they selected interface has been implemented for that class.)
6. Click the **Close** button.

Using Interfaces in your Applications

You can apply interfaces, as Jade types, in your applications in several ways. The following subsections include some of the most common of these.

Note You can address each of these cases without interfaces, but well-applied interfaces bring clear benefits, which are more significant depending on the size and complexity of your application specification.

Code Re-Factoring

As you make changes during the life cycle of an application, code re-factoring is necessary in order to maintain the application specifications in an organized manner. You can use interfaces as a flexible element to help in the maintenance of your application code.

You can define new interface methods that reflect new behavior together with the creation of new code in your application classes. For example, when you add significant and discrete pieces of new functionality to the application, you can use interfaces to keep readability and modularity in the specifications, which are now more complex.

If the new functionality involves the collaboration of several otherwise disjointed classes, instead of adding the new code directly into those classes, you can create new interfaces to reflect the operations required by the new functionally and then implement the interfaces in the existing classes later. This process clearly states in your application that the new code is related to each unit of new functionality.

Another benefit of involving interfaces in code re-factoring is the ability to use interfaces as extra dimensions for code navigation. The Jade Platform development environment provides a number of facilities that enable you to search for references to interfaces, their methods and constants, and the classes that implement them. (For details, see "Maintaining an Interface" and "Displaying all References to the Selected Interface", earlier in this chapter.)

Frameworks and Packages

Interfaces are often fundamental components of frameworks. In Jade, you can implement a framework: inline (that is, in a schema hierarchy) or as a package.

The following discussion applies to those implementations.

Frameworks usually require that client-supplied objects respond to some specific methods, which are known as *callback* methods. Most frameworks are written as generalized as possible, so that they can be utilized in a large number of situations. When you write a framework, you do not yet know the specific classes of the objects with which it will be working but that interaction with client-supplied code should occur according to your intentions.

To define the collaboration with the eventual code that will interact with the framework, you can use the following approaches.

- Use a very general method that is defined in a class that the framework assumes its clients know about. In a very generic framework, this class ends up being the **Object** class.
- Define an interface with the required methods and leave it to the users of the framework to implement it with the class or classes that are most convenient for them; that is, users will publish aspects of object collaboration without revealing its actual class.

The second of these approaches is the best in terms of flexibility and readability, and it allows for more-specific checks at compile time.

New System Development

During the design phase of a new system, it is good practice to keep the main modules of the system as independent as possible and their inter-communications clearly defined. This enables you to change the implementation details of different modules with a minimum impact on other modules.

The different modules of the system can be at the same level of abstraction and functionality, which is referred to as encapsulation at the module level. If the modules are at different levels, the separation is referred to as layering. In either case, you can use interfaces at this early stage to define the point of contact between two modules.

As you develop each module, provide the classes and methods that you intend to satisfy the different interface requirements. When interfaces and classes are paired together, the compiler can then verify that the classes supply the corresponding methods for each interface. This is a kind of compile-time dynamic binding mechanism that you can use to verify that the interaction between modules is well-defined yet you maintain the ability to replace those modules at different times in the system life time.

Note An accessed interface constant must be qualified as ***interface-name.constant-name***, not ***class-name.constant-name***.

Interface-Related Methods and Properties

Jade provides the interface-related methods summarized in the following table. (For details, see Volume 1 or Volume 2 of the *Encyclopaedia of Classes*.)

Class and Method	Description
Class::implementsInterface	Returns **true** if the specified interface is implemented by the receiver
Object::beginClassNotificationForIF	Sends notification events on instances of a class and its subclasses to methods mapped to the **userNotification** and **sysNotification** methods of the **theInterface** parameter, rather than to those of the subscriber
Object::beginClassesNotificationForIF	Sends notification events on instances of a class and optionally its subclasses to methods mapped to the **userNotification** and **sysNotification** methods of the **theInterface** parameter, rather than to those of the subscriber
Object::beginNotificationForIF	Sends notification events on an object to methods mapped to **userNotification** and **sysNotification** methods of the **theInterface** parameter, rather than to those of the subscriber
Object::beginTimerForIF	Arms a timer for methods mapped to the **timerEvent** method of the **theInterface** parameter, rather than to that of the subscriber
Object::endClassNotificationForIF	Ends an interface event notification registered using the **beginClassNotificationForIF** method for the corresponding parameters
Object::endClassesNotificationForIF	Ends an interface event notification registered using the **beginClassesNotificationForIF** method for the corresponding parameters
Object::endNotificationForIF	Ends an interface event notification registered using the **beginNotificationForIF** method for the corresponding parameters
Object::endTimerForIF	Terminates an interface timer initiated using the **beginTimerForIF** method for the corresponding parameters
Object::getTimerStatusForIF	Returns the status of a specified timer that was initiated using the **beginTimerForIF** method for the corresponding parameters, if it is currently active
Object::respondsTo	Returns **true** if the receiver's class or its superclasses implement the specified Jade interface
Schema::allJadeInterfaces	Returns a reference to all interfaces defined the receiver and its superschemas
Schema::getJadeInterface	Returns a reference to the specified interface

Jade provides the interface-related properties summarized in the following table. (For details, see Volume 1 or Volume 2 of the *Encyclopaedia of Classes*.)

Class and Property	Description
Class::implementedInterfaces	Contains references to the interfaces implemented by the class

Class and Property	Description
Notification::featureNumber	Contains a value that allows for identification of the interface method that was mapped by the subscriber
Notification::isInterface	Specifies whether the notification was registered by an interface notification method
Notification::typeNumber	Contains a value that allows for identification of the associated interface

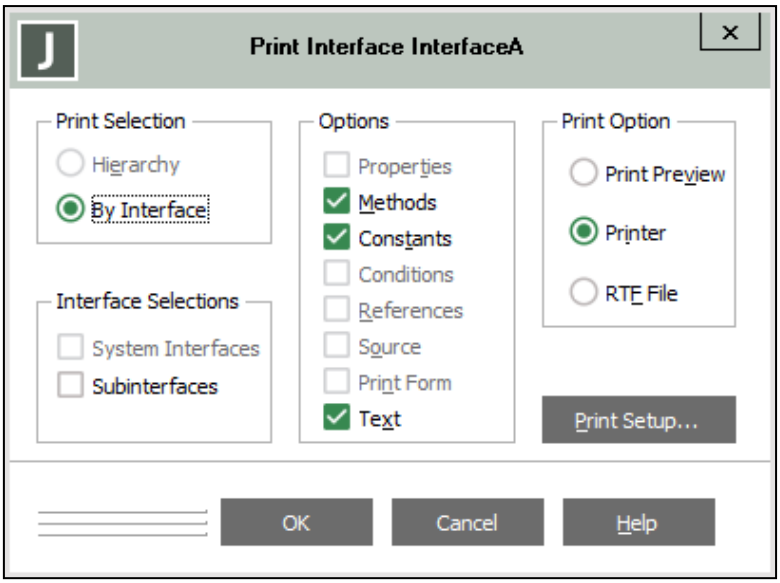
Printing an Interface

The Print Options dialog, accessed by clicking the **Print Selected** icon in the Browser toolbar or by selecting the **Print Selected** command from the File menu, enables you to print the interface selected in the Interface List of the Interface Browser.

» To select the interface print options

1. In the Interface List of the Interface Browser, select the interface that you want to print.
2. Perform one of the following actions.
   - Click the **Print Selected** icon in the Browser toolbar
   - Select the **Print Selected** command from the File menu

   The Print Interface dialog, shown in the following image, is then displayed.



As the concepts of hierarchies and system interfaces do not apply, the **Hierarchy** option button in the Print Selection group box and the **System Interfaces** option button in the Interface Selections group box are disabled. In addition, check boxes for print options that do not apply to interfaces (for example, properties and method source) are disabled in the Options group box.

3. In the Options group box, check the appropriate check boxes to select the options that you require for your interface documentation. A check mark is displayed in check boxes of options that are selected for printing. By default, the constants, methods, and any specified text are printed (that is, these check boxes are checked).
4. If you do not want your selections to be output to the default printer, select one of the following.
   - **Print Preview** option button, to preview the output on your workstation monitor. For details about previewing print output, see "Previewing Print Output" under "Printer Class", in Chapter 1 of your *Encyclopaedia of Classes*.
   - **RTF File** option button, to output your selections to a Rich Text Format (.**rtf**) file in your working (**bin**) directory. You can view your .**rtf** file output by using a text editor; for example, Word for Windows.
5. Click the **Print Setup** button if you want to change the setup of your printing. For details, see "Setting Up Your Printer", in Chapter 3.
6. Click the **OK** button to confirm your selections. Alternatively, click the **Cancel** button to abandon your selections.

Extracting an Interface

You can extract an interface as part of the schema in which it is defined, or you can extract only the interface itself (and any constants, methods, and text defined in the interface).

Note As the complete interface must be extracted when an interface method is required to be extracted, the extract process validates that methods specified in a parameter file are not defined as part of an interface. (For details about extracting to a parameter file, see "Specifying Your Parameter File Options", in Chapter 10.)

To extract an interface only

1. In the Interface List of the Interface Browser, select the interface that you want to extract.
2. Select the **Extract** command from the Interfaces menu.

The Extract dialog is then displayed, to enable you to select the options that you require. For details, see "Extracting Your Schema", in Chapter 10.

The file name defaults to the name of the current interface, with a .**cls** suffix. The location defaults to your Jade working directory; for example:

```
s:\jade\test\bin\InterfaceA.cls
```

As the associated constants, methods, and text are also extracted when you extract an interface, the **Extract** command is a quick means of extracting an interface for:

- A backup
- Passing the interface to another user
- Reconstructing the extracted interface

The interface **Extract** command is a simpler, faster way to extract a specific interface and all constants, methods, and text defined in the interface than performing a selective extract of your schema.

Note If you load an extracted interface into a different schema (by using the Advanced Load Options dialog to specify the target schema), the interface and any elements defined in that interface is created in the target schema but no corresponding class and stub methods are created.

If you require these, you must define the appropriate class (or classes) and stub methods manually or you can extract each class that implements mapping to the interface in the source schema and then load the extracted class or classes into the target schema after you have loaded the interface, before defining the mapping implementation that you require.

For details about extracting the interface as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10.

Removing an Interface

The **Remove** command from the **Interfaces** menu enables you to remove (delete) an interface selected in the Interface List of the Interface Browser.

To remove an interface

1. In the Interface List of the Interface Browser, select the interface that you want to physically delete from the Jade database.
2. Select the **Remove** command from the Interfaces menu. If the interface has sub-interfaces, property references, or it is exported, an error message box is then displayed, advising you why you cannot delete the interface.

   The **Remove** command is disabled when a method from an exported interface is checked out, so that neither the original method nor the checked out method or methods can be removed. Before you can remove the method, you must check it in or undo the checking out of the method.

   When deleting an interface or an imported package with an interface, a warning message advises you of the number of classes implementing the interface. If the interface is not exported or it does not have sub-interfaces or property references, a message box is then displayed.
3. To confirm that you want to remove the interface and all of its constants, methods, and text, click the **OK** button. Alternatively, click the **Cancel** button to abandon the deletion.

When the removal of the interface is successful, the Interface Browser is then updated to reflect the deletion.

For details about deleting schema elements, see "Removing a Schema Element", in Chapter 3.

Chapter 15 Using the Relational Population Service (RPS) Wizard

This chapter covers the following topics.

- Overview
- Defining an RPS Mapping
  - Using the RPS Relational Menu
- Adding an RPS Mapping
  - Setting Up the RPS Options
  - Selecting Classes for RPS
  - Mapping Classes to Tables
  - Selecting Columns for Tables
  - Adding Many-to-Many Relationships to an RPS Mapping
  - Mapping Many-to-Many Relationships
  - Maintaining RPS Column Mappings
  - Building the RPS Mapping
- Removing an RPS Mapping
- Changing an RPS Mapping
- Printing an RPS Mapping
- Extracting an RPS Mapping
  - Specifying Selective RPS Extract Options
- Loading an RPS Mapping
- Creating an Exclude Command File

Overview

The Relational Population Service (RPS) provides automatic replication of objects from a production Jade database to a relational database when running an RPS node.

The Relational Population Service wizard provides facilities that enable you to map classes, properties, and column-mapping methods to the required relational tables and columns.

The RPS mapping becomes part of the schema definition in the Jade database.

The Relational Population Service wizard supports:

- Microsoft SQL Server 2008 and higher
- Mapping a class to a table or to multiple tables
- Mapping of properties to multiple columns
- Mapping a column-mapping method (a method that has no parameters, that returns a non-exclusive object reference or a primitive type, and is not updating) to a table column
- Mapping many-to-many relationships to junction tables
- Mapping a class and its subclasses to a single table
- Historical table mode, where each transaction is preserved

For considerations when using RPS to replicate a production Jade database to one or more Relational Database Management System (RDBMS) target databases, see Chapter 2, "Relational Population Service (RPS) Support", in the *Synchronized Database Service (SDS) Administration Guide*.

For details about patch control for RPS mappings, see "Patch Control for RPS Mappings", in Chapter 3 of the *Development Environment Administration Guide*.

Defining an RPS Mapping

Use the Browse menu **RPS Mappings** command to browse for existing RPS mappings and to add and maintain RPS mappings.

» To open a Relational Population Service Browser

- Select the **RPS Mappings** command from the Browse menu.

A Relational Population Service Browser window is then opened, with the modification time of the selected RPS Mapping displayed in the status line.

If you have not yet defined an RPS mapping, nothing is displayed in the Relational Population Service Browser.

Using the RPS Relational Menu

The Relational Population Service Browser provides the Relational menu, to enable you to maintain your RPS mappings.

The Relational menu contains the commands listed in the following table.

Command	For details, see …	Action
Add	Adding an RPS Mapping	Displays the Relational Population Service wizard, to enable you to add a new RPS mapping
Remove	Removing an RPS Mapping	Deletes the selected RPS mapping
Change	Changing an RPS Mapping	Displays the Relational Population Service wizard, to enable you to maintain the selected RPS mapping
Print	Printing an RPS Mapping	Outputs the selected RPS mapping to the printer
Extract	Extracting an RPS Mapping	Extracts the selected RPS mapping

Command	For details, see …	Action
Load	Loading an RPS Mapping	Loads an RPS mapping from a file
Create Jcf	Creating an Exclude Command File	Creates an exclude command file for the selected RPS mapping

For details, see the following sections.

Adding an RPS Mapping

Add an RPS mapping to the current schema from the Relational menu in the Relational Population Service Browser. You can specify or override default values, exception policies, the database and user name, and the password on the RPS node.

Note You cannot add an RPS mapping with the same name as a relational view to a schema.

To add an RPS mapping to the current schema

- Select the **Add** command from the Relational menu.

The Relational Population Service wizard is then displayed. You can select one of the following buttons at any time (as appropriate).

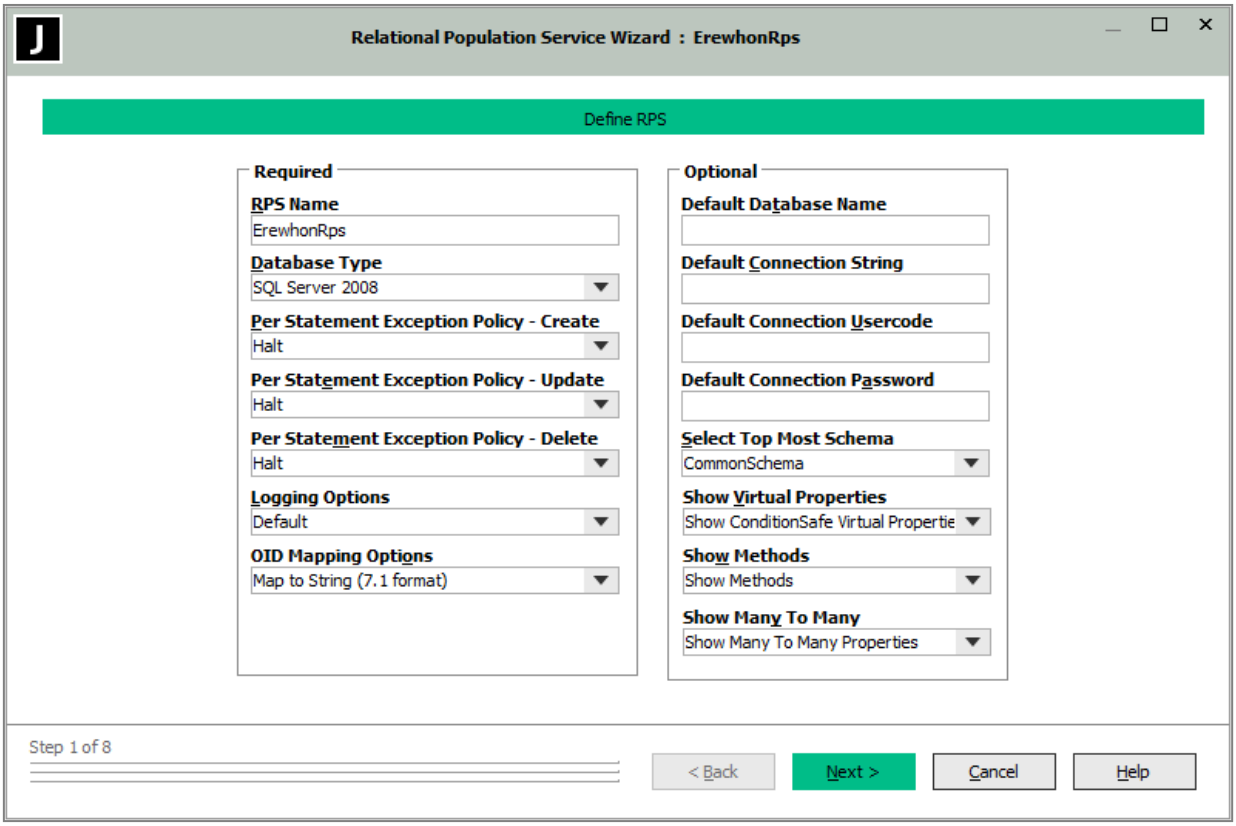
- **< Back**, to display the next sheet of the wizard
- **Next >**, to display the next sheet of the wizard
- **Cancel**, to exit from the wizard without saving any changes

For details about the RPS mapping steps, each contained on a separated sheet of the wizard, see the following subsections.

- Setting Up the RPS Options
- Selecting Classes for RPS
- Mapping Classes to Tables
- Selecting Columns for Tables
- Adding Many-to-Many Relationships to an RPS Mapping
- Mapping Many-to-Many Relationships
- Maintaining RPS Column Mappings
- Building the RPS Mapping

Setting Up the RPS Options

The **Define RPS** sheet of the Relational Population Service wizard, shown in the following image, enables you to define the name of your new RPS mapping and default connection information.



With the exception of defining a name, this information has default values or is optional.

To set up your RPS mapping options

1. In the Required group box, specify the following mandatory values.

   a. In the **RPS Name** text box, specify the name of your new RPS mapping.

   b. In the **Database Type** combo box, select the database type that the RPS mapping is targeting. The available database types are **SQL Server 2008** (the default), **SQL Server 2005**, and **SQL Server 2000**. Select **SQL Server 2008** if you are targeting versions of SQL Server 2008 and higher.

   Note If you change the selected database type from **SQL Server 2000** or **SQL Server 2005** to **SQL Server 2008**, you are prompted to confirm whether you want to retain existing legacy column mappings.

   Click the **Yes** button to retain existing legacy column mappings; for example, **datetime**, **image**, **text**, **ntext**, and so on. Alternatively, click the **No** button, to use the new default column mappings; for example, **datetime2**, and so on.

   c. Select the required action in the **Per Statement Exception Policy – Create** combo box. By default, the **Halt** value is selected, which aborts the transaction and ceases database tracking when a statement creation exception is raised.

Select the **Alternative Action** value if you want the create action to be attempted as an update if the create action fails. (For more details, see "Handling Exceptions", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.)

d. Select the required action in the **Per Statement Exception Policy – Update** combo box. By default, the **Halt** value is selected, which aborts the transaction and ceases database tracking when a statement update exception is raised.

   Select the **Alternative Action** value if you want the update action to be attempted as an insert if the update action fails. (For more details, see "Handling Exceptions", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.)

e. Select the required action in the **Per Statement Exception Policy – Delete** combo box. By default, the **Halt** value is selected, which aborts the transaction and ceases database tracking when a statement deletion exception is raised.

   Select the **Alternative Action** value if you want the deletion errors to be ignored. (For more details, see "Handling Exceptions", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.)

f. Select the **RPSTable** list item in the **Logging Options** combo box if you want create, update, or delete statement exception information recorded in a table in the relational database.

   By default, statement exceptions are logged only to the **jommsg.log** file. (For details, see "Exception Logging", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.)

g. Select the required object identifier (oid) mapping in the **OID Mapping Options** combo box. (For details, see "OID Columns", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.)

2. In the Optional group box, specify the following default connection options, if required.

   a. In the **Default Database Name** text box, specify the default name of the target relational database.

      If specified, the name is used when creating the relational table definitions and data load data to load into the correct database. If you do not specify a default database name, the user must specify it when the table definition or data are extracted.

   b. In the **Default Connection String** text box, specify the default connection string to the target database server from your RPS mapping.

      For SQL Server, this is the ODBC connection string, including the ODBC DSN name, required for the RPS node; for example:

      ```
      DSN=SqlServerODBC; Database=MyDatabase
      ```

   c. In the **Default Connection Usercode** text box, specify the default connection user code to be used to be used on the RPS node to connect to the relational database.

   d. In the **Default Connection Password** text box, specify the default connection password to be used on the RPS node to connect to the relational database.

      Note Passwords specified in the Relational Population Service wizard are extracted unencrypted in form and data definition (.**ddb** or .**ddx**) files.

   e. In the **Select Top Most Schema** combo box, select the highest-level superschema whose objects you want to include in the RPS mapping if you do not want to create the mapping from objects only in the current schema.

f. If you want all virtual properties displayed in the Relational Population Service wizard, select the **Show Virtual Properties** list item in the **Show Virtual Properties** combo box. Alternatively, if you want only condition-safe virtual properties (that is, they have the **conditionSafe** method option) included in the mapping, select the **Show ConditionSafe Virtual Properties** list item. By default, virtual properties are not displayed. See also "Toggling the Display of Virtual Properties" under "Selecting Columns for Tables", later in this chapter.

g. If you want methods displayed in the **Features** pane on the **Select Columns for Tables** sheet of the Relational Population Service wizard, select the **Show Methods** list item in the **Show Methods** combo box.

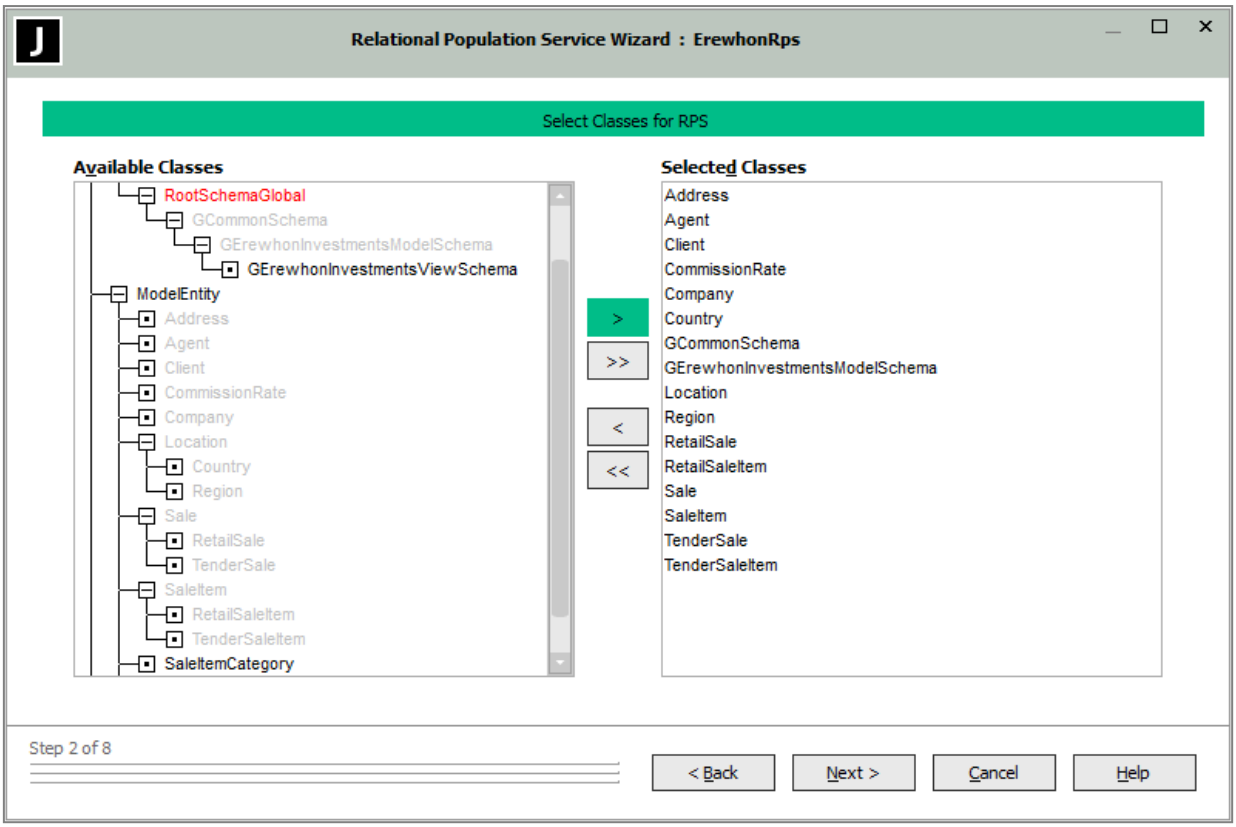
   By default, column-mapping methods (that is, methods defined in a class that have no parameters on the class, return a primitive type or a non-exclusive object reference, and that are not updating) are not displayed. See also "Toggling the Display of Methods" under "Selecting Columns for Tables", later in this chapter.

h. If you want to select the collections to be mapped to junction tables on the **Many To Many Table Selection** sheet of the Relational Population Service wizard, select the **Many To Many Properties** list item in the **Show Many To Many** combo box. See also "Adding Many-to-Many Relationships to an RPS Mapping" and "Mapping Many-to-Many Relationships", later in this chapter.

Selecting Classes for RPS

The **Select Classes for RPS** sheet of the Relational Population Service wizard enables you to specify the classes that you want to include in the mapping.

An example of the **Select Classes for RPS** sheet is shown in the following image.



If you selected a superschema of the current schema in the **Select Top Most Schema** combo box on the **Define RPS** sheet, all classes available for selection in superschemas up to and including the highest-level superschema that you selected are available for inclusion in the mapping. By default, superschema classes are not visible and cannot be included in the RPS mapping.

The Relational Population Service wizard:

- Uses the Window colors defined in your user profile (that is, specified on the **Editor** sheet of the Jade Platform development environment Preferences dialog) to display user objects and imported objects when displaying the class names on all sheets. For example, in the **Available Classes** list box by default, system classes are displayed in red, classes and interfaces imported from a package are displayed in green, user-defined classes that are inherited from a superschema are displayed in blue, and user-defined classes and interfaces that are defined in this schema are displayed in black.
- Displays bubble help on the classes displayed in the **Available Classes** list box, to enable you to distinguish imported classes with same name.

Notes Bubble help is displayed only if you have this option selected in your user profile (that is, the **Show Bubble Help** check box is checked on the **Window** sheet of the Jade Platform development environment Preferences dialog).

Only user-defined classes that can have persistent instances are included in your RPS mapping; that is, classes that have the **Allow Persistent Instances** check box checked on the **Lifetime** sheet of the Define Class dialog. (For details, see "Adding Classes to Your Schema", in Chapter 3.)

To include a class in your RPS mapping, perform one of the following actions

- Select the class in the **Available Classes** list box and then click the **>** button.

  Alternatively, use the Shift or Ctrl convention to select multiple classes.
- Double-click a class in the **Available Classes** list box to include it in your mapping.
- Right-click on a class in the **Available Classes** list box and then select the **Add Selected Items** command from the popup menu that is then displayed.
- Select a superclass in the **Available Classes** list box and then click the **>>** button, to include the selected class and all of its subclasses in the mapping.

Classes that are already selected are disabled (gray) in the **Available Classes** list box.

To locate a class in the Available Classes list box, perform one of the following actions

- Press F4.
- Right-click and then select the **Find** command from the popup menu that is displayed.

The Find Type dialog is then displayed, enabling you to specify the class you want to locate in the list box. (For details, see "Finding a Schema, Class, Interface, or Primitive Type", in Chapter 3.)

To remove a class from your RPS mapping, perform one of the following actions

- Select the class in the **Selected Classes** list box and then click the **<** button.

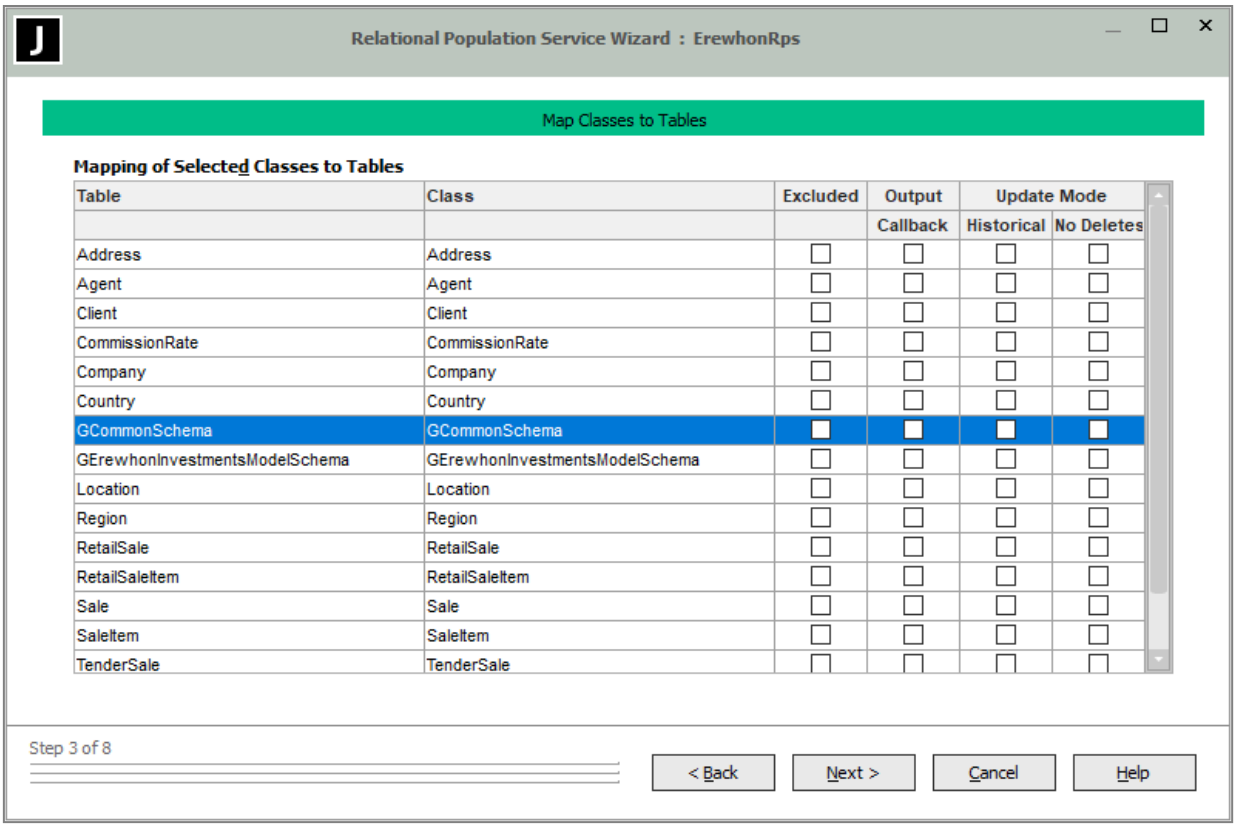
  Alternatively, use the Shift or Ctrl convention to select multiple classes.
- Double-click a class in the **Selected Classes** list box to remove it from your mapping.

- Right-click on a class in the **Selected Classes** list box and then select the **Remove Selected Items** command from the popup menu that is then displayed.
- Remove all classes from your mapping, by clicking the **<<** button.

Mapping Classes to Tables

The **Map Classes to Tables** sheet of the Relational Population Service wizard enables you to map the selected classes to relational database tables.

An example of the **Map Classes to Tables** sheet is shown in the following image.



The Relational Population Service wizard provides a default relational database table name for each selected class and default update modes. This sheet also enables you to edit or remove class-to-table maps.

For details about refining the tables displayed on this sheet of the Relational Population Service wizard, see the following subsections.

- Mapping Subclasses to a Class Map
- Creating a New Class Map
- Adding Additional Tables to a Class Map
- Removing a Table
- Removing Tables with no Columns

To refine the class to table mapping

1. In the **Table** column, click on the table name that you want to change and then specify the required table name. The name must be valid in the relational database, with a maximum length of 80 characters.

2. Check the check box in the cell of the **Excluded** column if you want to exclude the table from the RPS mapping when used on the RPS node, but retain the table in the schema definition.

   For details about customizing an RPS mapping for a specific site, see "Site-Specific RPS Mapping Customization", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

3. Check the check box in the cell of the **Output Callback** column if you want the table used in callback methods in a user-defined **Datapump** application; that is, you want to manipulate the output.

   The tables that are selected (checked) in this column are passed to your application before being output to the RDBMS database. The check boxes in this column are unchecked by default, to avoid the overhead of creating the dynamic objects and calling the user routine when not required.

   Note Selecting this option has no effect if callbacks are not registered in your **Datapump** application or if you have not defined your own **Datapump** application. For details about user-defined Datapump applications, see "RPS Datapump Application", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

4. In the **Update Mode** columns:

   - In **Historical** mode (when the check box in the **Historical** column is checked), object updates and deletions in the source (Jade) database are mapped to a new row in the target table or tables.

     In **Overwrite** Mode (when the check box in the **Historical** column is unchecked), an object update updates the corresponding row or rows "in place" and an object deletion results in the deletion of the target row or rows. (By default, tables are overwritten; that is, these check boxes are unchecked.)

   - Check the **No Deletes** check box of the table if you do not want the deletion of instances propagated to the target relational database. By default, deleted instances are deleted in the tables in the relational database to which they are mapped (that is, the **No Deletes** check box of each row is not checked).

     The **No Deletes** mode is not valid when **Historical** mode is selected.

5. Repeat steps 1 through 4 of this instruction for all class to table maps that you want to refine for your RPS mapping.

Mapping Subclasses to a Class Map

To map subclasses of a selected class to the same class map and table

- In the **Class** column, right-click on the class whose subclasses you want to include in the same class map and then select the **Map Subclasses to this Class Map** command from the popup menu that is displayed.

  This command is disabled if the selected class does not have subclasses.

All subclasses of that class are then mapped to the same table. Each subclass is displayed in a separate row. The down arrow at the right of the icon indicates that the class is a subclass of the class whose icon has a green arrow that points to the left.

Changes to the table name, update mode columns, or excluded or output callback options apply to all rows mapped to the same table.

Creating a New Class Map

To map a class to multiple tables with different update modes or excluded or output callback options, create a new class map and set the update modes to meet your requirements.

To create a new class map

1. In the **Class** column, right-click on the class for which you want to create a new class map and then select the **Create New Class Map for Class** command from the popup menu that is displayed.

   A new row is then created below the selected row, with the cell in the **Table** column highlighted with a bright yellow background, indicating that it is in edit mode and a value is required.

2. In the **Table** column, specify the new name for the table in the relational database. The name must be valid in the relational database and has a maximum length of 80 characters.

   An error message box is displayed if you do not specify a name for the table or the specified value is already defined in the RPS mapping.

The new class map is independent of the class map from which it was created.

Adding Additional Tables to a Class Map

To map a class to multiple tables with the same update mode or excluded or output callback options (for example, to split features from the class to different tables because of size), add a table to the class map.

To add additional tables to a class map

1. Right-click on the class to which you want to add additional tables and then select the **Add Additional Tables to Class Map** command from the popup menu that is displayed. (This command is disabled if the selected class has subclasses mapped to the same table. For details, see "Mapping Subclasses to a Class Map", earlier in this chapter.)

   A new row is then created below the selected row, with the cell in the **Table** column highlighted with a bright yellow background, indicating that it is in edit mode and a value is required. A link icon at the left of the name in the **Class** column indicates that the tables are linked to the same class map. The update modes and excluded and output callback options must be the same.

2. In the **Table** column, specify the name of the table to which the class is mapped. The name must be valid in the relational database and has a maximum length of 80 characters.

Removing a Table

To remove a table from a class map

- Right-click on the row of the table whose class map or shared class map you want to remove from the RPS mapping and then select the **Remove Table from Shared Class Map** or **Remove Table Class Map** command from the popup menu that is displayed, depending on the way in which the table was created.

The selected row is then removed from the RPS mapping.

Removing Tables with no Columns

To remove all tables with no columns from the RPS mapping

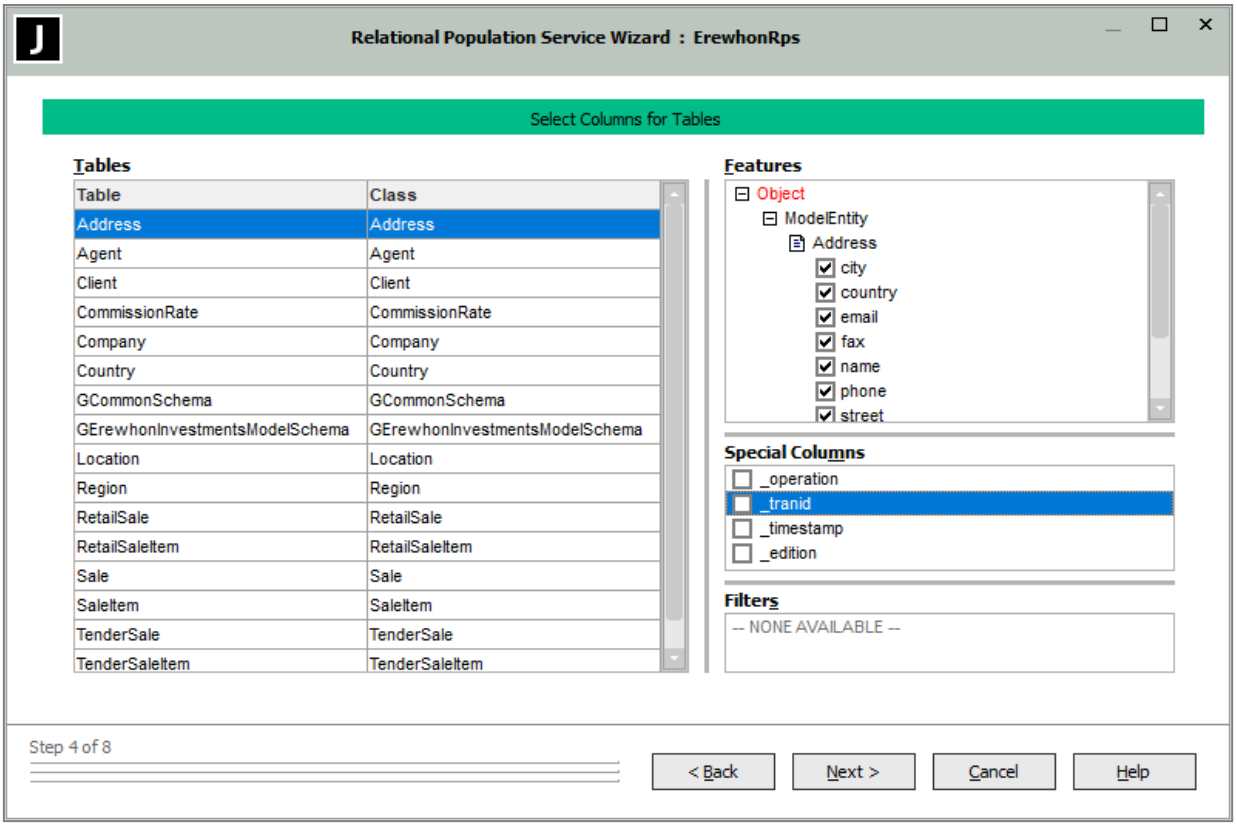
- Right-click on any row in the table and then select the **Remove Tables with no Columns** command from the popup menu that is displayed.

All tables that do not contain columns are then removed from the RPS mapping.

Selecting Columns for Tables

The **Select Columns for Tables** sheet of the Relational Population Service wizard enables you to select the columns for your relational database tables and to specify filters.

An example of the **Select Columns for Tables** sheet is shown in the following image.



For details about refining the columns displayed on this sheet of the Relational Population Service wizard, see the following subsections.

- Removing Tables
- Removing Tables with no Properties from the Map
- Exposing Properties for All Tables
- Exposing Properties for a Selected Table
- Clearing Exposed Properties for All Tables
- Clearing Exposed Properties for a Selected Table
- Exposing Special Columns for All Tables
- Exposing Special Columns for a Selected Table

- Clearing Exposed Special Columns for All Tables
- Clearing Exposed Special Columns for a Selected Table
- Toggling the Display of Virtual Properties
- Toggling the Display of Condition-Safe Virtual Properties
- Toggling the Display of Methods
- Toggling the Display of Many-to-Many Properties
- Displaying Tree Lines in Features and Filters Panes

To select columns for your relational database tables

1. In the **Tables** table, click on the row of the table whose features (attribute properties and column-mapping methods), special columns, or filters you want to refine.

   The **Features** pane is then populated with all Jade attributes, single-valued references, condition-safe or all virtual properties (if the display of all virtual properties or only condition-safe virtual properties is selected), and column-mapping methods (if the display of methods is selected) defined in the selected class. If the table has subclasses mapped to it, the hierarchy node tree contains subclasses of the selected class and properties and column-mapping methods defined in those subclasses.

   The properties (and methods, if selected) are sorted alphabetically with the properties first, followed by the methods that are also sorted alphabetically.

   All Jade attributes and single-valued references in the **Features** pane are selected by default. Column-mapping methods and virtual properties are not selected by default.

   The **Filters** pane is populated with all Jade condition methods on the selected class that can be used as filters for this table. No condition methods in the **Filters** pane are selected by default.

   If the table has subclasses mapped to it, the **Filters** pane is populated with all Jade condition methods defined in the selected class and the hierarchy node tree contains subclasses of the selected class and condition methods defined in those subclasses, if applicable.

   Tip You can display tree lines in the **Features** and **Filters** panes. For details, see "Displaying Tree Lines in Features and Filters Panes", later in this chapter.

2. In the **Features** pane, uncheck the check box at the left of each Jade attribute and single-valued references that you want to exclude from the table column.

   If virtual properties and column-mapping methods are displayed, check the check box at the left of each virtual property and column-mapping method that you want to include in the table column.

3. If you want the mapped table to include special columns, check the check box at the left of each of the special columns (that is, **_operation**, **_tranid**, **_timestamp**, and **_edition**) listed in the **Special Columns** pane that is to be included.

   Historical tables, by default, always include the **_edition**, **_operation**, and **_tranid** special columns.

4. If you want a condition method to be used by RPS to filter object instances (for example, that the value must be over or under a specific amount), check the check box at the left of the condition method in the **Filters** pane that you want to apply. For a table mapped from a single class, you can select one condition method only.

   If the table has subclasses mapped to it, you can select a condition method for each subclass.

A selected filter is evaluated as each operation is processed on the RPS node. An object instance is replicated to the target relational database only when the value of a condition is **true**.

Removing Tables

To remove a table from the RPS Mapping

- Right-click on the row of the **Tables** table whose class map or shared class map you want to remove from the RPS mapping and then select the **Remove Table Class Map** command from the popup menu that is displayed, depending on how the table was created.

The selected table row is then removed from the RPS mapping.

Removing Tables with no Properties from the Map

To remove all tables with no properties from the RPS mapping

- Right-click on any row in the **Tables** table and then select the **Remove Tables with no Properties** command from the popup menu that is displayed.

All table rows mapped to classes that do not contain properties are then removed from the RPS mapping.

Exposing Properties for All Tables

To expose properties for all tables in the RPS mapping

- Right-click on any row in the **Tables** table and then select the **Expose Properties for All Tables** command from the popup menu that is displayed.

All properties in all tables in the RPS mapping are then exposed for inclusion as columns in the relational database tables; that is, each property check box in the **Features** pane for a selected table is checked, indicating that all Jade properties will be replicated as columns in the relational database tables.

Exposing Properties for a Selected Table

To expose properties for a selected table

- Right-click on the table row in the **Tables** table and then select the **Expose Properties for Selected Table** command from the popup menu that is displayed.

All properties in the selected table are then exposed for inclusion as columns in the relational database table; that is, each property check box in the **Features** pane for that table is checked, indicating that all Jade properties will be replicated as columns in the relational database table.

Clearing Exposed Properties for All Tables

To remove properties in all tables from exposure as columns

- Right-click on any row in the **Tables** table and then select the **Clear Exposed Properties for All Tables** command from the popup menu that is displayed.

All properties in all tables in the RPS mapping are then removed from inclusion as columns in the relational database tables; that is, each check box in the **Features** pane for a selected table is unchecked, indicating that no Jade properties will be replicated as columns in the mapped relational database tables.

Clearing Exposed Properties for a Selected Table

To remove properties for a selected table from exposure as columns in the table

- Right-click on the table row in the **Tables** table and then select the **Clear Exposed Properties for Selected Table** command from the popup menu that is displayed.

All properties in the selected table in the RPS mapping are then removed from inclusion as columns in the relational database table; that is, each check box in the **Features** pane for that table is unchecked, indicating that no Jade properties in the table will be replicated as columns in the mapped relational database table.

Exposing Special Columns for All Tables

To expose special columns for all tables in the RPS mapping

- Right-click on any row in the **Tables** table and then select the **Expose Special Columns for All Tables** command from the popup menu that is displayed.

All special columns in all tables in the RPS mapping are then exposed for inclusion in the relational database tables; that is, each check box in the **Special Columns** pane for a selected table is checked, indicating that the **_operation**, **_tranid**, **_timestamp**, and **_edition** special column will be replicated as columns in the relational database tables.

Exposing Special Columns for a Selected Table

To expose special columns for a selected table

- Right-click on the class row in the **Tables** table and then select the **Expose Special Columns for Selected Table** command from the popup menu that is displayed.

All special columns in the selected table are then exposed for inclusion in the relational database table; that is, each check box in the **Special Columns** pane for that table is checked, indicating that the **_operation**, **_tranid**, **_timestamp**, and **_edition** special columns will be replicated as columns in the relational database table.

Clearing Exposed Special Columns for All Tables

To remove special columns in all tables from exposure as columns in the RPS mapping

- Right-click on any row in the **Tables** table and then select the **Clear Exposed Special Columns for All Tables** command from the popup menu that is displayed.

All special columns in all tables in the RPS mapping are then removed from inclusion in the relational database tables; that is, each check box in the **Special Columns** pane for a selected table is unchecked, indicating that no **_operation**, **_tranid**, **_timestamp**, and **_edition** special columns will be replicated as columns in the mapped relational database tables.

Note Historical tables will still include the required **_operation** and **_timestamp** special columns.

Clearing Exposed Special Columns for a Selected Table

To remove special columns for a selected table

- Right-click on the table row in the **Tables** table and then select the **Clear Exposed Special Columns for Selected Table** command from the popup menu that is displayed.

All special columns in the selected table in the RPS mapping are then removed from inclusion in the relational database table; that is, each check box in the **Features** pane for that table is unchecked, indicating that no **_operation**, **_tranid**, **_timestamp**, and **_edition** special columns in the class will be replicated as columns in the mapped relational database table.

Note Historical tables will still include the required **_operation** and **_timestamp** special columns.

Toggling the Display of Virtual Properties

To toggle the display of virtual properties

- Right-click on any row in the **Tables** table and then select the **Show Virtual Properties** command from the popup menu that is displayed.

A check mark is displayed to the left of the **Show Virtual Properties** command in the popup menu when virtual properties are shown in the table.

All virtual properties are then displayed in a red font or are removed from display in the **Tables** table.

The initial value depends on the value selected in the **Show Virtual Properties** combo box on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

Toggling the Display of Condition-Safe Virtual Properties

To toggle the display of condition-safe virtual properties

- Right-click on any row in the **Tables** table and then select the **Show ConditionSafe Virtual Properties** command from the popup menu that is displayed.

A check mark is displayed to the left of the **Show ConditionSafe Virtual Properties** command in the popup menu when condition-safe virtual properties are shown in the table.

All virtual properties whose mapping methods are condition-safe (that is, have the **conditionSafe** method option) are then displayed in a red font or are removed from display in the **Tables** table.

The initial value depends on the value selected in the **Show Virtual Properties** combo box on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

Toggling the Display of Methods

You can map a column to any *column-mapping method* (that is, a method that has no parameters, returns a primitive type or a non-exclusive object reference, and is not updating).

To toggle the display of column-mapping methods

- Right-click on any row in the **Tables** table and then select the **Show Methods** command from the popup menu that is displayed.

A check mark is displayed to the left of the **Show Methods** command in the popup menu when column-mapping methods are displayed.

All column-mapping methods are then displayed in a blue font or are removed from display in the **Features** pane.

The initial value depends on the value selected in the **Show Methods** combo box on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

Toggling the Display of Many-to-Many Properties

To toggle the display of many-to-many properties in collections

- Right-click on any row in the table at the left of the sheet and then select the **Show Many To Many** command from the popup menu that is displayed.

If many-to-many property relationships in collections were not displayed, a check mark is displayed at the left of the **Show Many To Many** command and all many-to-many relationships are displayed. Conversely, if many-to-many relationships were displayed, the check mark symbol is removed from the command and the display of many-to-many relationships is hidden.

The initial value depends on the value selected in the **Show Many To Many** combo box on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

Displaying Tree Lines in Features and Filters Panes

To toggle the display of tree lines in the Features and Filters panes

- Right-click in the **Features** or **Filters** pane whose tree lines you want to display and then select the **Show Tree Lines** command from the popup menu that is displayed.

Tree lines linking hierarchy nodes are then displayed in the pane or removed from display in the pane.

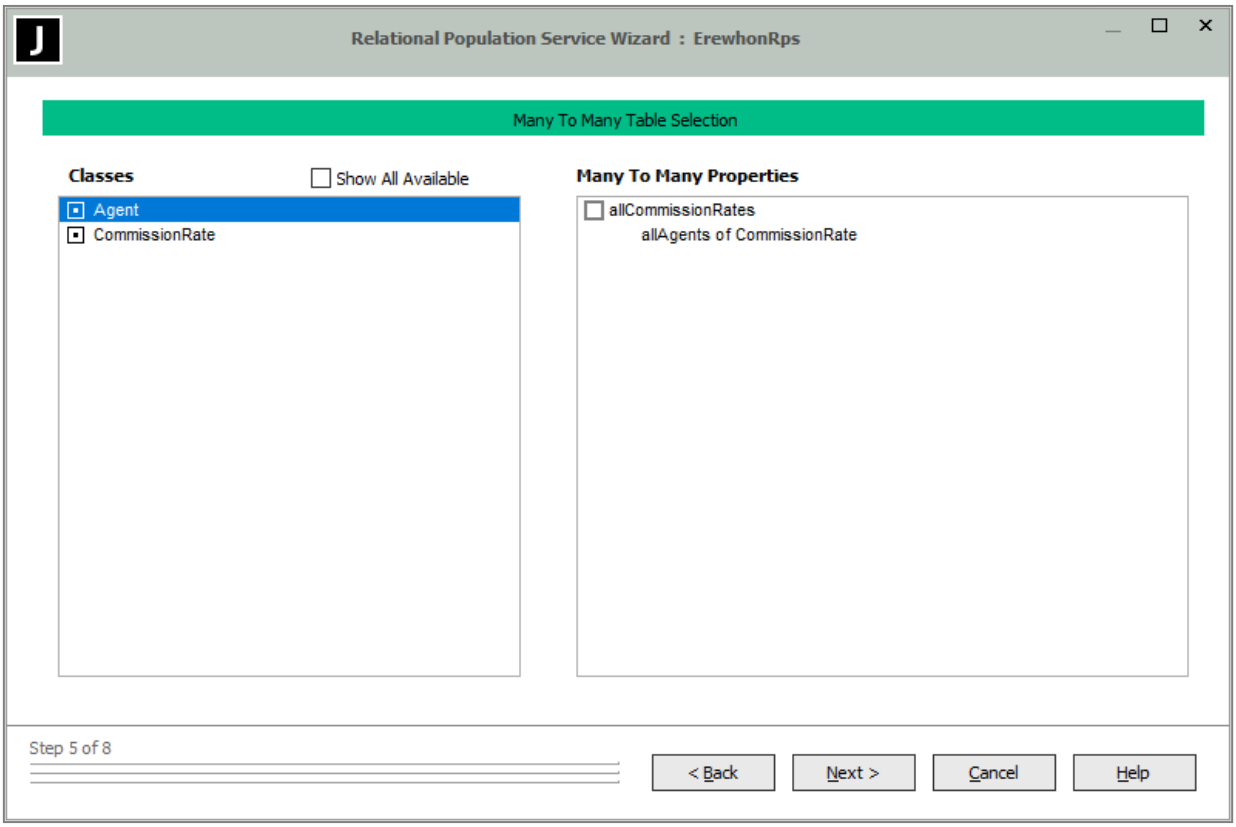
A check mark is displayed to the left of the command in the popup menu when tree lines are displayed in the **Features** or **Filters** pane.

Adding Many-to-Many Relationships to an RPS Mapping

The **Many To Many Table Selection** sheet of the Relational Population Service wizard enables you to select the many-to-many inverse properties in a Jade object model to an RPS database. This sheet is displayed only if the **Show Many To Many** combo box is checked on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

The many-to-many relationship is defined by creating a junction table in the relational database to split the *m-n* relationship in the Jade database into two *1-n* relationships.

An example of the **Many To Many Table Selections** sheet is shown in the following image.



When this sheet is first displayed, only the collections that are on classes that have been mapped in the RPS mapping are displayed in the list at the left of the sheet.

To select the collections to map to junction tables

1. If you want to display all persistent classes in the selected schemas that have many-to-many collection mappings, check the **Show All Available** check box.

   Note The junction table can be included in the RPS mapping even if the classes in the collections are not in the RPS mapping, although the table may be of limited use.

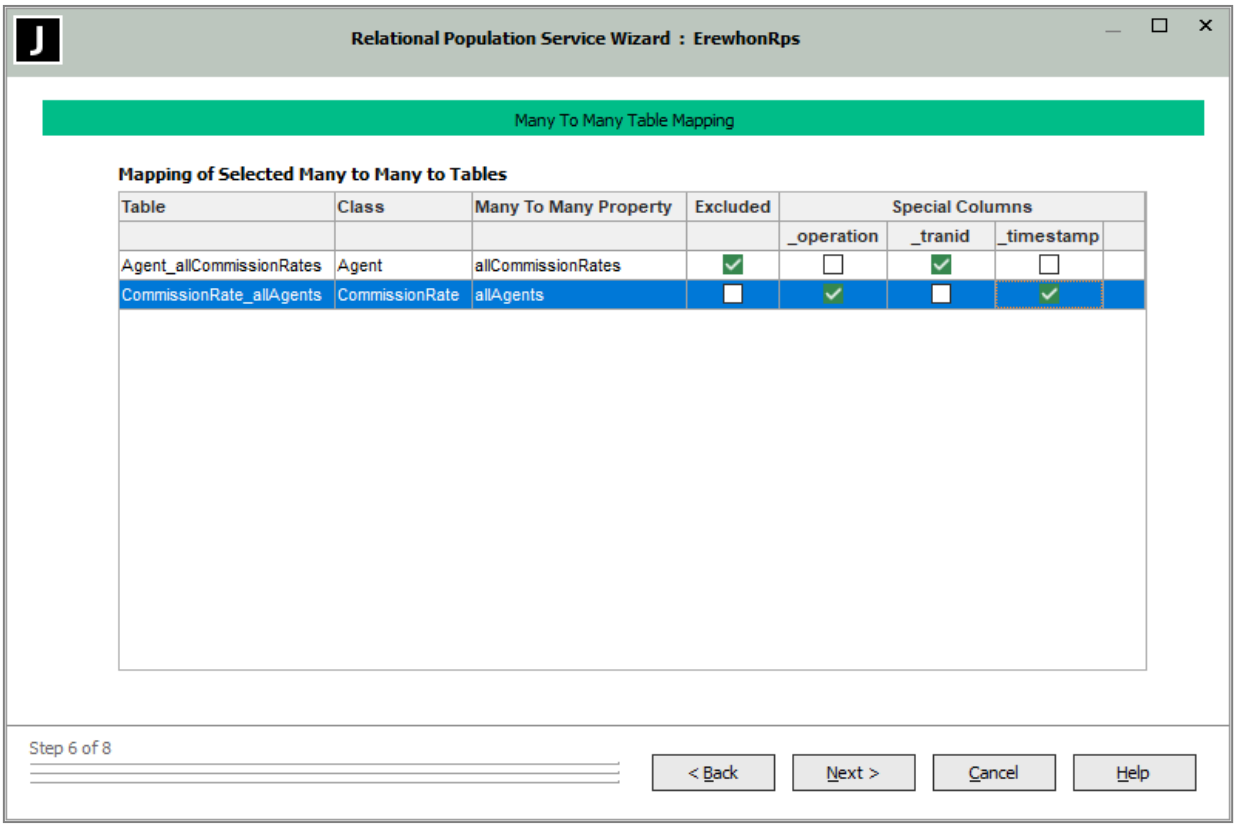
2. Select the required collection in the table at the left of the sheet, to create a junction table for the many-to-many relationship.

   You only have to select one end of the many-to-many relationship for inclusion in the RPS mapping. If you include both ends, the information is duplicated in the two junction tables that are created.

Mapping Many-to-Many Relationships

The **Many To Many Table Mapping** sheet of the Relational Population Service wizard enables you to specify options for the many-to-many inverse relationships selected on the **Many To Many Table Selection** sheet. (For details, see "Adding Many-to-Many Relationships to an RPS Mapping", in the previous section.) This sheet is displayed only if the **Show Many To Many** combo box is checked on the **Define RPS** sheet in step 1 of the RPS mapping. (For details, see "Setting Up the RPS Options", earlier in this chapter.)

An example of the **RPS Column Mappings** sheet is shown in the following image.



This sheet displays the junction tables included in the mapping and the class to which it is mapped. The default name of the junction table is in the *class-name_collection-name* format.

To specify generated table and columns names for many-to-many relationships

1. If you want to change the name of a table, enter the required name in the appropriate row in the **Table** column.
2. To mark the table as excluded, check the **Excluded** check box in the appropriate row.
3. To include one or more special columns, check the appropriate **_operation**, **_tranid**, or **_timestamp** check box in the respective columns.

Note The **_edition** special column is not applicable to junction tables. The operation of setting and clearing special columns for all tables does not apply to the junction tables.

Mapping a Subclass to a Selected Class

To map a subclass to a selected class map

- In the **Class** column, right-click on the class whose subclasses that contain many-to-many inverse references you want to include in the same junction table class map and then select the **Map SubClasses to this Class Map** command from the popup menu that is displayed.

  This command is disabled if the selected class does not have subclasses that contain many-to-many inverse references.

All subclasses with many-to-many inverse references of that class are then mapped to the same junction table. Each subclass is displayed in a separate row. The down arrow at the right of the icon indicates that the class is a subclass of the class whose icon has a green arrow that points to the left.

Changes to the table name, update mode columns, or excluded or output callback options apply to all rows mapped to the same table.

Removing Selected Junction Table Class Maps

To remove a selected junction table class map from the RPS mapping

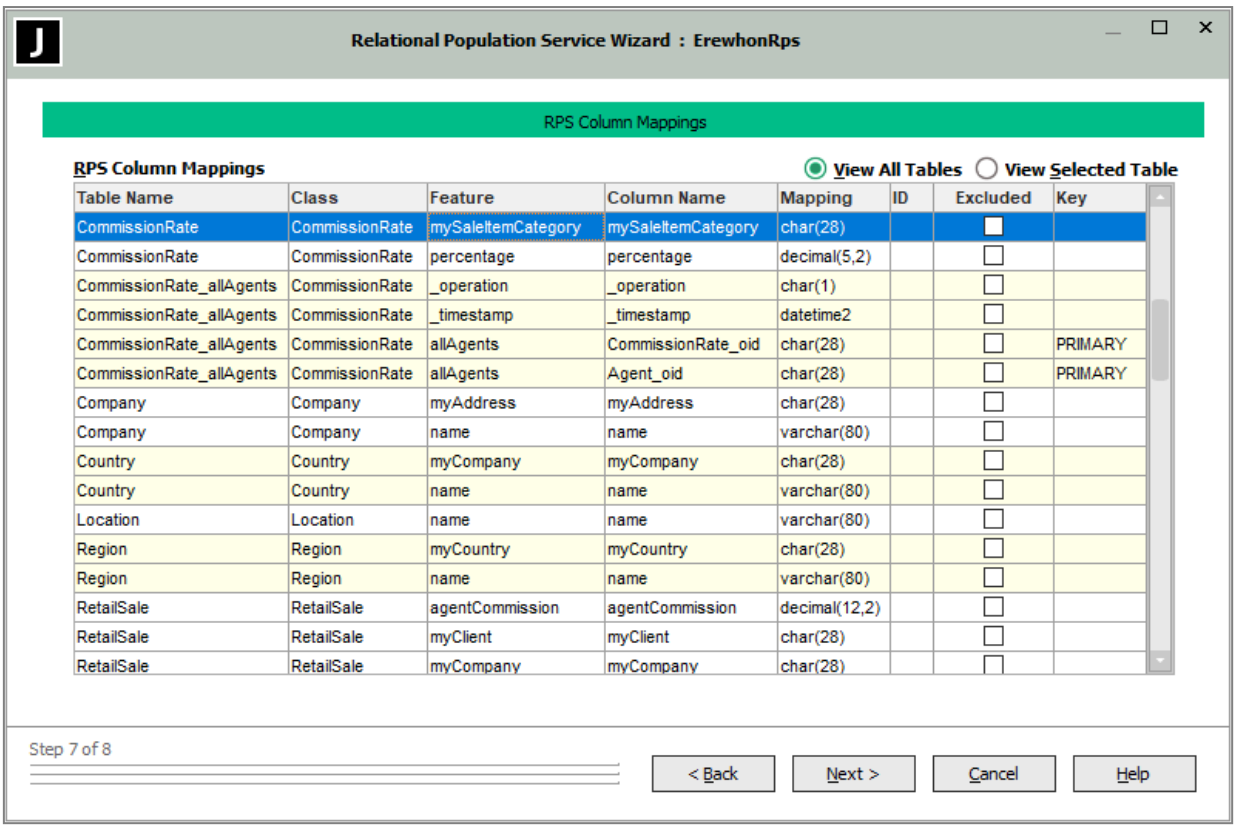
- Right-click on the row of the junction table whose class map you want to remove from the RPS mapping and then select the **Remove Table Class Map** command from the popup menu that is displayed.

The selected junction table class map is then removed from the RPS mapping.

Maintaining RPS Column Mappings

The **RPS Column Mappings** sheet of the Relational Population Service wizard enables you to map the columns in the relational database tables.

An example of the **RPS Column Mappings** sheet is shown in the following image.



By default, the **RPS Column Mappings** sheet displays all tables in the RPS mapping.

The default column name for a junction table column is ***class-name*_oid**. When the class of both collections is the same type, the second column in the junction table is created with the name ***class-name*_oid2**. You can change these junction column names but you cannot remove or split them.

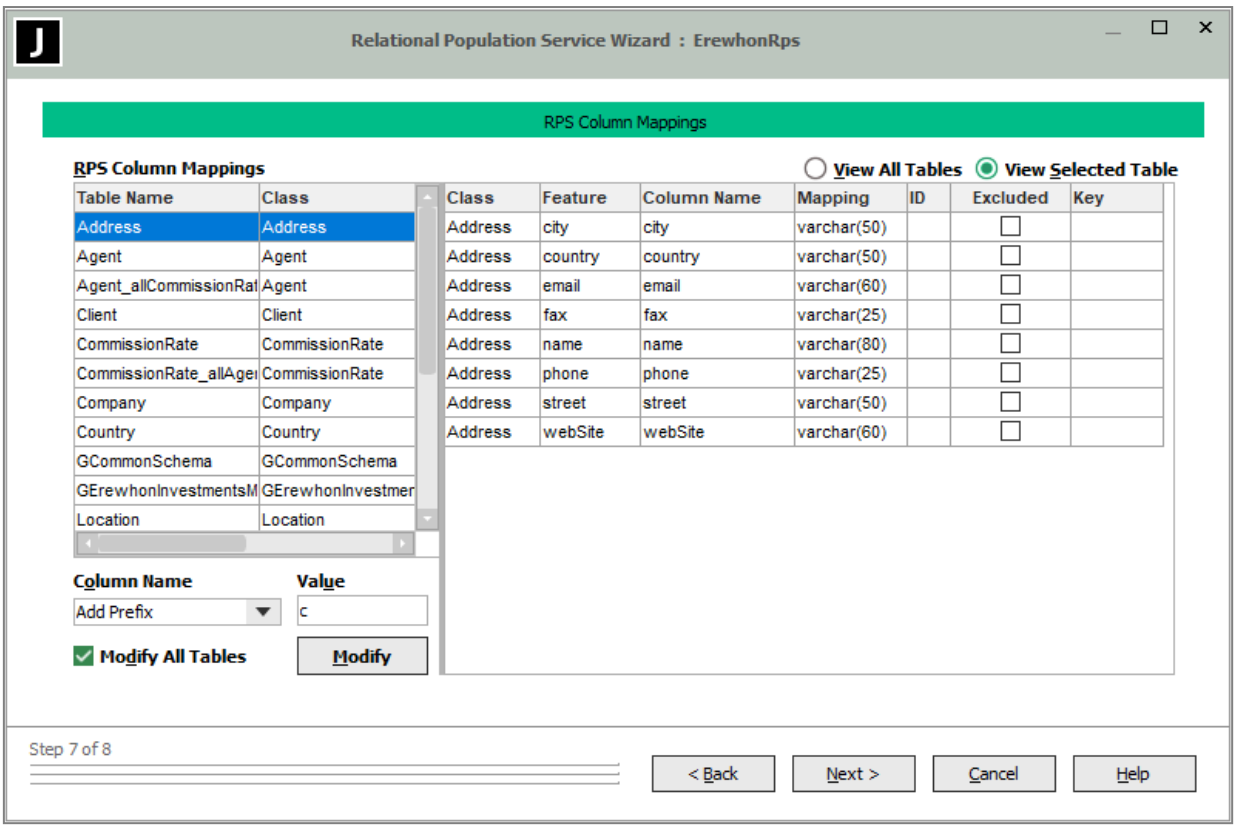
In the **Feature** column, the key icon at the left of a feature indicates that it is the **_type** feature of a subclass of the Jade class mapped to the table of its superclass. (For details, see "Mapping Subclasses to a Class Map", earlier in this chapter.) The value assigned to the column for the subclass is shown in the **ID** column.

The properties (and methods, if selected) are sorted alphabetically with the properties first, followed by the methods that are also sorted alphabetically.

As columns in the RPS mapping are grouped by table, each alternating table in the RPS mapping has an alternating background color of light yellow or white, making it easier for you to distinguish between the tables when all tables are viewed on the **RPS Column Mappings** sheet.

If your mapping consists of more than 50 tables whose columns you want to map (for example, your RPS mapping has more than the maximum of 32,000 table columns), the **View Selected Table** option button is selected by default, to display only the columns in the table whose row is selected in the **RPS Column Mapping**s table. You can then click the **View All Tables** option button at the upper right of the sheet, to display all tables in the **RPS Column Mapping**s table. If fewer than 50 tables are available for view, the **View All Tables** option button is selected by default.

An example of the **RPS Column Mappings** sheet for a single relational database table is shown the following image.



To specify your column mappings

1. If you want to change the name of a column, click in the required cell of the **Column Name** and then change the name to the value that you require. The name must be a valid column name in the target relational database and it can be a maximum of 80 characters.

2. A default type mapping is defined for each Jade property type or column-mapping method return type. The drop-down list box in the **Mapping** column contains any valid alternative mapping types for this column. To change the mapping type, select the required value in this drop-down list.

   The options available depend on the type of database selected in the **Database Type** combo box on the **Define RPS** sheet of this wizard. For example, under SQL Server 2000 and SQL Server 2005, strings with fewer than 255 characters are mapped to **VARCHAR** and those 255 characters or more are mapped to **TEXT**. For more details, see "RPS Mapping", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

   If column-mapping methods are included in the RPS mapping, these are displayed in a blue font. Column-mapping methods that return a **String**, **Binary**, or **Decimal** primitive type have default mapping types defined to accommodate the largest possible values that could be returned.

   You can redefine the column type of a column-mapping method to a more appropriate size for the data, by performing the following actions.

   a. In the drop-down list box in the **Mapping** column, select the value that has **?** in the length (for example, **VARCHAR[?]...** or **DECIMAL[?,?]...**). The RPS Wizard Metrics dialog is then displayed.

   b. Uncheck the **Maximum Length** check box if you want to specify a length or scale factor for the method column type and then enter the appropriate value in the **Length** or **Scale Factor** text box.

      It is your responsibility to ensure that the values returned by the column-mapping method fit into the allocated lengths. An exception is raised in the RPS **Datapump** application if the lengths are not valid.

   c. Click the **OK** button to update your method column type value. Alternatively, click the **Cancel** button to abandon your selection.

3. If you want to change the column name of a table or the column name of all tables:

   a. If you do not want the prefix or suffix to apply to all tables in the RPS mapping, uncheck the **Modify All Tables** check box. (A prefix or suffix value applies to all tables by default.)

   b. Ensure that the appropriate **Add Prefix**, **Add Suffix**, **Remove Prefix**, or **Remove Suffix** value is displayed in the **Column Name** list box.

   c. Specify the prefix or suffix value in the **Value** text box (for example, **test_**) that you want to apply to a table or to all tables in the RPS mapping.

   d. Click the **Modify** button.

   If the change that is being attempted would result in a duplicate column name, a message box is displayed, informing you that the change cannot be actioned. As no update action takes place, you must change the column name manually.

4. If you want to change the column identifier generated by the Relational Population Service wizard for a **_type** column of a subclass of the Jade class mapped to the table of its superclass, enter the value that you require in the **ID** column.

   Although you can change the identifier of a **_type** column mapped to a column, you cannot delete it.

   When you specify a string in the **ID** column, the length value in the associated **Mapping** column is updated to reflect the size of the longest identifier value. (The **ID** column does not contain user data but a user-defined identifier that enables you to determine the type of data that is in the record.)

5. In the **Excluded** column, check the check box in the cell of the **Excluded** column if you want to exclude the column from the RPS mapping when used on the RPS node, but retain the column in the schema definition.

   For details about customizing an RPS mapping for a specific site, see "Site-Specific RPS Mapping Customization", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

6. Each Jade class in a relational database table must have a primary key, to identify individual rows. The Jade object identifier (oid) maps to a relational database primary key attribute, which in **Overwrite** tables, identifies individual rows.

   Historical tables have **oid**, **_edition**, **_operation**, and **_tranid** defined as primary keys by default. You can add **_timestamp** special columns either in addition to or as a substitute for **_tranid** if it has been selected on the **Select Columns for Tables** sheet of the Relational Population Service wizard.

   See also "Primary Keys" under "RPS Mapping", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

   You can change a primary key mapping for tables marked as **No Deletes**. Select a different key in the **Key** column of an attribute, if required. For details, see "Mapping Classes to Tables", earlier in this chapter.

   Caution If the selected column has multiple duplicated values, insertions, updates, or deletions to the relational database may fail or overwrite data unintentionally. In most cases, you should leave the oid as the primary key.

7. If you want to map the selected item into multiple columns:

   a. Right-click on the appropriate row and then select the **Split Item Into Multiple Columns** command from the popup menu that is then displayed. A row is then created below the selected row, with the background color of the **Column Name** column displayed in bright yellow as an indication that you must specify a name.

   b. In the highlighted **Column Name** column, specify a valid column name that is unique in the table.

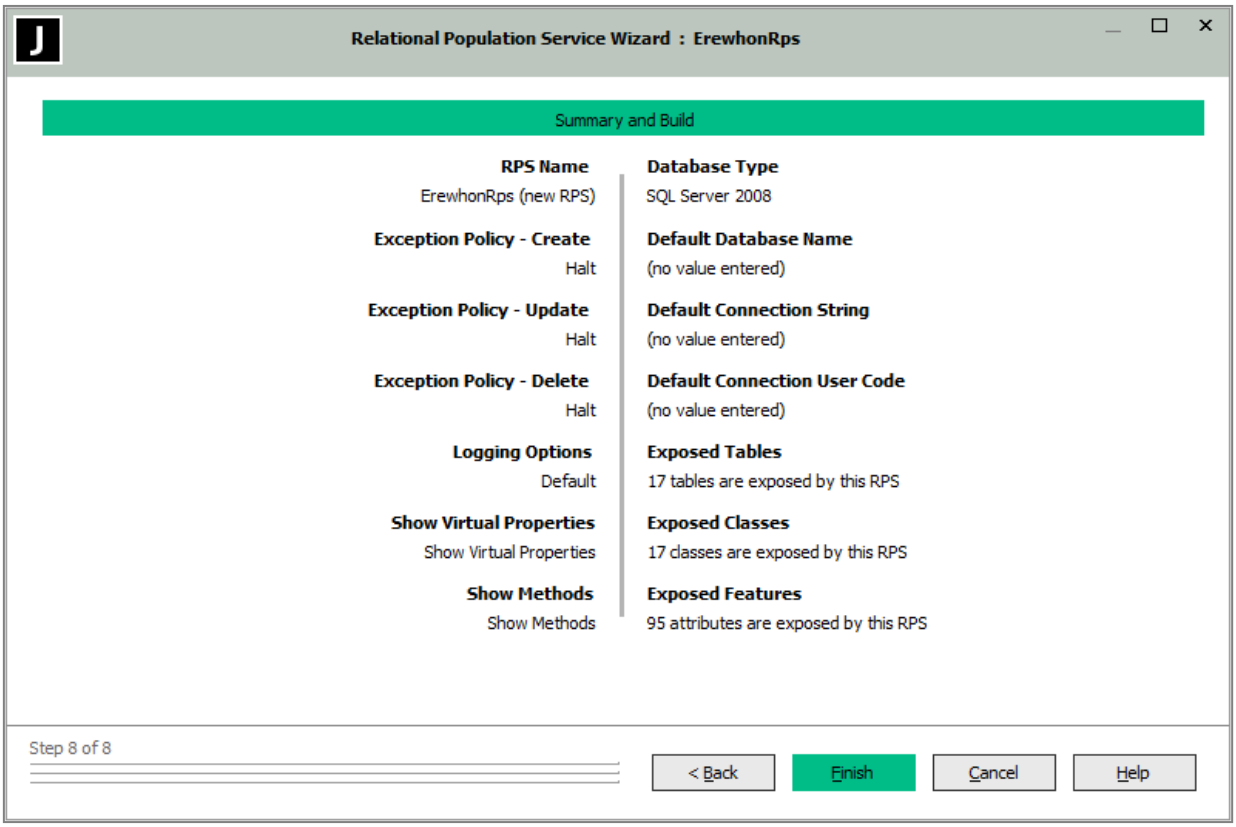
   c. Change or specify values in the **Mapping**, **ID**, and **Key** columns for the selected attribute or method to column map, if required.

8. To remove a selected column, right-click on that row and then select the **Remove Column** command from the popup menu that is displayed.

Building the RPS Mapping

The **Summary and Build** sheet of the Relational Population Service wizard enables you to view a summary of and build (generate) your RPS mapping.

An example of the **Summary and Build** sheet is shown in the following image.



This sheet summarizes the entire RPS mapping and enables you to build the RPS mapping.

Building the RPS mapping indicates that the mapping process is complete and it marks the status of the database mapping as ready for use. As this is the last step in the wizard, the **Next** button is replaced with a **Finish** button.

» To build your RPS mapping

- Click the **Finish** button when you have completed the RPS mapping. This generates the RPS mapping that can now be used by the RPS engine and it closes the Relational Population Service wizard.

When the build process is complete, the Relational Population Service Browser is then displayed with your new or updated RPS mapping. For an existing RPS mapping, the schema is versioned if it was not already versioned.

The RPS mapping can now be used by an RPS node. For details, see "Managing an RPS Node", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

Removing an RPS Mapping

From the Relational Population Service Browser, the **Remove** command from the Relational menu enables you to remove (delete) the RPS mapping that is currently selected.

Note You cannot remove an RPS mapping that is currently being accessed.

To remove an RPS mapping

1. In the Relational Population Service Browser, select the RPS mapping that you want to remove.
2. Select the **Remove** command from the Relational menu.
3. A message box is then displayed, to enable you to confirm that you want to remove the specified RPS mapping.
4. Click the **OK** button to confirm that the selected RPS mapping is to be removed.

   Alternatively, click the **Cancel** button to abandon the deletion.

The Relational Population Service Browser is then updated to reflect the removal of the selected RPS mapping. There may be a momentary delay while this updating occurs.

Changing an RPS Mapping

Changing an existing RPS mapping in the current schema versions the schema and the RPS mapping.

When you change an RPS mapping, the new RPS mapping is used when the reorganization is replayed on the RPS node.

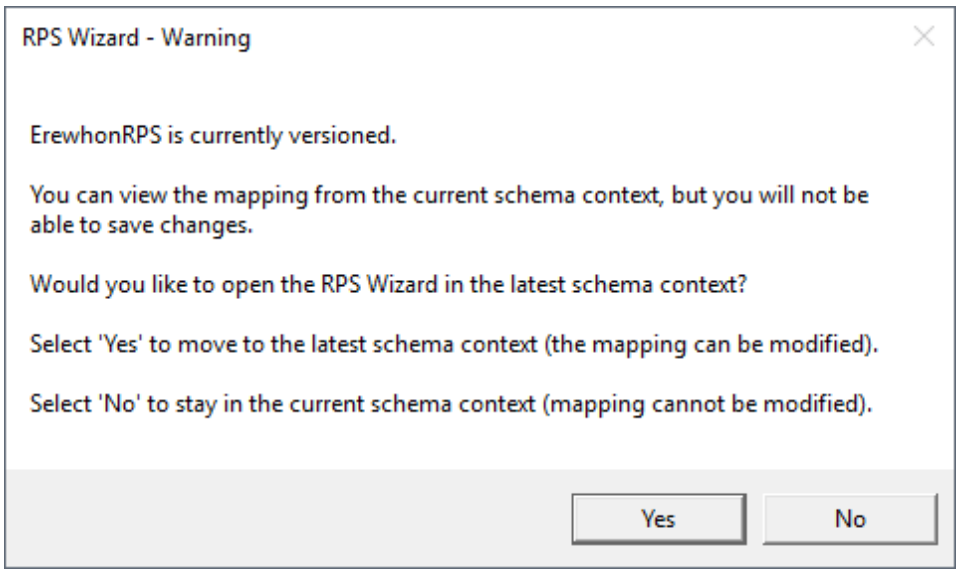
If you change the oid mapping in an existing RPS mapping to **Map to String (7.1 format)**, the database is reorganized and table alter scripts are generated.

Automatic versioning of the RPS mapping may occur when you add, change, or remove a Jade property or column-mapping method used in the RPS mapping.

To change an RPS mapping

- Select the **Change** command from the Relational menu.

If the RPS mapping is currently versioned and you are in the current schema context, the message box shown in the following image is then displayed.



The Relational Population Service wizard is then displayed. For details about the Relational Population Service wizard steps, see "Adding an RPS Mapping", earlier in this chapter.

When you are changing an existing RPS mapping, the text box in the first dialog of the Relational Population Service wizard displays the name of your RPS mapping.

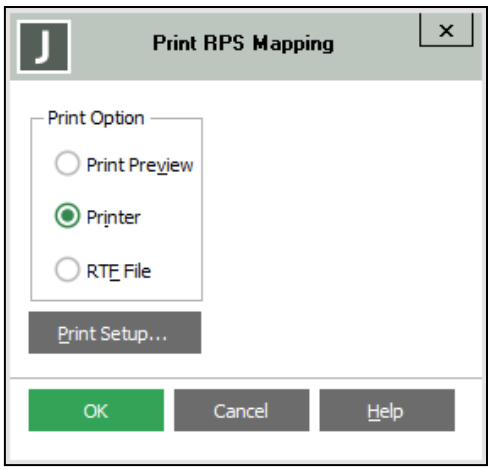
As you cannot change the name of an existing RPS mapping, the text box is disabled.

Printing an RPS Mapping

From the Relational Population Service Browser, the **Print** command from the Relational menu enables you to output information about the currently selected RPS mapping to your printer, your workstation monitor, or to a rich text format (.**rtf**) file.

To print your RPS mapping

1. In the Relational Population Service Browser, select the RPS mapping that you want to print.
2. Select the **Print** command from the Relational menu. The Print RPS Mapping dialog, shown in the following image, is then displayed.



3. In the Print Option group box, select the **Print Preview** option button to see a preview of your print output on your workstation monitor or select the **RTF File** option button to direct your print output to a rich text format (.**rtf**) file. By default, output is directed to your default printer.
4. Click the **Print Setup** button to change the printer to which output is directed and to change print setup details, if required. Print setup details are retained for subsequent print requests.
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

The Printing progress dialog is then displayed. (You can click the **Cancel** button from the Printing progress dialog to cancel your print request.)

Your print output contains details of your RPS mapping, and details of each of the RPS mapping tables and their columns.

Extracting an RPS Mapping

You can extract an RPS mapping as part of the schema in which it is defined or you can extract only the RPS mapping itself.

You cannot extract a schema from a system with the **Map to String (7.1 format)** option set and load it into a Jade 7.1.06 or earlier system where the OID mapping is **Map to String**.

To extract an RPS mapping only

1. In the Relational Population Service Browser, select the RPS mapping that you want to extract.
2. Select the **Extract** command from the Relational menu. The Extract dialog is then displayed.

   As you can extract only the version of the RPS mapping already selected in the RPS Browser, when extracting an RPS mapping, the **Multiple Schemas** option button and **Extract Latest** check box are disabled.

   By default, all entities (class maps) in the RPS mapping are extracted; that is, the **Extract All** option button in the Current Schema group box is selected.
3. Select the **Parameter File** option button if you want to use an existing parameter file to specify the selected class maps to extract. The **Parameter** sheet is then displayed. For details about using this sheet, see "Specifying Your Parameter File Options", in Chapter 10.

   Note If you select the **Parameter File** option, you must first have defined a parameter text file by using a text editor (for example, Notepad) or by saving the parameters from a selective extract.
4. Select the **Selective** option button if you want to select specific class maps to be extracted. The **Selective** sheet is then displayed. For details, see "Specifying Selective RPS Extract Options", in the following subsection.

   Any specified class maps that you select are for this extract only. You can save the selective extract information, if required, by using the **Save Parameters** check box in the **Selective** sheet.
5. Select the **Changes** option button if you want to extract only the changed class maps. The **Changes** sheet is then displayed. For details about using this sheet, see "Specifying Your Change Options", in Chapter 10.
6. If you want to extract form and data definitions in the legacy DDB format instead of the XML DDX format, uncheck the **Forms/Mappings as XML (ddx file)** check box. The value of this check box defaults to the value of the **Extract as DDX (xml format) instead of DDB** check box on the **Source Management** sheet of the Preferences dialog. By default, this check box is checked; that is, schemas are extracted as .**ddx** files.
7. In the **Schema File Name** and **DDX File Name** text boxes of the **Extract Options** sheet, specify the appropriate names and location of your .**scm** and .**ddb** or .**ddx** files, respectively.

   Note If the **Forms/Mappings as XML (ddx file)** check box is unchecked, this text box is named **DDB File Name** and the file suffix is .**ddb**.

   By default, your RPS mapping is extracted to your Jade working directory (for example, **s:\jade\test\bin**) as .**scm** and .**ddb** or .**ddx** files that are prefixed with the name of your RPS mapping; for example, **Documentation Example.scm** and **Documentation Example.ddx**.

   If you want to extract the RPS mapping to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

   An error is raised if you do not specify the name of a file, your specified file name or path is invalid, or an existing file cannot be accessed.
8. To specify UTF-8 encoding and compact DDX extract options, click the **Schema Options** tab. On the **Schema Options** sheet that is displayed, only the **UTF-8 Encoded** and **Compact DDX** check boxes are enabled. For details about using these check boxes, see "Specifying Your Extract Options", in Chapter 10.

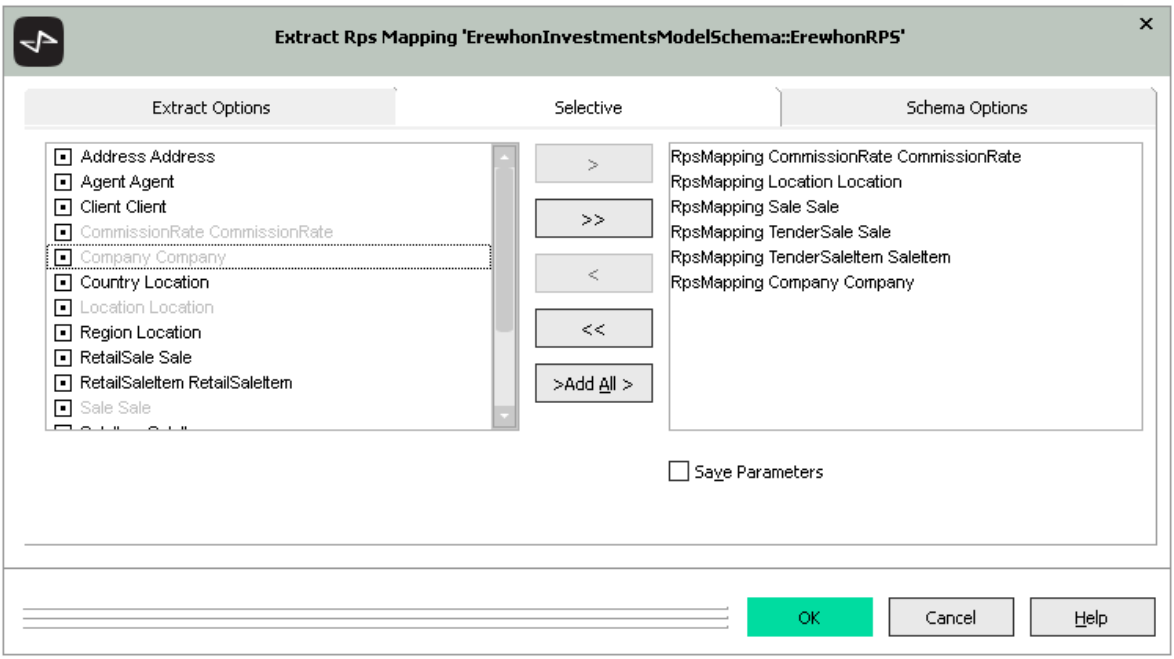
9. Click the **OK** button to confirm your selections, the **Parameter** sheet to specify your parameter option, the **Selective** sheet to specify your selective extract options, the **Changes** sheet to specify your RPS mapping change options, or the **Cancel** button to abandon your selections.

For details about schema extraction options, see "Extracting Your Schema", in Chapter 10.

Specifying Selective RPS Extract Options

The **Selective** sheet is displayed when you have selected the **Selective** option button in the **Extract Options** sheet and you then click the **Selective** sheet.

The **Selective** sheet, shown in the following image, is also displayed when you click the **Show Items on Selective Sheet** button on the **Changes** sheet if you are extracting patch version changes only.



The **Selective** sheet enables you to specify selected class maps that are to be extracted from your RPS mapping. (For details about using **Schema Options** sheet, see "Extracting an RPS Mapping", earlier in this chapter.)

All class maps in the RPS mapping are displayed in the list box at the left of the sheet by default, to enable you to make your selections. If the class is mapped to multiple tables in a class map, the *table-list* value contains all tables in that class map. If a class has subclasses mapped to a single table, each subclass mapping is listed separately.

To specify your RPS mapping selections

1. From the candidate class maps list box at the left of the sheet, select one or more class maps that you want to extract to a file. Use one of the following actions to select the required class maps.
   - Double-click on the appropriate class map.

     The selected class map is then displayed in the selected objects list box at the right of the sheet.
   - Click on a class map that you want to extract and then click the **>** button to move the selected class map to the selected objects list box.

- To move all displayed class maps to the selected objects list box, click the **>>** button or the **>Add All >** button. All class maps are then displayed in the selected objects list box at the right of the sheet.
- Use the Shift or Ctrl key to select a group of class maps and then click the **>** button to move the selected objects to the selected objects list box.

When class maps have been selected for extraction, they are disabled in the candidate objects list box.

Tip You can locate a specific class map in the candidate objects list box by pressing the F4 key. The Find Type dialog is then displayed, to enable you to specify or select the class map that you want to locate.

2. You can move class maps selected for extract from the selected objects list box back to the candidate objects list box, to exclude them from the extract.

   The actions required to do this are identical to those described in step 1, except that you use the **<** button to move single items and the **<<** button to move all items selected for extract back to the candidate objects list box.

3. Check the **Save Parameters** check box if you want your selected class maps to be saved to a parameter file.

   A parameter file can be reused for subsequent RPS mapping extracts to specify the class maps to extract. By default, the selections that you make are not saved in a parameter file.

4. When all class maps that you want to extract from your RPS mapping are displayed in the selected objects list box on the right of the sheet, click the **OK** button.

   If you are creating a parameter file, the common File dialog is then displayed, to enable you to select the location and name of your parameter file. By default, the file location is your Jade working directory and the file name is the RPS mapping name with an .**unl** extension; for example:

   ```
   ErewhonRPS.unl
   ```

For details about the parameter file, see "Parameter File Syntax" under "Specifying Your Parameter File Options", in Chapter 10.

Loading an RPS Mapping

You can load an RPS mapping as part of the schema in which it is defined or you can load only an extracted full or partial RPS mapping.

You cannot extract a schema from a system with the **Map to String (7.1 format)** option set and load it into a Jade 7.1.06 or earlier system where the OID mapping is **Map to String**. (As it may break your RPS mapping, you must upgrade the system into which you want to load the schema.)

Notes Loading a partial RPS mapping updates only the class maps in the RPS mapping that have been included in the extract file.

When no reorganization is required after the .**scm** file load, the .**ddb** or .**ddx** file can be loaded (as normal) before the reorganization is performed. If a reorganization is required after the .**scm** file load and before the .**ddb** or .**ddx** file load, the .**ddbx** or .**ddxx** file can be loaded before the reorganization, to keep the RPS mappings up-to-date with the schema changes. After the reorganization, the .**ddb** or .**ddx** file is loaded as normal, to complete the changes. When extracting a schema with an RPS mapping and the:

- DDX format style is selected, a separate .**ddxx** file is produced (the contents of this file are in XML format).
- DDB format style is selected, a separate .**ddbx** file is produced.

See "Before You Get Started", in the *Schema Load User's Guide* for details about the order in which to load separate RPS mapping extract files and selective schema extract files into a deployed database.

To load an RPS mapping

- Select the **Load** command from the Relational menu.

Note The Load Options dialog is then displayed.

For details about using the Load Options dialog, see "Loading Your Schema", in Chapter 10.

Creating an Exclude Command File

You can create an exclude JCF file (Jade command file) for a selected RPS mapping. You can then use this file, which includes all excluded table and columns for the RPS mapping, as input to the batch Jade Schema Load (**jadloadb**) executable, the **JadeSchemaLoader** application in **jadclient**, **jade**, or the **Application** class **startApplicationWithParameter** method so that all excluded RPS mapping tables and columns are reapplied to the RPS mapping.

Reapplying the exclusions causes the specified schema and RPS mapping to be versioned and a reorganization is required.

For details about:

- Excluding tables and columns from the RPS mapping, see "Mapping Classes to Tables" and "Maintaining RPS Column Mappings", earlier in this chapter.
- Customizing an RPS mapping for a specific site if your Jade applications are deployed to multiple sites with different Business Intelligence requirements, see "Site-Specific RPS Mapping Customization", in Chapter 2 of the *Synchronized Database Service (SDS) Administration Guide*.

To create an exclude JCF file for an RPS mapping

1. In the Relational Population Service browser, select the RPS mapping whose excluded tables and columns you want to output to a command file.
2. Select the **Create Jcf** command from the Relational menu.

The Select Jcf File command dialog is then displayed, to enable you to select the directory and to specify the name of your file (for example, **TestSiteExclusions.jcf**).

When you click the **Save** button, the JCF file is created.

Chapter 16 Importing External Components

This chapter covers the following topics.

- Overview
- Maintaining ActiveX Control and Automation Server Objects
  - Importing ActiveX Control Libraries and Automation Libraries
  - Changing the Generated Schema
  - Removing an ActiveX Type Library
  - Extracting and Loading ActiveX Schema Definition Data
  - Registering an ActiveX Server
  - Unregistering an ActiveX Server
- Maintaining .NET Objects
  - Importing .NET External Component Libraries
  - Changing the Generated .NET Schema
  - Removing a .NET Library
  - Extracting and Loading .NET Schema Definition Data
- Maintaining OpenAPI Objects
  - Importing OpenAPI Specifications
  - Removing an OpenAPI Specification
- Displaying All Superschema External Components in Your Current External Components Browser

Overview

The Jade Platform development environment provides the External Components Browser, which is accessed by selecting the **External Component Libraries** command from the Browse menu.

The External Components Browser contains three sheets: the **ActiveX** sheet, the **.NET Framework** sheet, and the **OpenAPI** sheet, to enable you to import and maintain ActiveX controls and automation server library objects, .NET framework library objects, and OpenAPI specifications, respectively.

Each sheet of the Browser contains a list of all Active X control and automation server libraries, OpenAPI specifications, or .NET libraries that were previously imported into the current schema.

If you have not yet imported an ActiveX type, OpenAPI specification, or .NET library, nothing is displayed in the respective sheet of the External Components Browser. Only one External Components Browser for the current schema can be open at any time. However, you can have concurrent open External Components Browsers for different schemas in a Jade development session.

You can change your default browser options, if required, by using the **Browser** sheet, accessed from the Options menu **Preferences** command.

Jade does not support the creation of ActiveX, OpenAPI, and .NET assemblies without a default constructor. If the constructor method of a class expects a parameter, exception 14577 is raised, stating that the constructor could not be found.

If you cannot modify the class, consider writing an ActiveX, OpenAPI, or .NET wrapper class. For example, the following class definition does not have a default constructor.

```
DotNetClass
{
     public DotNetClass(int arguments)
     {
     }
}
```

When you attempt to run the **JadeDotNetType** class **createDotNetObject** method after importing this example .NET assembly into Jade, exception 14577 is raised. If you have been supplied with only the assembly and not the code, you can create a wrapper method, as shown in the following example.

```
using AssemblyName.DotNetClass;
    public class DotNetClassWrapper
    {
        DotNetClass dnc;
        int passInt=0;
        public void setDotNetClass(int x)
        {
            passInt = x;
        }
        public void createDotNetClass()
        {
            dnc = new DotNetClass(passInt);
        }
        public DotNetClass getDotNetClassObj()
        {
            if(dnc != null)
                return dnc;
            else
                return null;
        }
    }
```

Compile the wrapper method, linking it to the **DotNetClass** assembly, before importing the .NET wrapper assembly and access the **DotNetClass** class as follows.

```
vars
    dnc        : DotNetClass;
    dncWrapper : DotNetClassWrapper;
begin
    create dncWrapper transient;
    dncWrapper.createDotNetObject;
    dncWrapper.setDotNetClass(10.Integer);
    dncWrapper.createDotNetClass();
    dnc := dncWrapper.getDotNetClassObj().DotNetClass;
epilog
    delete dncWrapper;
end;
```

Maintaining ActiveX Control and Automation Server Objects

When the **ActiveX** sheet of the External Components Browser has focus, the Components menu ActiveX submenu provides the commands listed in the following table that enable you to maintain ActiveX control and automation server libraries.

Menu Command	For details, see …	Action
Import	Importing ActiveX Control Libraries and Automation Libraries	Displays the ActiveX Import Wizard, which enables you to import a control or automation type library (accessed by selecting the **Control Library** or the **Automation Library** submenu command)
Remove	Removing an ActiveX Type Library	Removes the ActiveX type library selected in the External Components Browser
Extract	Extracting and Loading ActiveX Schema Definition Data	Extracts the ActiveX type library selected in the External Components Browser
Register Server	Registering an ActiveX Server	Displays the common File dialog, to enable you to select the ActiveX server that you want to register
Unregister Server	Unregistering an ActiveX Server	Removes the selected ActiveX server from the operating system registry

In addition, you can use the **Superschema** command from the View menu to view all ActiveX control and automation server libraries imported into superschemas in the External Components Browser of your current schema. For details, see "Displaying All Superschema External Components in Your Current External Components Browser", later in this chapter.

For details about the actions that you can perform from the **ActiveX** sheet of the External Components Browser, see the following subsections.

Importing ActiveX Control Libraries and Automation Libraries

ActiveX type libraries are available for import only after they have been registered in the operating system registry of the workstation into which they are being imported, under the key **HKEY_CLASSES_ROOT\typelib**. For details, see "Registering an ActiveX Server", later in this chapter.

For details about the default names generated by the ActiveX Import Wizard, see "Default Names" under "ActiveX Default Values and Considerations", in Chapter 4 of the *External Interface Developer's Reference*.

Before you can use an ActiveX type library registered in the operating system registry for your workstation, you must first import the definition of the ActiveX object into Jade. The ActiveX object definitions are contained in *type libraries*.

A type library can be a stand-alone file, usually with a .**tlb** extension, or it can be imbedded within the executable module (that is, the executable or library file). A type library can describe one object or more typically, multiple objects.

Notes The ActiveX Import Wizard is a wizard-style dialog that consists of five steps, each represented by a sheet of the dialog. Use the **Next >** and **< Back** buttons to navigate forwards or backwards through the steps. No step is enabled until the previous step has been completed.

If a parameter does not specify [**in**], [**out**], or [**in**, **out**] and it is a reference when you import an ActiveX type library, Jade sets the parameter to [**in**, **out**].

To import an ActiveX type library

1. Select the **ActiveX** command from the Components menu. The Import submenu is then displayed.
2. Select the type library that you want to import, as follows.
   - Select the **Control Library** command if you want to import an ActiveX control object (a user interface object that implements a number of interfaces that support its use on forms).
   - Select the **Automation Library** command if you want to import an ActiveX automation object.

   Both the **Control Library** and the **Automation Library** submenu commands invoke the ActiveX Import Wizard. The differences between these commands are the libraries that are displayed in the initial list of ActiveX type libraries and some minor differences in the code that is generated.

   In addition, ActiveX control objects are generated as subclasses of the **ActiveXControl** class and ActiveX automation objects are generated as subclasses of the **ActiveXAutomation** class.

The ActiveX Import Wizard is then displayed. The ActiveX Import Wizard systematically guides you through the import process. When all of the information has been gathered, the wizard generates all of the required classes, properties, and methods that you need to access the ActiveX object from Jade.

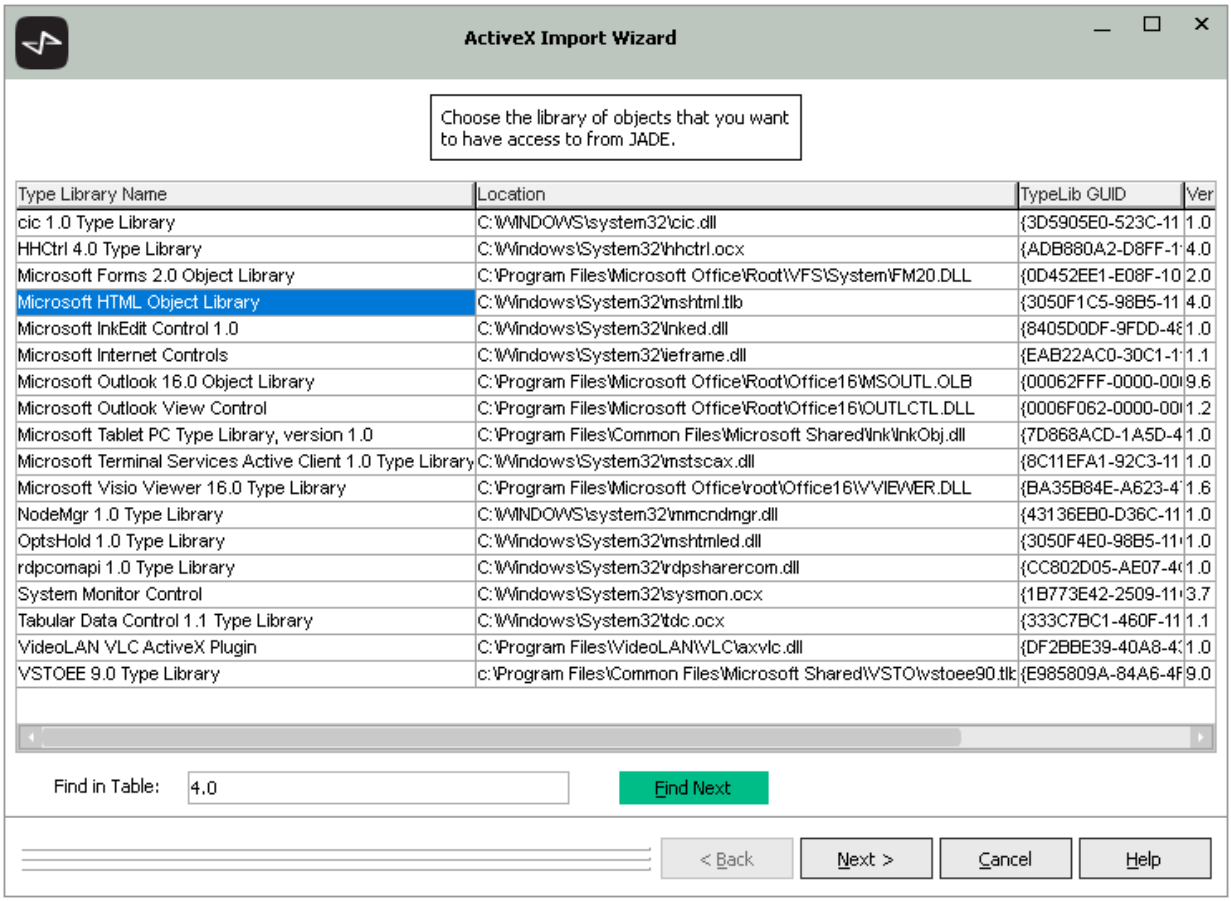
Sheets that have a table have a **Find in Table** text box and a **Find Next** button below the table, to enable you to search for text in the displayed table. Entering text in the **Find in Table** text box and then clicking the **Find Next** button searches the displayed table for the specified text. The search is case-insensitive.

If the search string is found within a cell, the cell is then displayed with red text and the font increased in size to highlight the located entry. Click the **Find Next** button again to search for the next cell with that text. If the search text is not found in subsequent cells, a message box is displayed.

Changing the text or pressing the **Next** or **Back** button causes the next search to start from the beginning of the cells in the table.

Selecting Your ActiveX Type Library

When you have selected the **Control Library** or **Automation Library** submenu command, the first sheet of the ActiveX Import Wizard, shown in the following image, is then displayed.



This sheet enables you to select the ActiveX type library that you want to import. (The type libraries displayed in the **Type Library Name** column box are determined by the **Import** submenu command that you selected; that is, the **Control Library** or the **Automation Library** command.)

The location of each of the type libraries registered in the operating system registry for your workstation is displayed in the **Location** column.

When importing control objects, the **Type Library Name** column includes all libraries that contain at least one control or libraries marked as being control libraries in the operating system registry. When importing automation objects, the **Type Library Name** column contains all libraries that are not marked as control libraries in the operating system registry.

Note A library that is not marked as a control library can also contain controls. If a library contains both controls and automation objects and you require access to both types of objects from Jade, you must import the library twice; once as a control library and again as an automation library.

To select the type library that you want to import

1. In the **Type Library Name** column, select the ActiveX type library whose ActiveX object definition you want to import into Jade. If the type library that you want to import is not listed, you must first register that library in the operation system registry of your workstation. For details, see "Registering an ActiveX Server", later in this chapter.

   For details about searching for text in a table, see "Importing ActiveX Control Libraries and Automation Libraries", earlier in this chapter.

2. Click the **Next >** button.

When you click the **Next >** button, the ActiveX Import Wizard then enables you to change the default name assigned by Jade for your abstract class.

Specifying a Name for Your ActiveX Type Library

When you have selected the ActiveX type library that you want to import, the next sheet of the ActiveX Import Wizard is then displayed. This sheet enables you to specify the Jade class name that you require for the type library that is imported. When the ActiveX object is imported, Jade creates an abstract class of this name that represents the library.

If you import an ActiveX control type library, the abstract class is created as a subclass of the **ActiveXControl** class. If you import an automation type library, the abstract class is created as a subclass of the **ActiveXAutomation** class. The class that is created becomes the superclass for all classes that are subsequently generated corresponding to objects in the imported library.

To specify a name for your ActiveX object

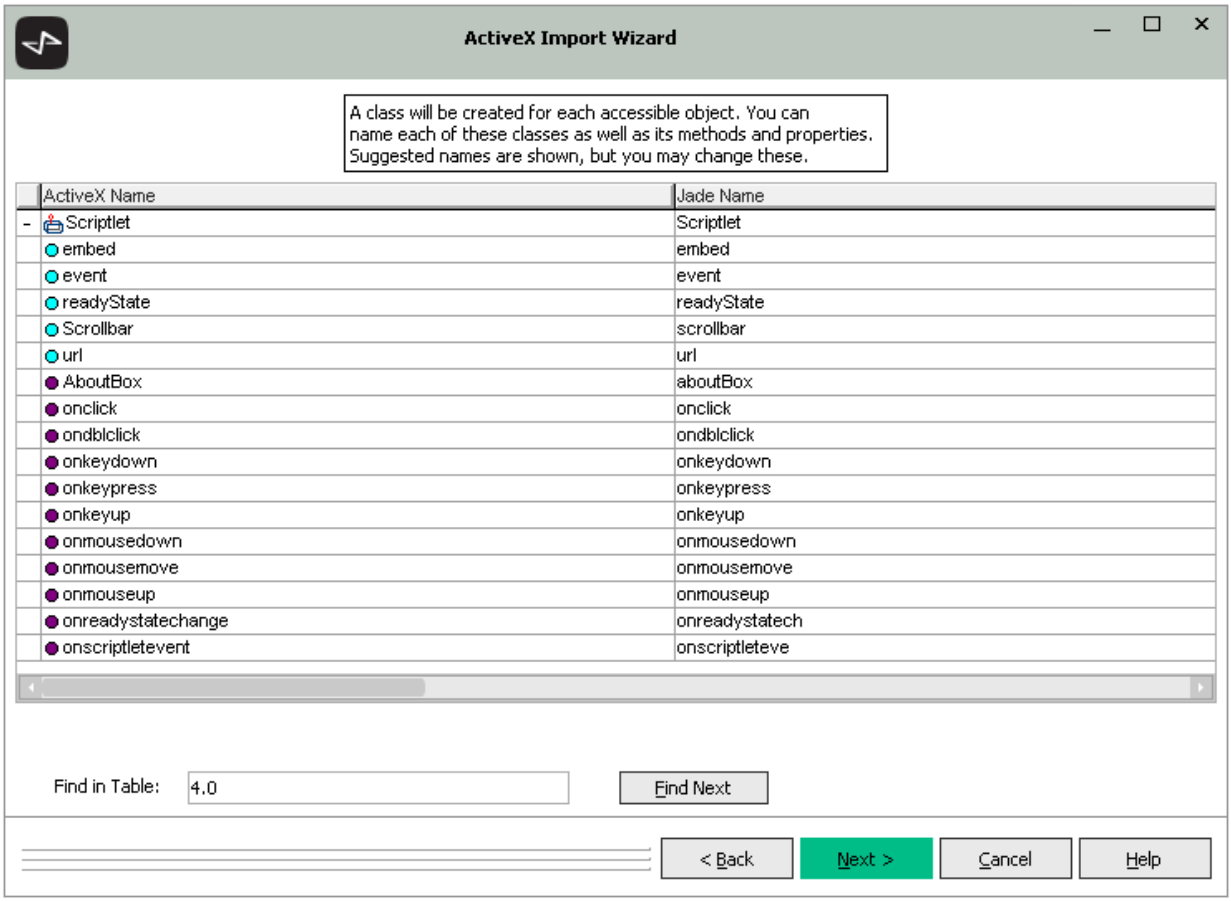
1. In the **Library Name** text box, specify the name that you want created for the abstract class if you do not want the default name assigned by Jade. (The name must be a valid name unique to the current Jade schema branch. Jade converts the first character to uppercase, if required.)

2. Click the **Next >** button when you have specified your ActiveX object class name. Alternatively, click the **Cancel** button (or press the Esc key) to abandon the ActiveX object import process.

When you click the **Next >** button, the ActiveX Import Wizard then enables you to change the default names assigned by Jade for each of the classes that are created for each object in the imported library that you can access, as well as the methods and properties provided by these classes.

Naming Object Classes in the ActiveX Type Library

When you have specified the name to be created for the abstract ActiveX object, the next sheet of the ActiveX Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify names for each of the classes that are created for each object that you can access in the imported library. You can also specify names for the methods and properties provided by instances of these classes, if required.

The ActiveX name for each of the objects that you can access in the imported library is displayed in the **ActiveX Name** column and the default name supplied by Jade for each object is displayed in the **Jade Name** column.

An ActiveX object is displayed in the list only when its **CLSID** is registered in the system registry under the key **HKEY_CLASSES_ROOT\CLSID**. Jade creates a class for each object as a subclass of the class that was created corresponding to the imported type library. (For details, see "Specifying a Name for Your ActiveX Type Library", earlier in this chapter.)

The icons in the left column of the table on this sheet are listed in the following table.

Icon	Description
+	Collapsed node, which you can expand to see the properties and methods of the class by double-clicking the object or clicking the plus sign (**+**)

Icon	Description
–	Expanded node, which you can collapse to hide the properties and methods of the class by double-clicking the object or clicking the minus sign (**-**)
	ActiveX object
	Method (represented by a purple-filled circle)
	Attribute or reference property (represented by a light blue-filled circle)

To specify names for each of the created classes

1. In the **Jade Name** column, specify the name that you want created for the object class if you do not want the default name assigned by Jade. The name must be a valid name unique to the schema branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to uppercase, if required.

   Note Help text may be displayed below the table. This help text from the type library describes the currently selected table row. If you have expanded an ActiveX object to display its methods and properties, the help text describes the selected method or property.

2. If an ActiveX object contains methods or properties (that is, it is displayed as a collapsed node), expand that node to display all methods and properties for that object if you want to specify names for methods or properties that differ from the default names assigned by Jade. The method and property names must be valid names unique in the class hierarchy branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to lowercase, if required.

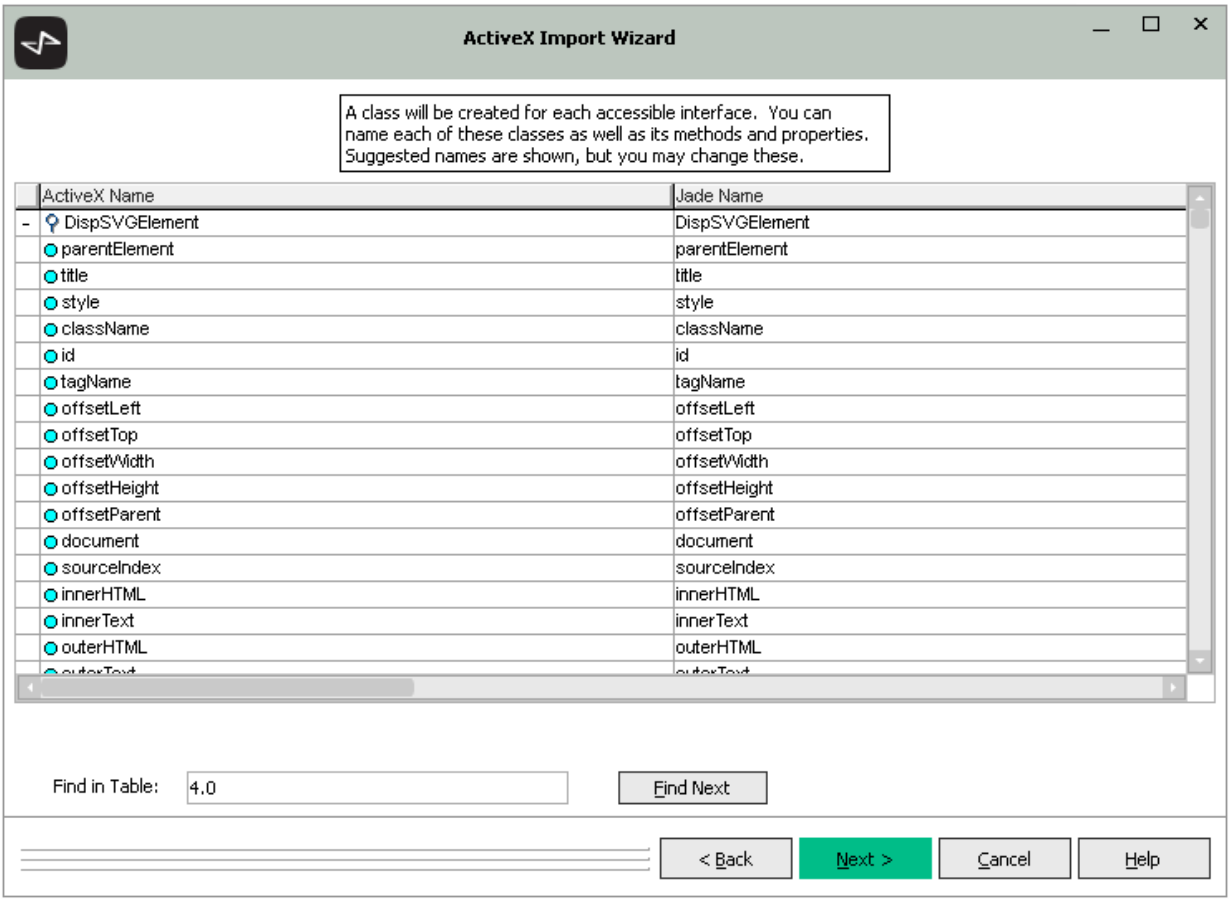
   For details about searching for text in a table, see "Importing ActiveX Control Libraries and Automation Libraries", earlier in this chapter.

3. Click the **Next >** button when you have specified your Jade class names and any method or property names that you want to change from the default Jade values. Alternatively, click the **Cancel** button (or press the Esc key) to abandon the ActiveX object import process.

When you click the **Next >** button, the wizard then enables you to change the default names assigned by Jade for each interface in the imported ActiveX library that you can access.

Naming Interfaces in the ActiveX Type Library

When you have specified the names to be created for the each of the created classes in the ActiveX object and optionally their methods and properties, the next sheet of the ActiveX Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify names for each of the interfaces in the imported ActiveX library that you can access. You can also specify names for the methods and properties provided by instances of these interfaces, if required.

The ActiveX name for each of the interfaces that you can access in the imported library is displayed in the **ActiveX Name** column and the default name supplied by Jade for each interface is displayed in the **Jade Name** column.

All ActiveX objects are accessed using an interface and each object may (and usually has) more than one interface. The ActiveX Import Wizard displays only the **dispInterfaces**. (For details, refer to your COM documentation.) Jade creates a subclass of the **IDispatch** class for each interface class that it generates.

The icons displayed at the left of a row are listed in the following table.

Icon	Description
+	Collapsed node, which you can expand by double-clicking the object or clicking the plus sign (**+**)
–	Expanded node, which you can collapse by double-clicking the object or clicking the minus sign (**-**)

Icon	Description
	ActiveX interface
	Method (represented by a purple-filled circle)
	Attribute or reference property (represented by a light blue-filled circle)

To specify names for each of the classes created for ActiveX interfaces

1. In the **Jade Name** column, specify the name that you want created for the subclass of the **IDispatch** class if you do not want the default name assigned by Jade. The name must be a valid name unique to the schema branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to uppercase, if required.

   Note Help text may be displayed below the table. This help text is from the type library, and describes the table row that you have currently selected. If you have expanded an ActiveX object to display its methods and properties, the help text describes the selected method or property.

2. If an ActiveX interface contains methods or properties (that is, it is displayed as a collapsed node), expand that node to display all methods and properties for that interface if you want to specify names that differ from the default names assigned by Jade. The method and property names must be valid names unique in the class hierarchy branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to lowercase, if required.

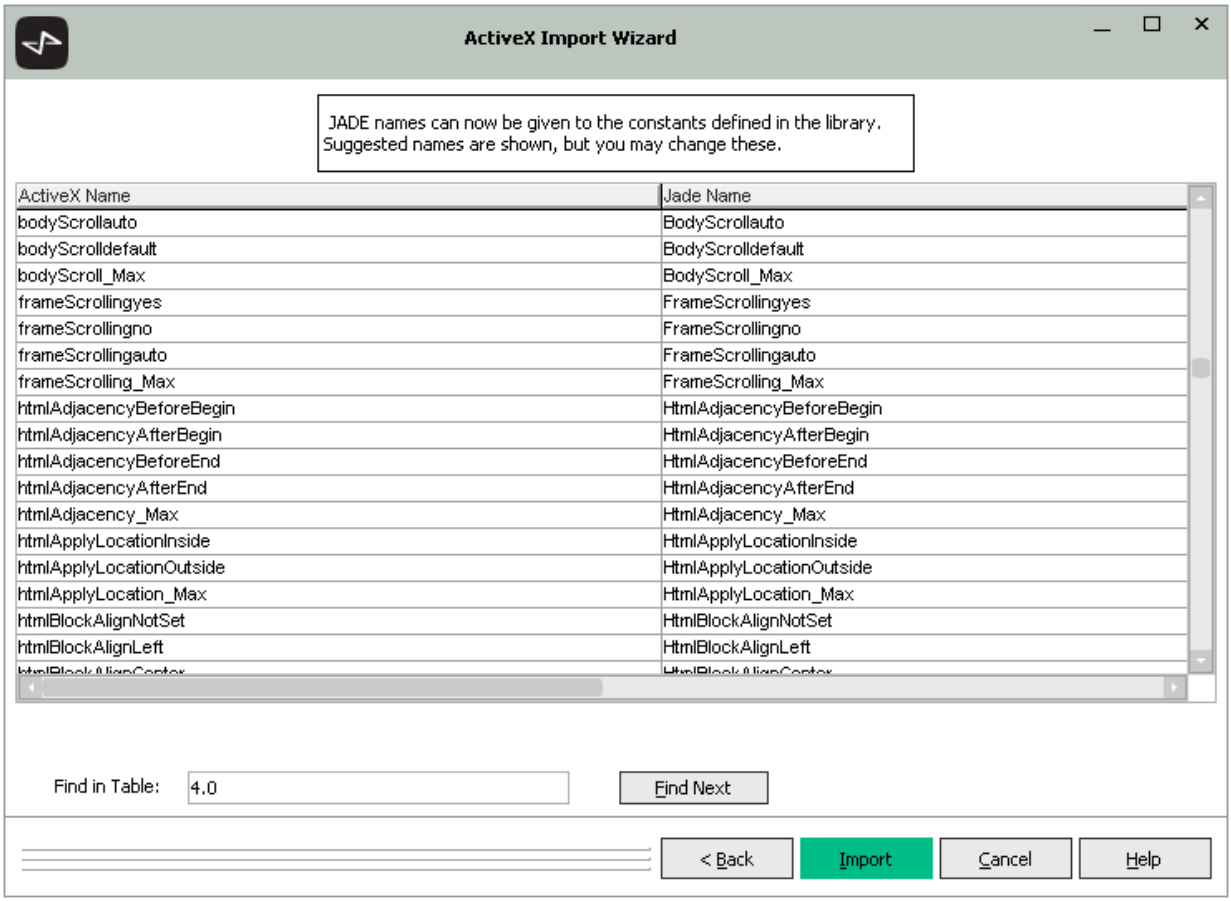
   For details about searching for text in a table, see "Importing ActiveX Control Libraries and Automation Libraries", earlier in this chapter.

3. Click the **Next >** button when you have specified your Jade interface class names and any method or property names that you want to change from the default Jade values. Alternatively, click the **Cancel** button (or press the Esc key) to abandon the ActiveX object import process.

When you click the **Next >** button, the ActiveX Import Wizard then enables you to change the default names assigned by Jade for each constant in the imported ActiveX library, if required.

Naming Constants in the ActiveX Type Library

When you have specified the names to be created for the each of the interfaces in the ActiveX object, the next sheet of the ActiveX Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify names for each constant in the imported ActiveX library. The ActiveX name for each of the constants in the type library is displayed in the **ActiveX Name** column, and the default name supplied by Jade is displayed in the **Jade Name** column. All ActiveX constants are integer values that are created in the **ActiveXControl** or **ActiveXAutomation** subclass.

To specify names for each of the constants created for ActiveX object classes

1. In the **Jade Name** column, specify the name that you want created for the class constants if you want a name that differs from the default name assigned by Jade. The name must be a valid name unique to the schema branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to uppercase, if required.

   For details about searching for text in a table, see "Importing ActiveX Control Libraries and Automation Libraries", earlier in this chapter.

2. To confirm that you have finished specifying names for your ActiveX object classes, methods, properties, and constants, click the **Import** button. Alternatively, click the **< Back** button to redisplay the previous sheet, or the **Cancel** button (or press the Esc key) to abandon the ActiveX object import process.

When you click the **Import** button, Jade starts the ActiveX library import process. Large libraries may take several minutes to import. (For example, the automation server library for Microsoft Word takes over three minutes on a 400M Hz Pentium II.) Any errors (for example, naming conflicts) that are detected during the import process abort the load. A message is displayed, and the class, method, property, or constant that is in error is highlighted on the appropriate sheet of the wizard, to enable you to amend the value before restarting the import process.

The ActiveX Import Wizard is closed, and the External Components Browser then contains your new ActiveX object.

You can now use your ActiveX classes as you can any other Jade class. For details, see "Using Generated ActiveX Classes", in Chapter 4 of the *External Interface Developer's Reference*.

When you import ActiveX object definitions into Jade, the actions that are taken are determined by the type of object; that is, an ActiveX interface, control, or automation class. For details, see the following subsections. For details, see "How Jade Imports ActiveX Object Definitions", in Chapter 4 of the *External Interface Developer's Reference*.

Changing the Generated ActiveX Schema

During the import process, names for method parameters are generated without you being able to override these method parameter names. For details about the default names generated by the ActiveX Import Wizard, see "Default Names" under "ActiveX Default Values and Considerations", in Chapter 4 of the *External Interface Developer's Reference*.

To change method parameter names

- Use the Methods List of the Class Browser to select the appropriate method and make the changes that you require to method parameters.

You can change the parameter name only. Do not change the parameter type.

To change class, method, or property names

- Use the Class, Methods, or Properties List of the Class Browser to change the name of the element that you want to change, if required.

You can change the element name only. Do not change the element type.

You should not add new classes to an ActiveX object or interface. In addition, although it is not recommended that you remove classes from an ActiveX object or interface, you *can* do so safely. For example, if you require only a subset of the features provided by an ActiveX type library, you *could* delete the unnecessary interfaces and objects.

Caution Although you can change the name of methods, properties, and parameters, do *not* change the types of these.

Removing an ActiveX Type Library

From the External Components Browser, the **Remove** command in the ActiveX submenu of the Components menu enables you to remove (delete) the currently selected ActiveX type library from your Jade database.

Note You cannot remove an ActiveX type library when any references exist to classes belonging to that library.

This action removes the selected ActiveX type library only from your Jade database. To remove the type library from the Windows operating system registry of your workstation, select the **Unregister Server** command from the ActiveX menu. (For details, see "Unregistering an ActiveX Server", later in this chapter.)

To remove an ActiveX type library

1. In the External Components Browser, select the ActiveX type library that you want to remove.
2. Select the **Remove** command from the ActiveX submenu of the Components menu. A message box is then displayed, to enable you to confirm that you want to remove the selected ActiveX type library.
3. Click the **OK** button to confirm that you *do* want to delete the selected ActiveX type library. (Alternatively, click the **Cancel** button to abandon the deletion.)

The External Components Browser is then updated to reflect the removal of the selected ActiveX type library. There may be a momentary delay while this updating occurs.

When you delete an ActiveX type library, Jade performs the following actions.

- Removes the selected type library.
- Removes all associated classes; that is, all ActiveX object classes that are subclasses of the **ActiveXAutomation** or **ActiveXControl** class.
- Removes all interfaces used by the type library.

Caution Although you can remove ActiveX object classes and interfaces from the Schema Browser, it is not recommended as it results in the type library remaining partially defined in Jade. If you later wanted to add the classes that you deleted, you must first completely remove the type library and then import that library again.

Extracting and Loading ActiveX Schema Definition Data

You can extract or load an ActiveX type library as part of the schema into which it is imported or you can extract only the ActiveX type library itself. Generated classes, methods, properties, and constants are extracted as part of a full schema extraction.

As no persistent instances of interfaces or automation objects are created, no instance data is extracted or loaded. ActiveX controls are treated like all other controls during the extract and load processes.

To extract an ActiveX type library only

1. In the External Components Browser, select the type library that you want to extract.
2. Select the **Extract** command from the ActiveX submenu of the Components menu.

The Extract dialog is then displayed, to enable you to select the options that you require. For details about using the Extract dialog, see "Extracting Your Schema", in Chapter 10. (You can only extract the entire .NET library selected in the current schema.)

Note When you extract an ActiveX type library, its associated classes, methods, properties, and constants are also extracted.

For details about extracting or loading the ActiveX type library as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10.

Registering an ActiveX Server

Before you can import an ActiveX control library or automation library, that type library must be registered in the operating system registry of the workstation into which it is being imported.

To register an ActiveX server

1. Select the **Register Server** command from the ActiveX submenu of the Components menu when the External Components Browser is the current window.

   The common file Open dialog is then displayed, to enable you to locate the ActiveX type library object. (The file suffix can be .**dll**, .**exe**, or .**ocx**, for example.)

2. When you have located the required file, click the **OK** button to register the ActiveX server with your Windows operating system registry.

You can now import the ActiveX object contained in the registered type library into your Jade Platform development environment.

Unregistering an ActiveX Server

You can unregister any ActiveX server from the Windows operating system registry from within Jade. You can unregister any ActiveX server, regardless of whether you have imported it into Jade or not.

To unregister an ActiveX server

1. Select the **Unregister Server** command from the ActiveX submenu of the Components menu when the External Components Browser is the current window. The common file Open dialog is then displayed, to enable you to locate the of the ActiveX type library object. (The file suffix can be .**dll**, .**exe**, or .**ocx**, for example.)

2. When you have located the required file, click the **OK** button to unregister the ActiveX type library from your Windows operating system registry.

The ActiveX server that you unregistered is removed only from the Windows operating system registry of your workstation. As the Jade database itself is unchanged, you can use this feature to remove a type library from your registry when you want the imported ActiveX object retained in the Jade database; for example, so that you can ship the database to a third-party site at which the ActiveX type library is registered.

For details about removing an ActiveX type library from your Jade database, see "Removing an ActiveX Type Library", earlier in this chapter.

Maintaining .NET Objects

The .NET Import wizard enables you to import a .NET assembly of .NET classes and their members (methods, properties, and so on) into Jade.

A .NET *assembly* is a **.dll** or **.exe** file with extra .NET header information (the manifest) that describes the additional class methods and properties in that .NET object.

Note In this release, Jade supports only in-process assemblies; that is, assemblies defined in Dynamic Link Libraries (DLLs).

When the **.NET Framework** sheet of the External Components Browser has focus, the Components menu .NET submenu provides the commands listed in the following table that enable you to maintain .NET external object libraries.

Menu Command	For details, see …	Action
Import	Importing .NET External Component Libraries	Displays the .NET Import Wizard, which enables you to import a .NET library (accessed by selecting the **.NET** submenu command)
Remove	Removing a .NET Library	Removes the .NET library selected in the External Components Browser
Extract	Extracting and Loading .NET Schema Definition Data	Extracts the .NET library selected in the External Components Browser

In addition, you can use the **Superschema** command from the View menu to view all .NET libraries imported into superschemas in the External Components Browser of your current schema. For details, see "Displaying All Superschema External Components in Your Current External Components Browser", later in this chapter.

For details about the actions that you can perform from the **.NET Framework** sheet of the External Components Browser, see the following subsections.

Importing .NET External Component Libraries

The selection phase of the .NET Import Wizard enables you to specify the appropriate Jade names to the items being imported.

When using .NET external component libraries:

- The import process can optionally import the super-classes referred to by classes in the assembly when the super-classes are in other assemblies.
- Re-importing an assembly deletes entities that are no longer defined in the assembly.
- If a class is re-parented in the assembly, the re-import process moves the existing class to the new parent.

When including superclasses in other assemblies, note the following.

- Moving a class to a new parent is possible only under strict rules.

  If the moving of a class fails, the **jommsg log** will contain a description of the reason for the failure. To achieve the move, it may be necessary to change the existing system before the move can be performed; for example, if the method definition of the manually added method clashes with a method in the new parent hierarchy.

- When an assembly is re-imported and the previous import used a different import philosophy of superclass classes (that is, Jade 2016 and earlier releases effectively used a check box value of **false**), you may need to change existing Jade logic.

  A change in the import style means that method signatures can be different for parameters and return types. The method may be renamed and existing application logic may not compile because the method has been removed (and re-added under another name), or parameters or return types are different.

- If two different imported assemblies refer to the same superclasses from another assembly, each import will have its own set of differently named superclasses.

  They are not shared between the two assembly hierarchies; the shared structure is not possible to achieve because the root class for the assembly holds global constants for the assembly.

- When importing classes from the **System:Windows:Forms** assembly, the classes are split into two separate parts in the hierarchy: **JadeDotNetType** and **JadeDotNetVisualComponent**, according to whether or not the class is a control.

  The superclasses of **System.Windows.Forms.Control** (**System::MarshalByRefObject** and **System.Windows.Forms.Component**) are not included in the Jade control hierarchy because the classes are not controls. However, the properties and methods for these classes are also added to the **Control** class, so they *are* available.

When you have imported the .NET assembly, Jade classes, methods, properties, and constants are generated for the items selected from the assembly. For details about the abstract grouping class names, see "Abstract Grouping Classes", in Chapter 4 of the *External Interface Developer's Reference*.

Note The .NET Import Wizard is a wizard-style dialog that consists of five steps, each represented by a sheet of the dialog. Use the **Next >** and **< Back** buttons to navigate forwards or backwards through the steps. No step is enabled until the previous step has been completed.

.NET objects are generated as subclasses of the non-GUI **JadeDotNetType** and the GUI **JadeDotNetVisualComponent** classes.

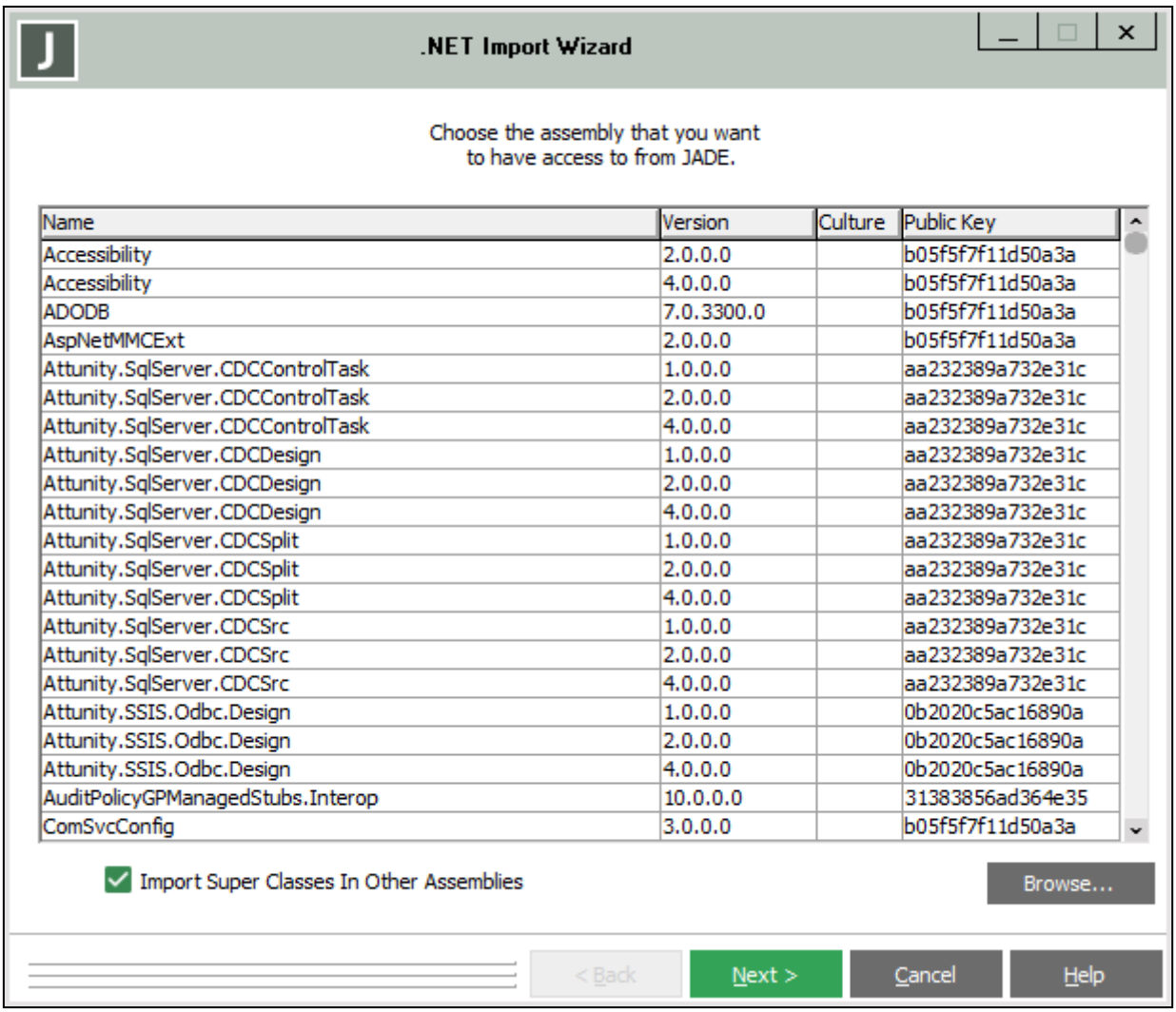
To import a .NET assembly

1. Select the **.NET** command from the Components menu. The .NET submenu is then displayed.
2. Select the **Import** command from the .NET submenu of the Components menu.

The .NET Import Wizard, which is then displayed, systematically guides you through the import process. When all of the information has been gathered, the wizard generates all of the required classes, properties, and methods that you need to access the .NET object from Jade.

Selecting Your .NET Assembly

The first sheet of the .NET Import Wizard is shown in the following image.



This sheet lists all .NET assemblies located in the Windows Global Assembly Cache (GAC) on your workstation.

Note Jade accesses the GAC on the client node (that is, the application server when you are running in thin client mode) during import. At run time, if the object is used on a presentation client (for example, a control), that object must also be in the presentation client's GAC.

To select the .NET assembly that you want to import

1. In the **Name** column, select the .NET assembly you want to import into Jade.

   Alternatively, if you want to import a .NET assembly from another location, click the adjacent **Browse** button to access the common Open dialog that enables you to select the location of the .**dll** or .**exe** .NET assembly file that you want to import from your Jade working directory in which the Jade executable **jade.exe** is located (for example, **c:\jade\bin\SimpleDemo.dll**).

Note The .NET import process is node-centric; that is, the path to the component being imported is relative to the node. In thin client mode, the node is the application server. This is significant when the presentation client is on a machine different from the application server. In this case, it is the application server components that can be imported, not those on the presentation client.

When you click the **Browse** button, the common Open dialog is displayed when you are running in standard (fat) client mode or the browse dialog of the application server when you are running on a presentation client.

2. If you want the functionality of earlier Jade releases to apply (that is, the import process does not include referenced superclasses that are in other assemblies nor events that reference classes that are not in the assembly), uncheck the **Import Super Classes in Other Assemblies** check box.

   This check box is checked by default, in which case the import process includes superclasses of classes in the assembly when the superclasses are in other assemblies. The import process includes:

   - Classes, properties, methods, and events defined for the superclasses
   - All events that refer to those superclass classes

   For details, see "Importing .NET External Component Libraries", in the previous section.

   Note When you change the check box status and re-import the assembly, the schema is versioned if it is not already versioned and classes are moved so that they are parented according to the resulting imported hierarchy structure.

3. Click the **Next >** button.

When you click the **Next >** button, the .NET Import Wizard then enables you to change the default name assigned by Jade for the imported .NET assembly.

Specifying the Name of Your .NET Assembly

Specify the name of your .NET assembly on the second sheet of the .NET Import Wizard.

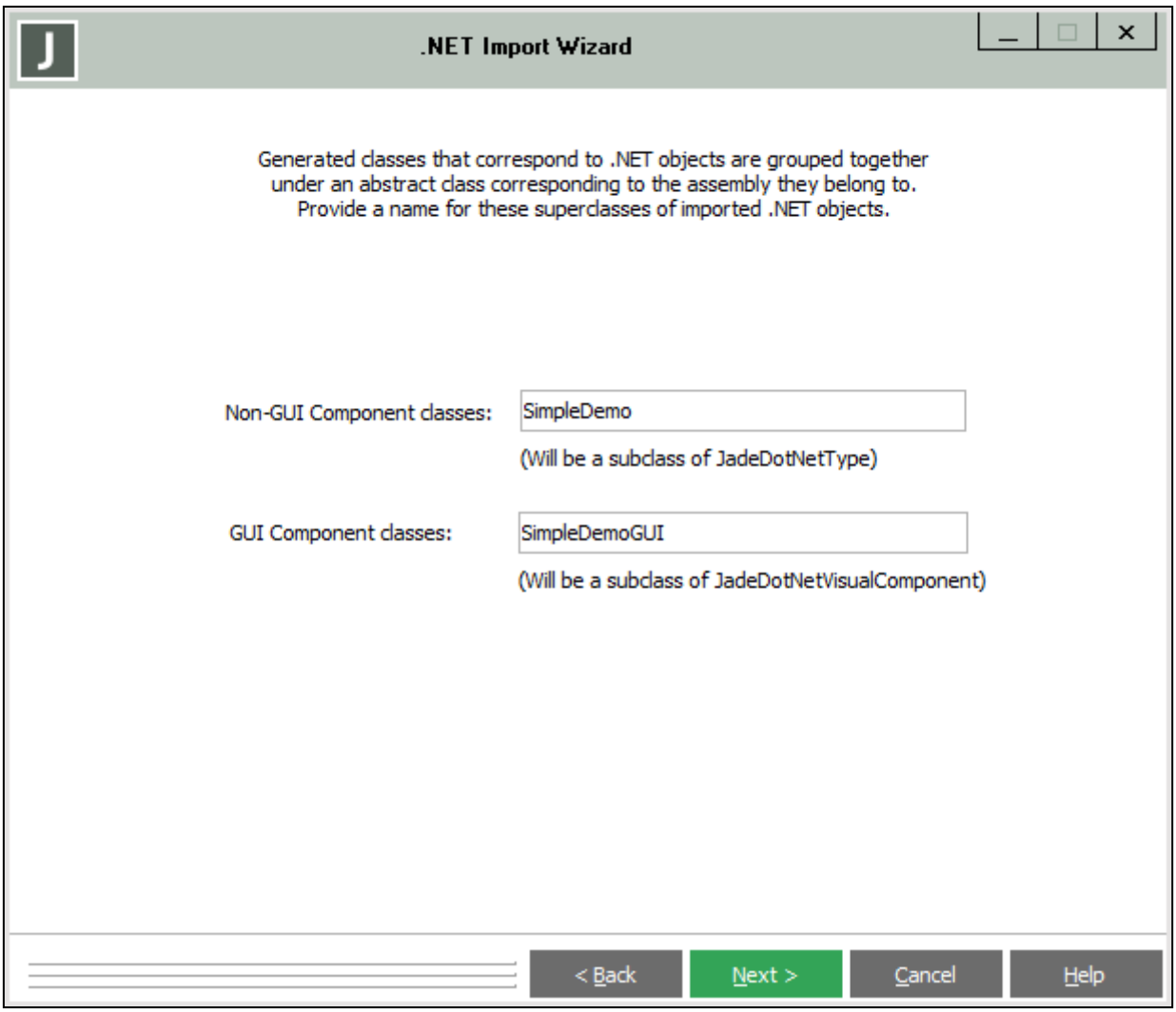
» To specify the name of the imported .NET assembly

1. In the **Import Name** text box, change the default name of the .NET assembly if you want a name other than the name selected on the first sheet of this wizard.
2. Click the **Next >** button.

When you click the **Next >** button, the .NET Import Wizard then enables you to change the default name assigned by Jade for your abstract classes.

Specifying Names for Imported .NET Objects

When you have specified the name that you require for the imported .NET assembly, the next sheet of the .NET Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify the Jade class name that you require for the .NET non-GUI and GUI abstract grouping superclass names for imported external objects.

When the .NET object is imported, Jade creates an abstract class of this name that represents the assembly.

To specify names for your .NET non-GUI and GUI objects

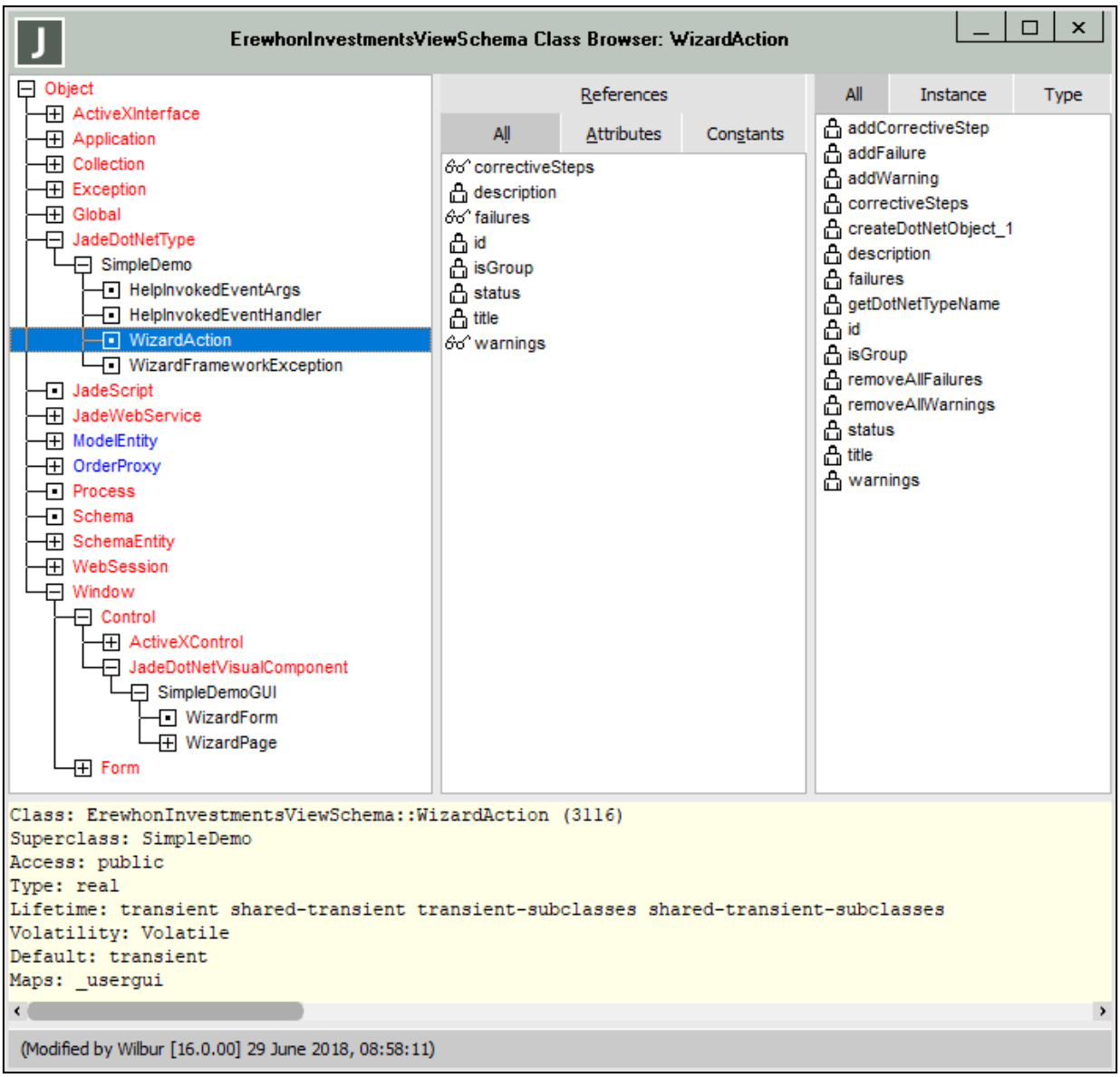
1. In the **Non-GUI Component classes** text box, specify the name of the abstract grouping subclass of the **JadeDotNetType** class under which all non-GUI classes are imported if you do not want the default name assigned by Jade. The name must be a valid name unique to the current Jade schema branch. Jade converts the first character to uppercase, if required.

2. In the **GUI Component classes** text box, specify the name of the abstract grouping subclass of the **JadeDotNetVisualComponent** class under which all GUI classes are imported if you do not want the default name assigned by Jade. The name must be a valid name unique to the current Jade schema branch. Jade converts the first character to uppercase, if required.

3. Click the **Next >** button when you have specified your .NET non-GUI and GUI abstract grouping class names. Alternatively, click the **Cancel** button (or press the Esc key) to abandon the .NET object import process.

Non-GUI components are created as subclasses of the **JadeDotNetType** class. GUI components are created as subclasses of the **JadeDotNetVisualComponent** class (a subclass of the **Control** class).

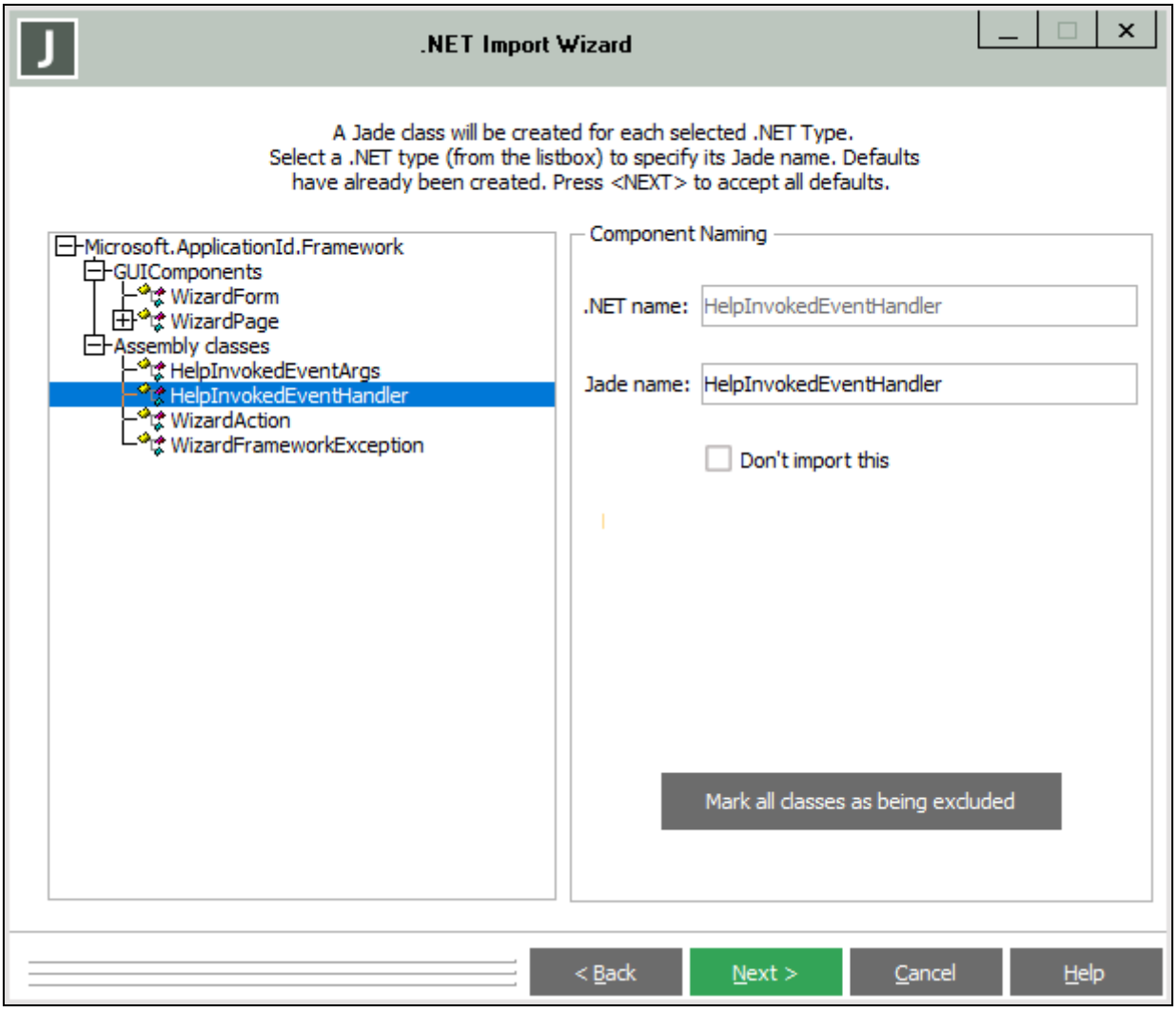
The abstract GUI and non-GUI grouping classes that are created are superclasses for all classes that are subsequently generated corresponding to objects in the imported library, as shown in the following image of the **SimpleDemo** non-GUI grouping class and the **SimpleDemoGUI** GUI grouping class.



When you click the **Next >** button, the .NET Import Wizard then enables you to change the default names assigned by Jade for each class that is created for each object in the imported library that you can access, as well as the methods and properties provided by these classes.

Naming Object Classes in the Imported .NET Library

When you have specified the grouping class names to be created for the .NET objects (types), the next sheet of the .NET Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify names for each class that is created for each object that you can access in the imported .NET assembly and to specify that a .NET component is excluded from the .NET assembly import process, if required.

When you select a type in the .NET assembly in the hierarchy list box at the left of the sheet, the .NET name for each object that you can access in the imported library is displayed in the **.NET name** text box and the default proxy name supplied by Jade for each object is displayed in the **Jade name** text box.

Only .NET classes can be mapped to Jade objects. These classes can be a reference of value type (for example, a .NET **'class'** or **'struct'**, or a C++ **'ref class'** or **'value class'**).

If an imported class does not have a definition in the assembly being imported, the class is mapped to the **JadeDotNetType** class (that is, a class is referenced but not defined in the assembly).

Jade creates a class for each object as a subclass of the grouping class that was created corresponding to the imported type library. (For details, see "Specifying a Name for Your ActiveX Type Library", earlier in this chapter.)

To define your Jade proxy class values for .NET components

1. In the **Jade name** text box, change the default Jade class name that will act as a proxy for the .NET object to the valid Jade class name of your choice if you do not want the default name assigned by Jade.

   The name must be a valid name unique to the schema branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to uppercase, if required.

   Note Help text may be displayed below the table. This help text from the .NET assembly library describes the currently selected object.

2. If you do not want the .NET object selected in the hierarchy list box at the left of the sheet imported into Jade, check the **Don't import this** check box. (By default, all public .NET objects in the assembly are imported into Jade.)

   Note You cannot import a subtype of a type that you specify is not to be imported.

3. If you want to import only a few classes in the assembly, click the **Mark all classes as being excluded** button and then individually select only those classes that you want to import, by unchecking the **Don't import this** check box.

   The button is then renamed the **Mark all classes as being imported** button. Clicking the button a second time marks all classes as being included, again.

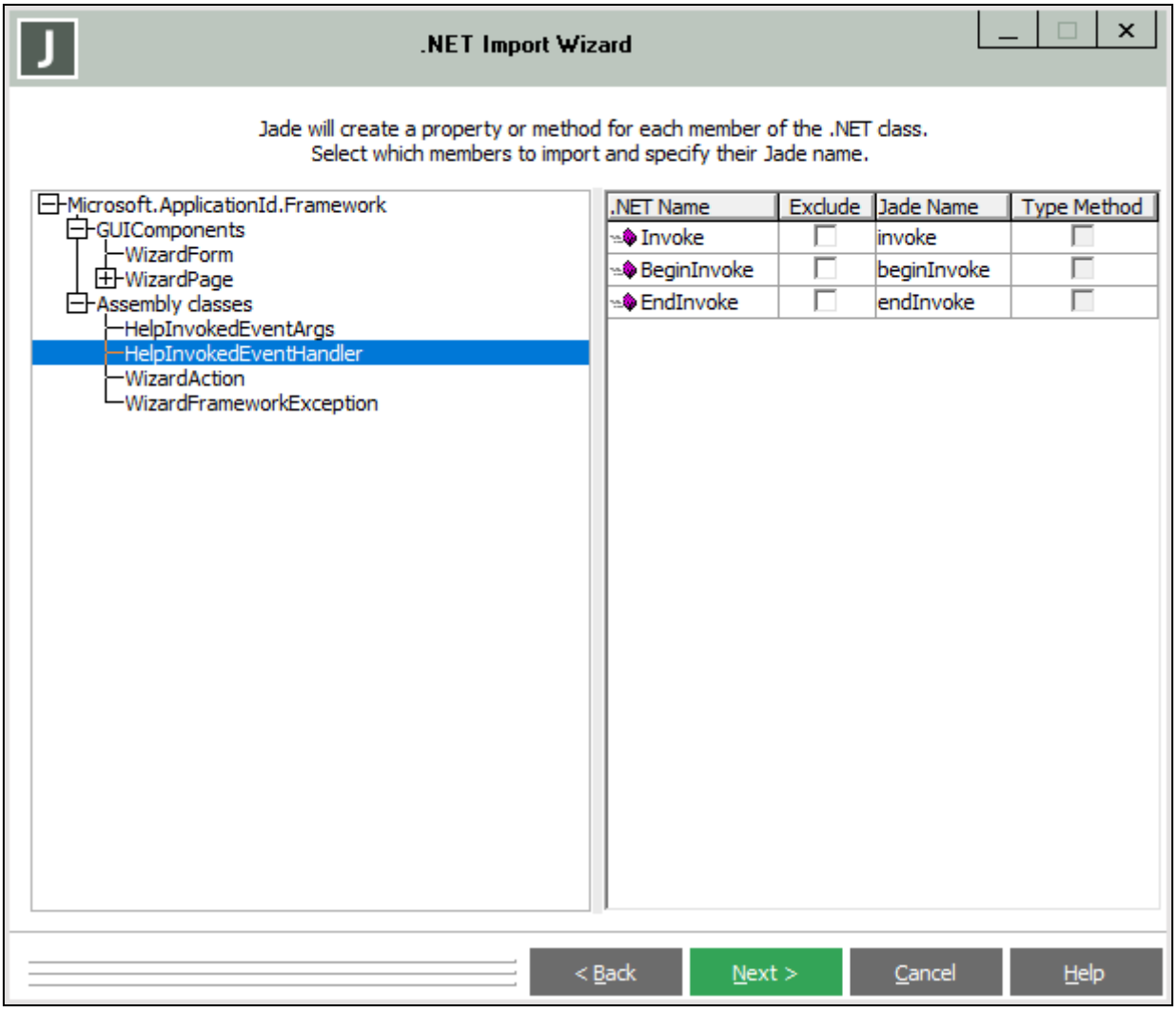
   Note As you cannot import a class unless its superclasses are also imported, after you click the button to exclude all classes, you must select each superclass of the classes to be imported and then uncheck the **Exclude** check box in the table at the right of the next sheet of the wizard. (See "Naming Members in .NET Classes", in the following section.)

4. Click the **Next >** button when you have specified your Jade proxy class names that you want to change from the default Jade values. Alternatively, click the **Cancel** button to abandon the .NET assembly import process.

When you click the **Next >** button, the wizard then enables you to change the default names assigned by Jade for each member (field, property, method, and event) that you can access in the imported .NET classes.

Naming Members in .NET Classes

When you have specified the names for each class that is created for each object that you can access in the imported .NET assembly, the next sheet of the .NET Import Wizard, shown in the following image, is then displayed.



This sheet enables you to specify Jade names for each member (fields, properties, methods, and events for the previously selected .NET objects (types) that you can access in the imported .NET assembly, and to mark each of them for exclusion from the .NET import, if required.

Although Jade supports mappings for all .NET types to Jade types, arrays of those types are not mapped to corresponding Jade array types but they are mapped to the **JadeDotNetType** class. Although an array can be returned by a method and passed into a method, the contents cannot be directly accessed via Jade.

When you select an object in the .NET assembly in the hierarchy list box at the left of the sheet, the .NET name for each field, property, method, or event that you can access in the imported library is displayed in the **.NET Name** text box and the default proxy name supplied by Jade is displayed in the **Jade Name** text box.

The icons displayed at the left of a row are listed in the following table.

Icon	Description
	.NET field, which is treated as a Jade property by the import process
	.NET property
	.NET method
	.NET event

To specify names for each of the members in a .NET class

1. If a .NET class contains types, properties, methods, or events (that is, it is displayed as a collapsed node in the class hierarchy list at the left of the sheet), expand that node to display all members for the object if you want to specify names for methods or properties that differ from the default names assigned by Jade.

   The names must be valid names unique in the class hierarchy branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to lowercase, if required.

2. If you do not want the .NET member imported into Jade, check the **Exclude** check box. (By default, all .NET members in imported classes are also imported into Jade.)

   Note You cannot import a .NET member of a .NET type (class) that you specify is not to be imported.

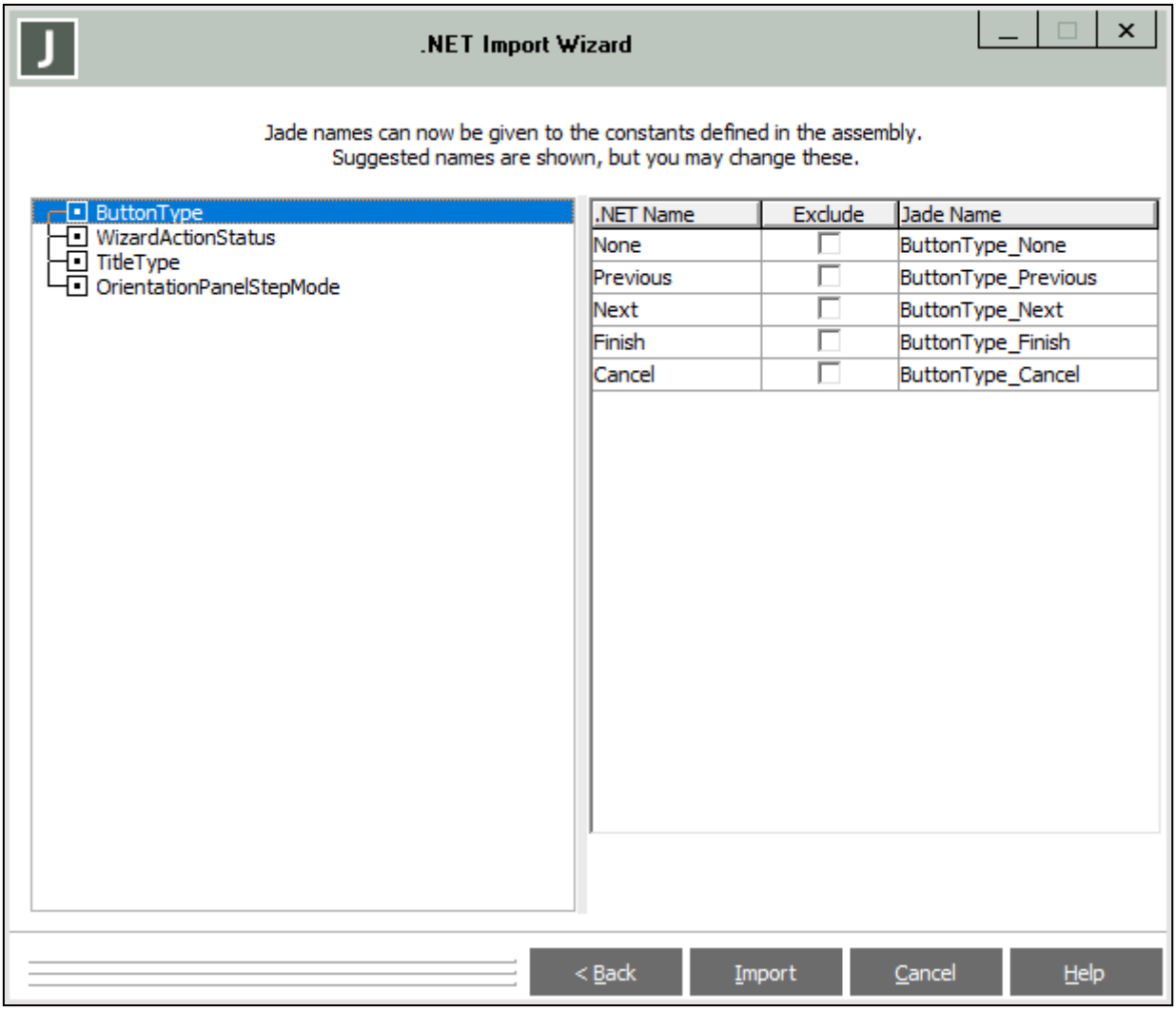
3. Click the **Next >** button when you have specified your Jade property, method, and event names and excluded any .NET members you do not want to import.

   Alternatively, click the **Cancel** button to abandon the .NET import process.

When you click the **Next >** button, the .NET Import Wizard then enables you to change the default names assigned by Jade for each constant in the imported .NET library, if required.

Naming Constants in .NET Classes

When you have specified the names to be created for the each of the .NET members that you want imported into Jade, the next sheet of the .NET Import Wizard, shown in the following image, is then displayed.



You can also import enumerations defined in the .NET assembly. The members of the enumerations that are imported have corresponding constants created on the non-GUI or GUI abstract Jade grouping class representing this .NET assembly (for details, see "Specifying Names for Imported .NET Objects", earlier in this chapter). This sheet enables you to specify names for each constant in the imported .NET library.

The .NET name for each of the constants in the library is displayed in the **.NET Name** column, and the default name supplied by Jade is displayed in the **Jade Name** column. All .NET constants are created in the **JadeDotNetType** and **JadeDotNetVisualComponent** subclasses.

» To specify names for each of the constants created for .NET object classes

1. In the **Jade Name** column, specify the name that you want created for the class constants if you want a name that differs from the default name assigned by Jade. The name must be a valid name unique to the schema branch (that is, the current schema or any of its superclasses of subclasses). Jade converts the first character to uppercase, if required.

2. If you do not want the .NET enumeration imported into Jade, check the **Exclude** check box. (By default, all .NET enumerations in imported classes are also imported into Jade.)

3. To confirm that you have finished specifying names for your .NET object classes, methods, properties, and constants, click the **Import** button. Alternatively, click the **< Back** button to redisplay the previous sheet, or the **Cancel** button to abandon the .NET object import process.

When you click the **Import** button, Jade starts the .NET library import process. Any errors (for example, naming conflicts) that are detected during the import process abort the load. A message is displayed, and the class, method, property, or constant that is in error is highlighted on the appropriate sheet of the wizard, to enable you to amend the value before restarting the import process.

When the .NET assembly is successfully imported, the .NET Import Wizard is closed, and the External Components Browser then contains your new .NET object. You can now use your .NET classes as you can any other Jade class. For details, see "Using Generated ActiveX Classes", in Chapter 4 of the *External Interface Developer's Reference*.

The property types, method parameters, and method return types depend on the original .NET type. For details, see "How Jade Imports .NET Object Definitions", in Chapter 4 of the *External Interface Developer's Reference*. For details about using non-GUI and GUI .NET components Imported into Jade, see "Using .NET Components", in Chapter 4 of the *External Interface Developer's Reference*.

Changing the Generated .NET Schema

During the import process, names for method parameters are generated without you being able to override them. For details about the default names generated by the .NET Import Wizard, see "Default Names", under ".NET Default Values and Considerations", in Chapter 4 of the *External Interface Developer's Reference*.

To change method parameter names

- Use the Methods List of the Class Browser to select the appropriate method and make the changes that you require to method parameters.

You can change the parameter name only. Do not change the parameter type.

To change class, method, or property names

- Use the Class, Methods, or Properties List of the Class Browser to change the name of the element that you want to change, if required.

You can change the element name only. Do not change the element type.

You should not add new classes to a .NET object. In addition, although it is not recommended that you remove classes from a .NET object, you *can* do so safely. For example, if you require only a subset of the features provided by a .NET library, you *could* delete the unnecessary members and objects.

Caution Although you can change the name of methods, properties, and parameters, do *not* change the types of these.

Removing a .NET Library

From the External Components Browser, the **Remove** command in the .NET submenu of the Components menu enables you to remove (delete) the currently selected .NET library from your Jade database.

Note You cannot remove a .NET library when any references exist to classes belonging to that library.

To remove a .NET library

1. In the External Components Browser, select the .NET library that you want to remove.
2. Select the **Remove** command from the .NET submenu of the Components menu. A message box is then displayed, to enable you to confirm that you want to remove the selected .NET library.
3. Click the **OK** button to confirm that you *do* want to delete the selected .NET library. (Alternatively, click the **Cancel** button to abandon the deletion.)

The External Components Browser is then updated to reflect the removal of the selected .NET library. There may be a momentary delay while this updating occurs.

When you delete a .NET library, Jade removes:

- The selected library.
- All associated classes; that is, all .NET object classes that are abstract grouping subclasses of the **JadeDotNetType** or **JadeDotNetVisualComponent** class.

Caution Although you can remove .NET object classes from the Schema Browser, it is not recommended as it results in the library remaining partially defined in Jade. If you later wanted to add the classes that you deleted, you must first completely remove the .NET library and then import that library again.

Extracting and Loading .NET Schema Definition Data

You can extract or load a .NET library as part of the schema into which it is imported or you can extract only the .NET library itself. Generated classes, methods, properties, and constants are extracted as part of a full schema extraction.

As no persistent instances of .NET objects are created, no instance data is extracted or loaded. .NET controls are treated like all other controls during the extract and load processes.

To extract a .NET library only

1. In the External Components Browser, select the .NET library that you want to extract.
2. Select the **Extract** command from the .NET submenu of the Components menu. The Extract External Component dialog is then displayed. (You can only extract the entire .NET library selected in the current schema.)
3. In the **Schema File Name** text box, specify the name and location of the .NET schema file you want to extract; for example, **s:\jade\bin\dotnet\SimpleDemo.scm**. If you do not specify a location, the file is extracted to your Jade working directory.

   If you want to extract the .NET schema to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required. The file name defaults to the default name assigned by Jade to the .NET library, with a .**scm** suffix.
4. In the **DDB File Name** text box, specify the name and location of the .NET forms file that you want to extract; for example, **s:\jade\bin\dotnet\SimpleDemo.ddx**. If you do not specify a location, the file is extracted to your Jade working directory.

   If you want to extract the .NET form and control definitions to an existing file or you are unsure of existing extract file names or location, click the **Browse** button. The common File dialog is then displayed, to enable you to select the appropriate file or location, if required.

5. If you want to specify options for your extracted .NET library (for example, to encrypt extracted source or to extract all global constants, external functions, and translatable strings rather than only those that are used), select the **Schema Options** sheet and select the extract options that you require.

6. Click the **OK** button to extract the .NET library selected in the External Components Browser. Alternatively, click the **Cancel** button to abandon your selections.

The Extract dialog is then displayed, to enable you to select the options that you require. For details about using the Extract dialog, see "Extracting Your Schema", in Chapter 10. (You can only extract the entire .NET library selected in the current schema.)

Note When you extract a .NET library, its associated classes, methods, properties, and constants are also extracted.

For details about extracting or loading the .NET library as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10.

Maintaining OpenAPI Objects

The OpenAPI Import Wizard enables you to import an OpenAPI specification into Jade, generating proxy classes based on the OpenAPI specification as subclasses of the **JadeRestProxy** class. These proxy classes have properties and methods derived from the imported specification and the methods of these classes use the **JadeRestClient**, **JadeRestRequest**, and **JadeRestResponse** classes to make the appropriate request and deserialize the result.

By default, names for the proxy classes are as close as possible to those in the OpenAPI specification. However, you can modify them during the import process; for example, to add prefixes or rename imported classes that clash with the names of user classes or classes from other imported OpenAPI specifications.

When the **OpenAPI** sheet of the External Components Browser has focus, the Components menu **OpenAPI** submenu provides the commands listed in the following table that enable you to the maintain the proxy classes generated from OpenAPI specifications.

Menu Command	For details, see …	Action
Import	Importing OpenAPI Specifications	Displays the OpenAPI Import Wizard, which enables you to import an OpenAPI specification (accessed by selecting the **OpenAPI** submenu command)
Remove	Removing an OpenAPI Specification	Removes the OpenAPI specification selected in the External Components Browser

For details about the actions that you can perform from the **OpenAPI** sheet of the External Components Browser, see the following subsections.

Applies to Version: 2020.0.01 and higher

From Jade 2020.0.02, you can create an OpenAPI specification from Jade REST APIs. For details, see "Generating OpenAPI Specifications from Jade REST APIs", in Chapter 11 of the *Developer's Reference*.

Importing OpenAPI Specifications

The selection phase of the OpenAPI Import Wizard enables you to specify appropriate Jade names for the items being imported.

When generating REST API proxy classes based on an OpenAPI specification:

- The import process generates proxy classes to represent the data model described in the OpenAPI specification as closely as possible, along with any required collection classes.
- Re-importing an OpenAPI specification deletes entities that are no longer defined in the specification.
- When an assembly is re-imported and the previous import used a different data model or if any resource operations have been removed or changed, you may need to change your existing Jade logic.

When you have imported the OpenAPI specification, Jade classes, methods, properties, and constants are generated for the items selected from the specification. For details about the abstract grouping class names, see "Abstract Grouping Classes", in Chapter 4 of the *External Interface Developer's Reference*.

The OpenAPI Import Wizard is a wizard-style dialog that consists of five steps, each represented by a sheet of the dialog. Use the **Next >** and **< Back** buttons to navigate forwards or backwards through the steps. Other than the first and last steps, each step provides default values that enable you to move to the next step.

OpenAPI objects are generated as subclasses of the **JadeRestProxy** class.

Note Some generated objects may have no properties or fewer properties than expected. This can be because the properties were not defined in the imported OpenAPI specification. If the missing properties are supplied in the response JSON text when you use your OpenAPI client, they are included in the **_additionalProperties** property of the **JadeRestDataModelProxy** class.

To import an OpenAPI specification

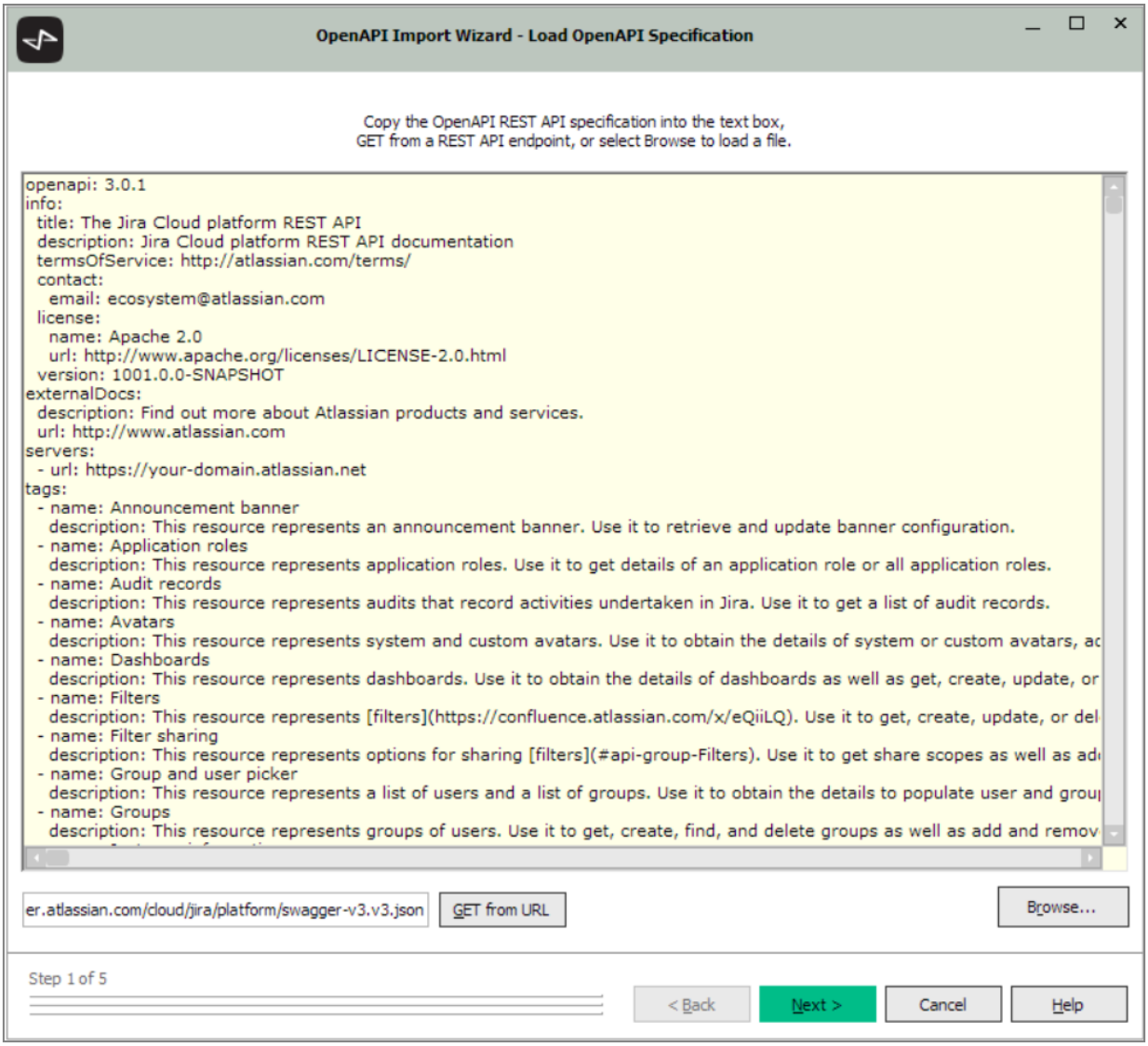
1. Select the **OpenAPI** command from the Components menu. The OpenAPI submenu is then displayed.
2. Select the **Import** command from the OpenAPI submenu of the Components menu or right-click on the **OpenAPI** sheet and then select the **Import** command from the context (popup) menu.

The OpenAPI Import Wizard, which is then displayed, systematically guides you through the import process. When all of the information has been gathered, the wizard generates all of the required classes, properties, and methods that you need to access the REST API from Jade.

Applies to Version: 2020.0.01 and higher

Selecting Your OpenAPI Specification

The first sheet of the OpenAPI Import Wizard is shown in the following image.



To select the OpenAPI specification that you want to import

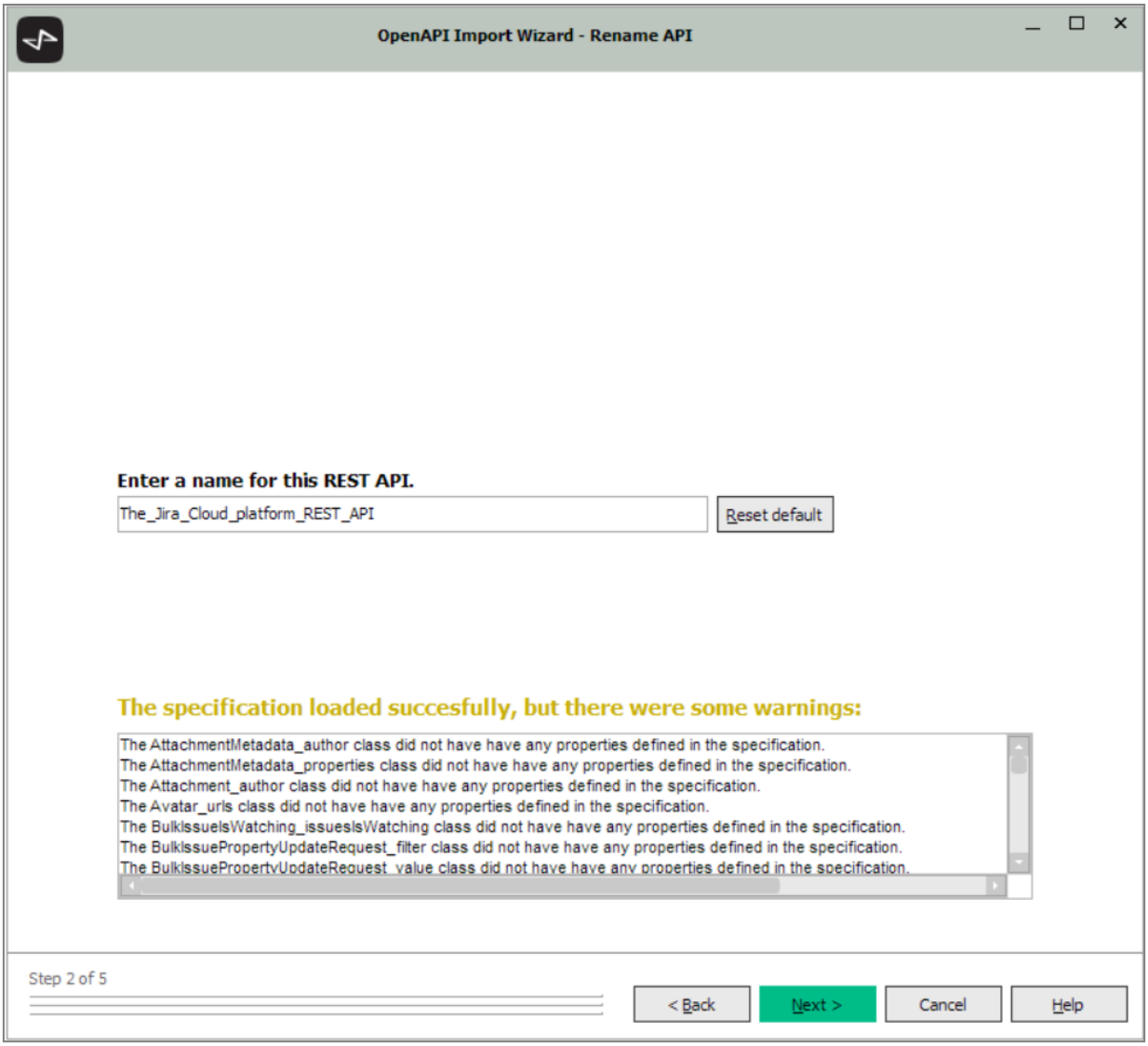
1. Perform one of the following actions.
   - Copy the OpenAPI specification into the large text box at the center of the sheet.
   - Enter the URL of the OpenAPI specification into the text box at the lower left of the sheet and then click the **GET from URL** button.
   - Click the **Browse** button to access the common Open dialog that enables you to select the location of the OpenAPI specification you want to import.
2. Click the **Next >** button.

When you click the **Next >** button, the OpenAPI Import Wizard then enables you to change the default name assigned by Jade for the imported OpenAPI specification.

Applies to Version: 2020.0.01 and higher

Specifying the Name of the REST API

You can change the name of the REST API on the second sheet of the OpenAPI Import Wizard, an example of which is shown in the following image.



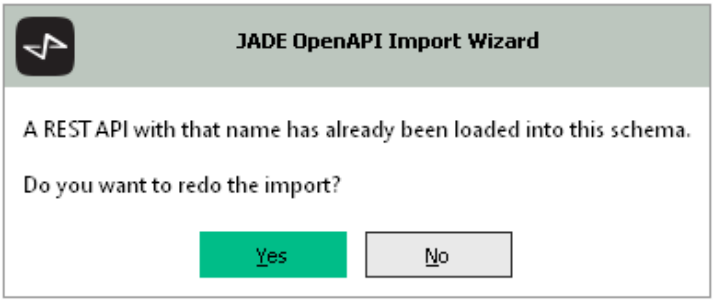
The lower area of the sheet may display warnings; for example, if the specification contains a class that does not have any properties. These do not prevent the loading of the specification but may indicate that something is wrong or missing.

To specify the name of the imported OpenAPI

1. In the text box, change the default name of the OpenAPI specification if you want a name other than the default name of the OpenAPI specification (that is, the **title** property of the **info** section).
2. Click the **Reset default** button if you want to reset the name to the name in the **title** property of the **info** section of the OpenAPI specification.
3. Click the **Next >** button.

When you click the **Next >** button, the OpenAPI Import Wizard then enables you to change the default name assigned by Jade for your proxy classes.

If a REST API with the selected name has already been loaded into your schema, the following message box is displayed.

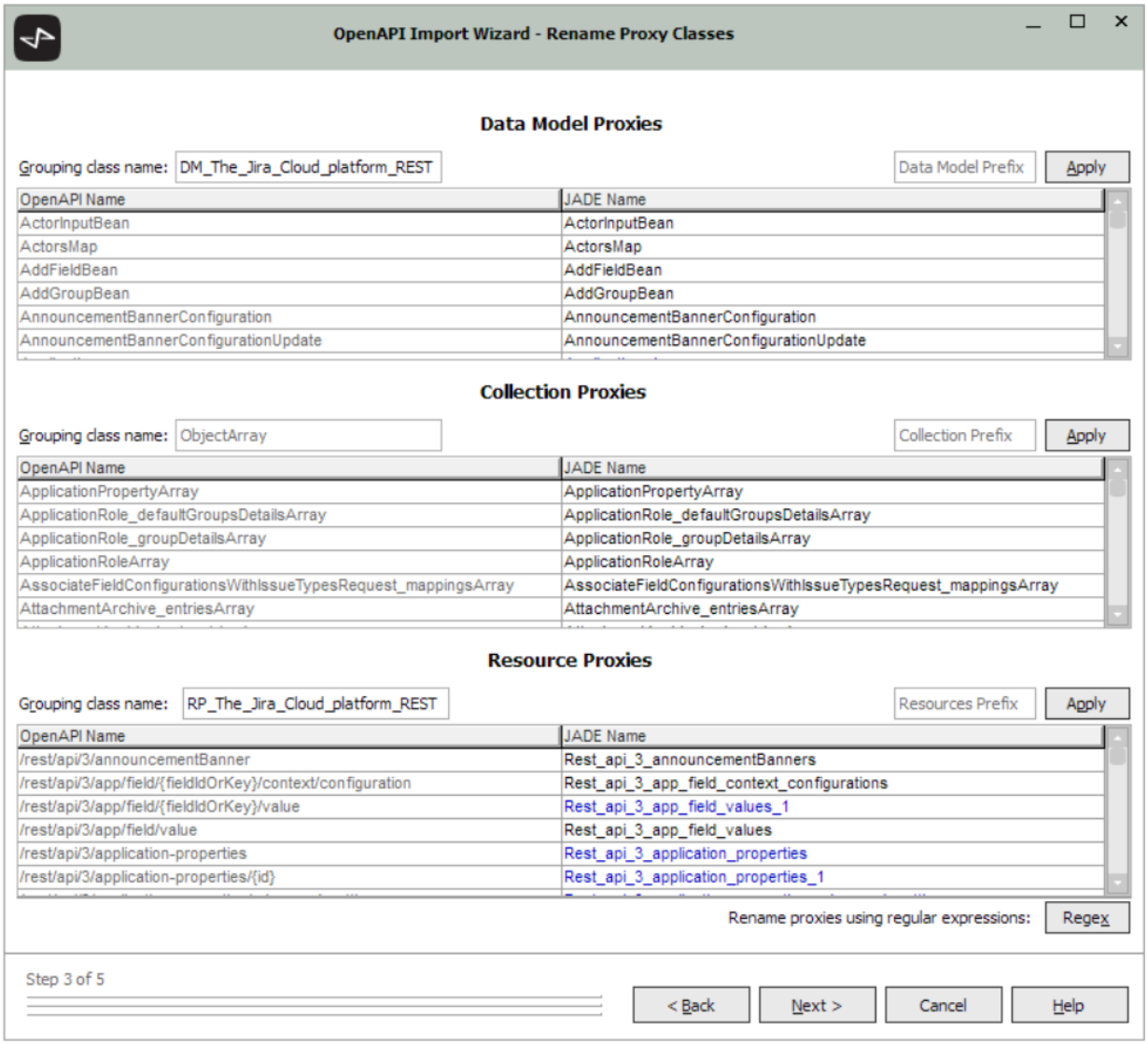


Click the **No** button if you want to go back to the first sheet of the wizard to select a different name for the REST API. Alternatively, click the **Yes** button so that the import process will overwrite the existing REST API proxy classes.

Applies to Version: 2020.0.01 and higher

Renaming Proxy Objects

Apply a prefix to or rename the default data model, collection, and resource proxy names of your REST API on the third sheet of the OpenAPI Import Wizard, shown in the following image.



To rename an imported REST proxy object or apply a name prefix

1. To add a prefix to the data model, collection, or resource proxy object names, specify your required prefix in the text box at the upper right of the **Data Model Proxies**, **Collection Proxies**, or **Resource Proxies** table and then click the **Apply** button.

2. To rename a data model, collection, or resource proxy Jade entity, click on the cell of an entity in the **Jade Name** column and then change the default name to meet your requirements.

3. To apply a regular expression to modify all names of the data model or resource proxy object names, click the **Regex** button. The OpenAPI Regex dialog is then displayed. For details, see "Applying a Regex to Data Model or Resource Proxy Names", in the following subsection.

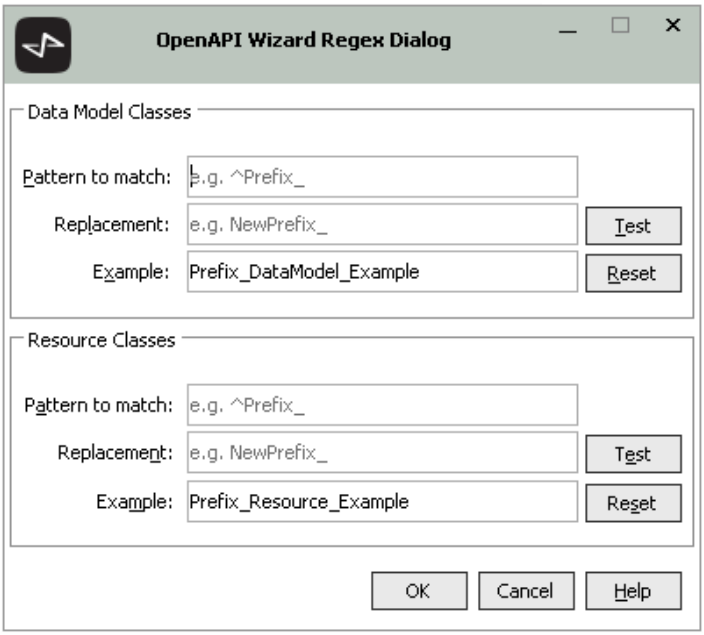
When there are no clashes (for example, the name of a class is used multiple times in the importing of the OpenAPI between data model, collection, and resource names), the **Next** button is enabled.

4. Click the **Next >** button, so that you can then rename proxy class properties and methods.

Applies to Version: 2020.0.01 and higher

Applying a Regex to Proxy Names

You can apply a regular expression (Regex) to data model or resource proxy classes using the OpenAPI Wizard Regex dialog shown in the following image.



You can apply regular expressions to the data model classes or resource classes, or to both data model and resource classes.

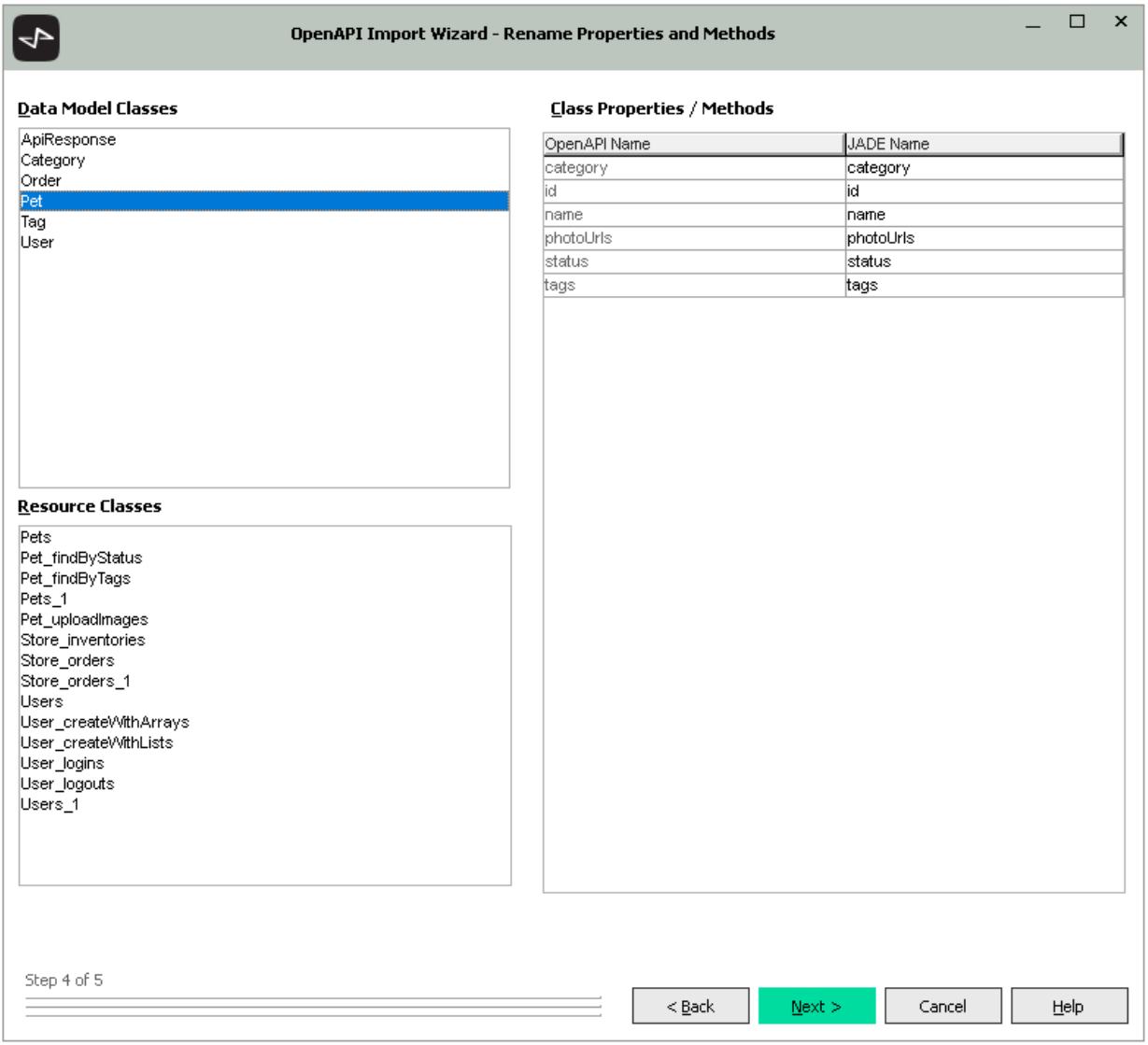
» To apply a regular expression to proxy classes in the appropriate group box

1. In the **Pattern to match** text box, specify the Regex pattern to match.
2. In the **Replacement** text box, specify the Regex replacement pattern.
3. If you want to test the effect of the Regex replacement, click the **Test** button so that the effect is applied to the **Example** text box.
4. If you want to reset the **Example** text box back to its original value, click the **Reset** button.
5. When you are satisfied with the regular expression or expressions, click the **OK** button to update the data model or resource class names, or both.

Applies to Version: 2020.0.01 and higher

Renaming Properties and Methods of Proxy Classes

Rename REST API proxy class properties and methods on the fourth sheet of the OpenAPI Import Wizard, shown in the following image.



To rename proxy class properties and methods

1. Select the data model or resource class in the **Data Model Classes** or **Resource Classes** list box.

   The properties or methods for that class are then be displayed in the **Class Properties / Methods** table on the right of the sheet.

2. Click on the cell of an entity in the **Jade Name** column and then change the default name to meet your requirements.

   When there are no clashes (for example, the name of a property or method is used multiple times in the same

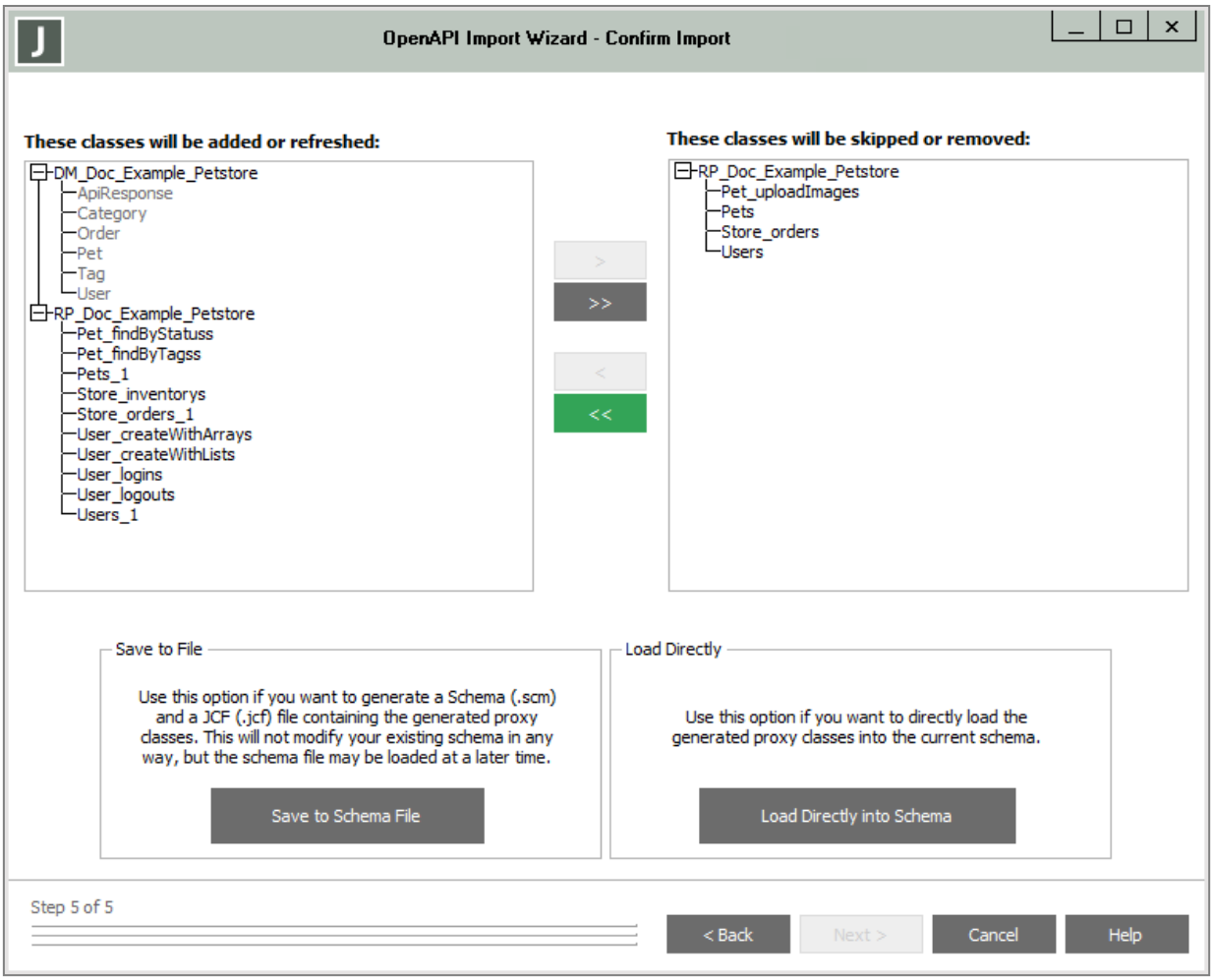
proxy class), the **Next >** button is enabled.

3. Click the **Next >** button, so that you can then confirm the importing of the proxy classes.

Applies to Version: 2020.0.01 and higher

Confirming the Importing of the OpenAPI Specification

Confirm the importing of the OpenAPI specification on the fifth sheet of the OpenAPI Import Wizard, shown in the following image.



The classes in the left-hand list box are included but those in the right-hand list box are excluded from the import process.

To exclude a resource proxy class from the import process

1. Select the resource class in the left-hand list box.
2. Click the **>** button. Alternatively, click the **>>** button to exclude *all* resource proxy classes from the import process.

Note Data model classes cannot be excluded.

To return an excluded resource proxy class to the import process

1. Select the proxy class in the right-hand list box.
2. Click the **<** button. Alternatively, click the **<<** button to return all excluded *all* resource proxy classes to the import process.

To generate a schema file to load later

1. Click the **Save to Schema File** button. The Browse for Folder dialog is then displayed.
2. Select a folder into which to generate the schema file.
3. Click the **OK** button on the Browse for Folder dialog.

A schema file of the proxy classes is then generated in the specified folder.

To load the proxy classes directly into your schema

1. Click the **Load Directly into Schema** button.
2. The schema is then versioned and proxy classes are generated into the latest version of the schema.
3. The OpenAPI Import wizard then closes automatically.

Applies to Version: 2020.0.01 and higher

Removing an OpenAPI Specification

From the External Components Browser, the **Remove** command in the OpenAPI submenu of the Components menu enables you to remove (delete) the currently selected OpenAPI specification from your Jade schema.

Note You cannot remove an OpenAPI specification when any references exist to classes belonging to that specification.

To remove an OpenAPI specification

1. In the External Components Browser, select the OpenAPI specification that you want to remove.
2. Select the **Remove** command from the OpenAPI submenu of the Components menu. A message box is then displayed, to enable you to confirm that you want to remove the selected OpenAPI definition.
3. Click the **Yes** button to confirm that you *do* want to delete the selected OpenAPI definition. (Alternatively, click the **No** button to abandon the deletion.)

The Schema Versioned message box is then displayed, stating that the schema has been versioned. When you click the **OK** button and you have reorganized the versioned schema, the External Components Browser is then updated to reflect the removal of the selected OpenAPI definition. There may be a momentary delay while this updating occurs.

When you delete an OpenAPI definition, Jade removes:

- The selected specification.
- All associated classes; that is, all **JadeRestProxy** classes other than the **JadeRestDataModelProxy** that are abstract grouping subclasses of the proxy definition.

Caution Although you can remove OpenAPI REST object classes from the Schema Browser, it is not recommended as it results in the library remaining partially defined in Jade. If you later wanted to add the classes that you deleted, you must first completely remove the OpenAPI specification and then import that specification again.

Applies to Version: 2020.0.01 and higher

Displaying All Superschema External Components in Your Current External Components Browser

You can set the level of visible schemas in your External Components Browser; that is, you can view all ActiveX control and automation libraries, .NET framework library objects, and OpenAPI specifications, respectively, imported into superschemas in the External Components Browser of your current schema.

Although a subschema inherits all ActiveX, .NET object, and OpenAPI specification classes and their methods and properties from its superschemas, these inherited classes are not displayed by default in the External Components Browser of the subschema. Instead, for the sake of clarity, only ActiveX, .NET object, >and OpenAPI specification classes local to the subschema are displayed.

» To display all superschema ActiveX control and automation libraries, .NET framework library objects, and OpenAPI specifications

1. Select the **Superschema** command from the View menu in the External Components Browser for your current schema. The View Superschemas dialog is then displayed. The immediate superschema of the current schema is displayed in the **Show classes and methods defined in:** combo box, by default.
2. Scroll down the list box of the **Show classes and methods defined in:** combo box until you locate the superschema whose ActiveX library, .NET object, or OpenAPI specification you want to display in your current External Components Browser.
3. When the appropriate superschema is selected, click the **OK** button.

After a few seconds, the External Components Browser for your current schema is populated with all of the ActiveX type libraries, .NET framework library objects, and OpenAPI specifications imported into all schemas up to the selected superschema.

ActiveX type libraries, .NET framework library objects, and OpenAPI specifications imported into superschemas are displayed in blue, or the **SuperSchema Objects** color of your choice specified in the **Window** sheet of the Preferences dialog. ActiveX type libraries, .NET framework library objects, and OpenAPI specifications imported into the current schema are displayed in black, by default.

Chapter 17 Using the C# Exposure Wizard

This chapter covers the following topics.

- Overview
- Accessing the Exposure Browser
- Adding a C# Exposure Definition
  - Setting Up Your C# Exposure Options
  - Selecting Classes for Inclusion in Your C# Exposure Definition
  - Selecting Features for Inclusion in Your C# Exposure Definition
  - Mapping C# Features
  - Saving Your C# Exposure
  - Generating the C# Classes
- Displaying a Hierarchical Exposure Browser
- Changing a C# Exposure Definition
- Removing a C# Exposure Definition
- Extracting a C# Exposure
- Finding an Existing C# Exposure
- Automating C# Exposure Generation and Extraction

Overview

A C# *exposure definition* identifies a set of Jade classes, methods, and properties that are used to generate a set of C# classes, which in turn are built into a .NET class library. (For details about using .NET to develop or maintain Jade applications from a .NET integrated development environment, see the *.NET Developer's Reference*.)

The C# framework, which encapsulates the Jade Object Manager and the core Jade programming model for Windows, enables C# and .NET developers to take advantage of Jade's server-side capabilities to:

- Use Jade to store and manipulate complex data
- Take advantage of the automatic object caching and distributed processing of Jade, by using the services of the Jade Platform from their C# code (for example, to store and retrieve Jade objects and to invoke Jade business rules)

The generated .NET class library supports the use of the .NET Language Integrated Query (LINQ) facilities to allow access to Jade objects in a natural and consistent manner for .NET developers.

As **DynaDictionary** instances cannot be included in a C# exposure, classes and properties whose type inherits from the **DynaDictionary** class are not available for selection. In addition, uses of the **ExternalArray**, **ExternalDictionary**, and **ExternalSet** subclasses of the **ExternalCollection** class are not supported by the Jade C# exposure, and are excluded from the generated C#. The C# Exposure Wizard ignores uses of these classes in the list of features that can be exposed (for example, a property reference).

Specify your C# exposure definitions within the Jade Platform development environment, by using the C# Exposure Wizard.

Note A C# exposure definition is extracted or loaded when the schema in which it is defined is extracted or loaded.

A new set of C# classes is created each time you generate the C# exposure. (For details, see "Generating the C# Classes", later in this chapter.)

The C# exposure definition is not updated when changes are made to your Jade database. (You must regenerate the C# classes or maintain them yourself, although we do not recommend the latter action.)

For details about the software required to use the C# classes, see "The .NET Class Library Framework", in Chapter 6 of the *.NET Developer's Reference*.

Accessing the Exposure Browser

» To open an Exposure Browser window for a C# exposure

1. From the Schema Browser, select the **Exposures** command from the Browse menu. An Exposure Browser window is then opened.
2. Click on the tab of the **C#** sheet.

If you have not yet defined a C# exposure definition, nothing is displayed in the **C#** sheet of this browser.

The Exposure Browser provides the Exposure menu, which enables you to maintain your C# exposure definitions.

The **C#** sheet of the Exposure Browser provides the Exposure menu, which enables you to maintain your C# exposure definitions. This menu contains the commands listed in the following table.

Command	For details, see…	Action
Add	Adding a C# Exposure Definition	Displays the C# Exposure Wizard, to enable you to add a new C# exposure definition
Browse	Displaying a Hierarchical Exposure Browser	Displays a hierarchical exposure browser that lists only the classes, properties, constants, and methods in the selected exposure
Change	Changing a C# Exposure Definition	Displays the C# Exposure Wizard, to enable you to maintain the selected C# exposure definition
Remove	Removing a C# Exposure Definition	Deletes the selected C# exposure definition
Extract	Extracting a C# Exposure	Extracts the current C# exposure to a file

Note The Exposure Browser enables you to maintain only your exposures. (Use the appropriate functions accessed from the Class Browser to maintain the Jade objects in an exposure definition.) For details about maintaining a .NET class library, see Chapter 7 of the *.NET Developer's Reference*.

Adding a C# Exposure Definition

From the **C#** sheet of the Exposure Browser, use the **Add** command from the Exposure menu to invoke the C# Exposure Wizard, which enables you to develop a C# exposure definition of classes and their entities, or class *features*, that can be exposed through a C# exposure mechanism (that is, a C# class library). The C# Exposure Wizard is then displayed.

You can select one of the following buttons at any time (as appropriate).

- **< Back**, to display the next sheet of the wizard
- **Next >**, to display the next sheet of the wizard
- **Cancel**, to exit from the wizard without saving any changes

Each schema within your Jade database can have a collection of C# exposure definitions.

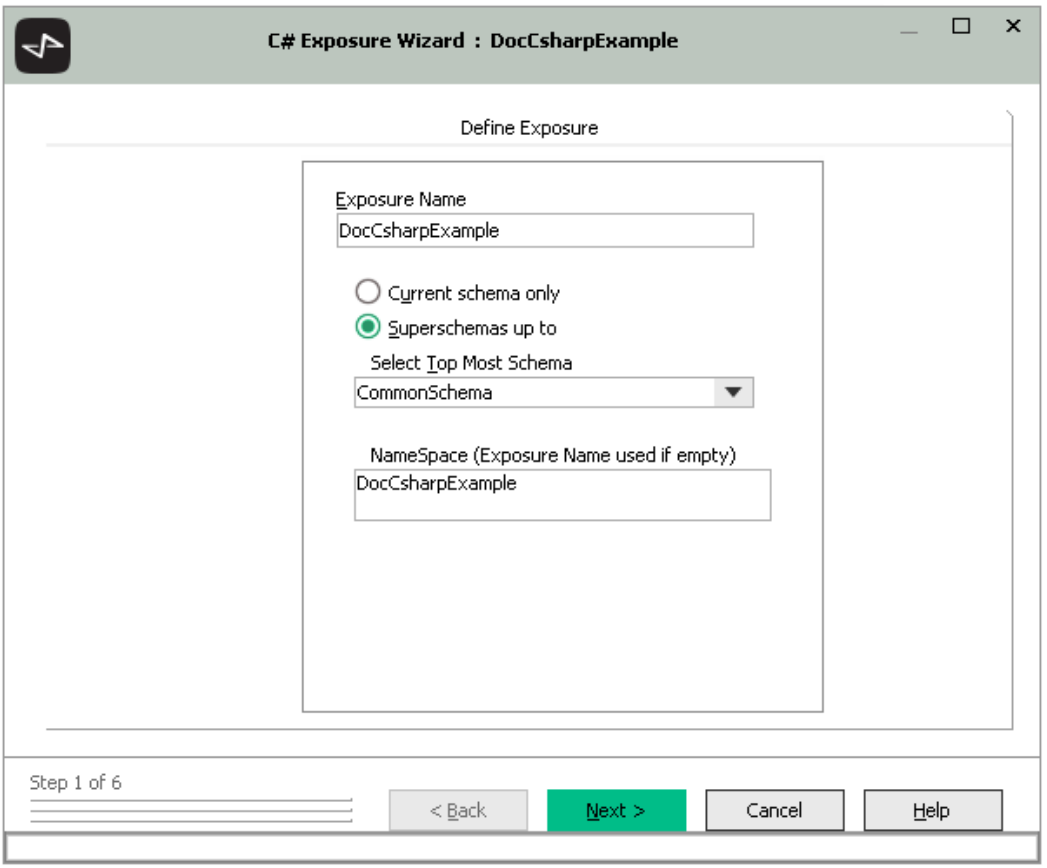
For details about the C# exposure definition steps, each contained on a separated sheet of the wizard, see the following subsections.

- Setting Up Your C# Exposure Options
- Selecting Classes for Inclusion in Your C# Exposure Definition
- Selecting Features for Inclusion in Your C# Exposure Definition
- Mapping C# Features
- Saving Your C# Exposure
- Generating the C# Classes

To add a C# exposure definition

- From the **C#** sheet of the Exposure Browser, select the **Add** command from the Exposure menu.

The **Define Exposure** sheet of the C# Exposure Wizard, shown in the following image, is then displayed.



Setting Up Your C# Exposure Options

You can define multiple C# exposures for a schema.

The **Define Exposure** sheet of the C# Exposure Wizard enables you to first define the name and source schema or superschemas of your new C# exposure.

» To specify your C# exposure options

1. In the **Exposure Name** text box, specify the name of your new C# exposure.

   The C# exposure definition name must be unique to the schema in which it is defined. The name must not already exist, can contain numeric characters, but cannot contain spaces. Jade converts the first character to uppercase, if required.

2. Select the **Superschemas Up To** option button if you do not want your C# exposure to be generated from classes in the current schema only. In the drop-down list of the **Select top most Schema** combo box, select the highest superschema in the hierarchy whose objects are also to be available for selection.

3. In the **NameSpace (Exposure Name is used if empty)** text box, specify the name of the namespace that you require for your generated C# if you want it to differ from the name of the exposure. The namespace can contain only the characters **a** through **z**, **A** through **Z**, **0** through **9**, a period character (**.**), and an underscore character (**_**).

When this text box receives focus and the text box is empty, the text is set to the name of the C# exposure and the entire text is selected. If the text box is empty, the C# exposure name is used to generate the:

- Namespace in the files
- Name of the project file
- Name of the **config** file

If this text box is not empty, the specified namespace text is used to generate these entities.

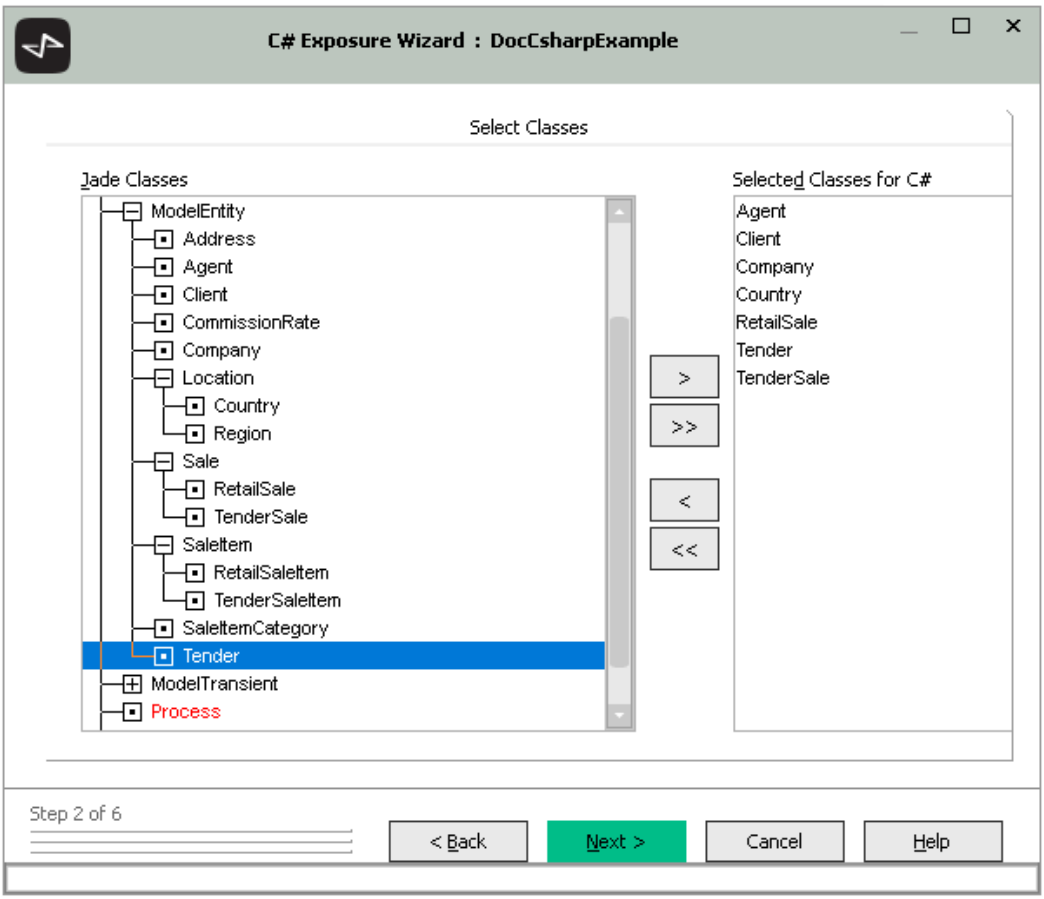
Notes The exposure name can be 100 characters in length only, and the namespace name can be any length.

As the value of the **NameSpace (exposure name is used if empty)** text box is not currently retained between initiations of the exposure wizard, you must specify it each time. (This may be implemented in a future major Jade release.)

Selecting Classes for Inclusion in Your C# Exposure Definition

The **Select Classes** sheet of the C# Exposure Wizard enables you to select the classes that are to be visible (included) in your C# exposure.

An example of the **Select Classes** sheet is shown in the following image.



If you selected a superschema of the current schema in the **Select top most Schema** combo box on the **Define Exposure** sheet, all classes available for selection in superschemas up to and including the highest-level superschema that you selected are available for inclusion in the C# exposure. By default, superschema classes are not visible and cannot be included in the C# exposure.

The classes available for selection are displayed in the **Jade Classes** list box at the left of the sheet, in their class hierarchy. If a class has subclasses, it is displayed as a collapsible and expandable node in the list box. When you perform an action on an expanded class item with subclasses, your action applies only to that class. When you perform an action on a collapsed class item with subclasses, your action applies to that class and each of its subclasses.

To preserve the class hierarchy, a class may need to be displayed in a list box of which it is not a member. When this occurs, the item is grayed (disabled from selection) to indicate to the user that it is omitted from this list box. The **Next >** button is disabled if no classes are included in the C# exposure definition.

To include a class in your C# exposure definition, perform one of the following actions

- Select the class in the **Jade Classes** list box and then click the **>** button. Alternatively, use the Shift or Ctrl convention to select multiple classes.
- Double-click a class in the **Jade Classes** list box to include it in your C# exposure.
- Right-click on a class in the **Jade Classes** list box and then select the **Add Selected Items** command from the popup menu that is then displayed.
- Select a superclass in the **Jade Classes** list box and then click the **>>** button, to include the selected class and all of its subclasses in the C# exposure.

To locate a class in the Jade Classes list box, perform one of the following actions

- Press F4.
- Right-click and then select the **Find** command from the popup menu that is displayed.

The Find Type dialog is then displayed, enabling you to specify the class you want to locate in the list box. (For details, see "Finding a Schema, Class, Interface, or Primitive Type", in Chapter 3.)

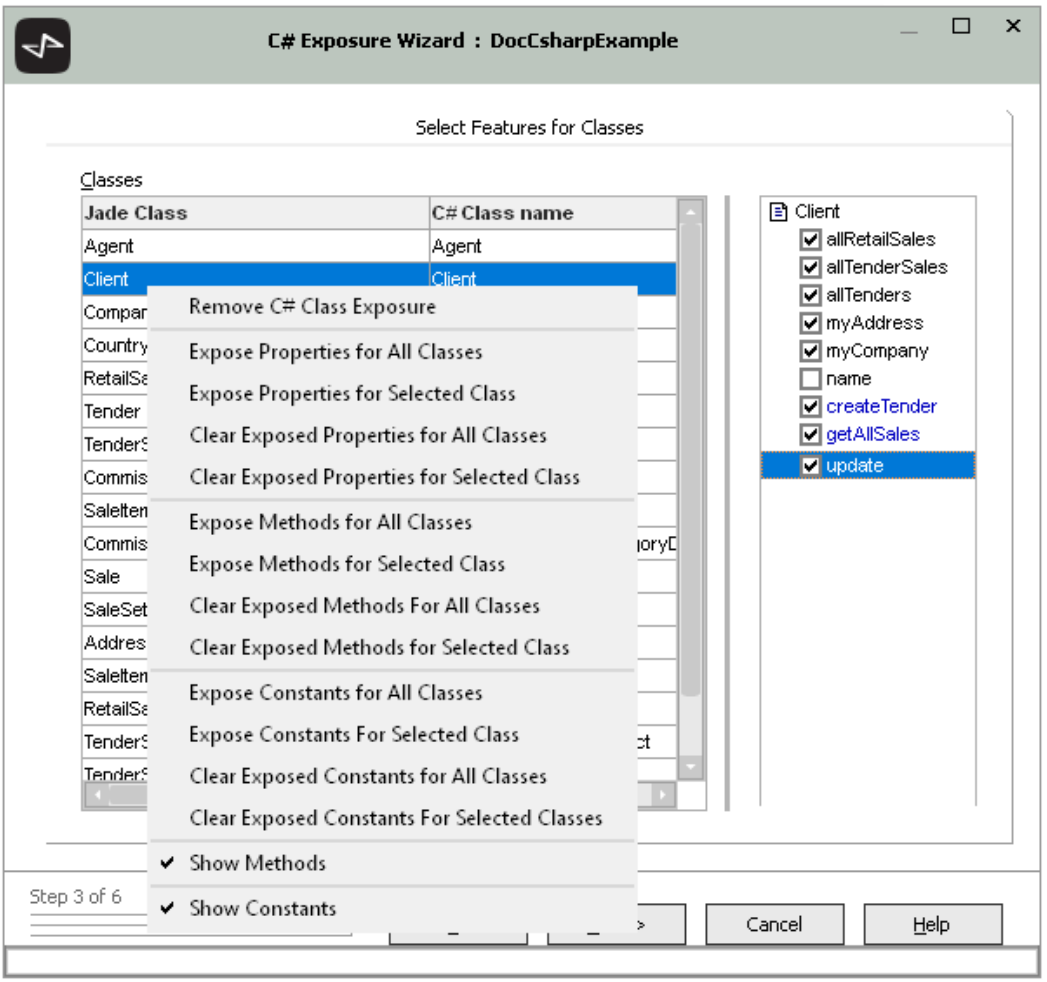
To remove a class from your C# exposure, perform one of the following actions

- Select the class in the **Selected Classes for C#** list box and then click the **<** button. Alternatively, use the Shift or Ctrl convention to select multiple classes.
- Double-click a class in the **Selected Classes for C#** list box to remove it from your C# exposure.
- Right-click on a class in the **Selected Classes for C#** list box and then select the **Remove Selected Items** command from the popup menu that is then displayed.
- Remove all classes from your C# exposure, by clicking the **<<** button.

Selecting Features for Inclusion in Your C# Exposure Definition

The **Select Features for Classes** sheet of the C# Exposure Wizard enables you to select the class features (properties and optionally methods and constants) that are included in the exposure.

An example of the **Select Features for Classes** sheet is shown in the following image.



The properties (and methods and constants, if selected) are sorted alphabetically with the properties first, followed by the methods and then the constants, which are also sorted alphabetically. Methods are displayed with a blue font and class constants with a green font.

For details about refining features displayed on this sheet of the C# Exposure Wizard, see the following subsections.

- Removing a Class from the C# Exposure
- Exposing Properties for All Classes
- Exposing Properties for a Selected Class
- Clearing Exposed Properties for All Classes
- Clearing Exposed Properties for a Selected Class
- Exposing Methods for All Classes
- Exposing Methods for a Selected Class
- Clearing Exposed Methods for All Classes

- Clearing Exposed Methods for a Selected Class
- Exposing Constants for All Classes
- Exposing Constants for a Selected Class
- Clearing Exposed Constants for All Classes
- Clearing Exposed Constants for a Selected Class
- Toggling the Display of Methods
- Toggling the Display Constants
- Displaying Tree Lines in the Features Pane

To select features for your C# exposure

1. In the **Classes** table, click on the row of the class whose features (properties and optionally methods and constants) you want to expose. The **Features** pane is then populated with all Jade properties, methods, and constants (if the display of methods and constants is selected) defined in the selected class.

   By default, all candidate properties, methods, and constants defined in the class selected in the **Classes** table are listed in the **Features** pane. If you do not want to list methods or constants, right-click on the class in the **Classes** table and then select the **Show Methods** or **Show Constants** command from the popup menu. (For details, see "Toggling the Display of Methods" or "Toggling the Display Constants", later in this chapter.)

   Tip You can display tree lines in the **Features** pane. For details, see "Displaying Tree Lines in the Features Pane", later in this chapter.

2. In the **Features** pane, check the check box at the left of each Jade property, method, or constant that you want to include in the C# exposure.

   Note When you select the exposure of a reimplemented method, the highest class in the hierarchy that declares that method is also exposed. Alternatively, you can double-click a feature in the **Features** pane, to include it in your C# exposure definition.

3. If you want a C# class to have a name that differs from the default Jade name, specify the name in the **C# Class name** column that you require for the C# class.

   The name must be a valid C# name. An error message box is displayed if you do not specify a name for the C# class or the specified value is already defined in the C# exposure.

Removing a Class from the C# Exposure

To remove a class from the C# exposure

- Right-click on the row of the **Classes** table whose class you want to remove from the C# exposure and then select the **Remove C# Class Exposure** command from the popup menu that is displayed.

The selected class is then removed from the C# exposure.

Exposing Properties for All Classes

To expose all properties in all classes included in the C# exposure

- Right-click on any row in the **Classes** table and then select the **Expose Properties for All Classes** command from the popup menu that is displayed.

Note This command does *not* automatically add methods or constants to the C# exposure, even if the **Show Methods** or **Show Constants** option is checked. (For details, see "Toggling the Display of Methods" or "Toggling the Display Constants", later in this chapter.)

All properties in the classes selected on the **Select Classes** sheet of the wizard are then exposed for inclusion in the C# exposure; that is, each property check box in the **Features** pane is checked, indicating that the properties will be generated in the C# class library.

You can tailor the property selection by unchecking the check box of any property that you want to exclude from the exposure.

Exposing Properties for a Selected Class

To expose all properties for a selected class

- Right-click on the class row in the **Classes** table and then select the **Expose Properties for Selected Class** command from the popup menu that is displayed.

Note This command does *not* automatically add methods or constants to the C# exposure, even if the **Show Methods** or **Show Constants** option is checked. (For details, see "Toggling the Display of Methods" or "Toggling the Display Constants", later in this chapter.)

All properties in that class are then exposed for inclusion in the C# exposure; that is, each property check box in the **Features** pane is checked, indicating that the properties for that class will be generated in the C# class library.

You can tailor the property selection by unchecking the check box of any property that you want to exclude from the exposure.

Clearing Exposed Properties for All Classes

To remove all exposed properties from the C# exposure

- Right-click on any row in the **Classes** table and then select the **Clear Exposed Properties for All Classes** command from the popup menu that is displayed.

All exposed properties are then removed from the C# exposure; that is, each property check box in the **Features** pane for each class is unchecked. (The classes remain included in the C# exposure.)

Clearing Exposed Properties for a Selected Class

To remove all exposed properties for a selected class from the C# exposure

- Right-click on the table row in the **Classes** table and then select the **Clear Exposed Properties for Selected Class** command from the popup menu that is displayed.

All exposed properties for that class are then removed from the C# exposure; that is, each property check box in the **Features** pane for that class is unchecked. (The class remains included in the C# exposure.)

Exposing Methods for All Classes

To expose all methods in all classes included in the C# exposure

- Right-click on any row in the **Classes** table and then select the **Expose Methods for All Classes** command from the popup menu that is displayed.

Note This command does *not* automatically add constants to the C# exposure, even if the **Show Constants** option is checked. (For details, see "Toggling the Display Constants", later in this chapter.)

All methods in the classes selected on the **Select Classes** sheet of the wizard are then exposed for inclusion in the C# exposure; that is, each method check box in the **Features** pane is checked, indicating that the methods will be generated in the C# class library. Exposed methods are displayed with a blue font.

You can tailor the method selection by unchecking the check box of any method that you want to exclude from the exposure.

Exposing Methods for a Selected Class

To expose all methods for a selected class

- Right-click on the class row in the **Classes** table and then select the **Expose Methods for Selected Class** command from the popup menu that is displayed.

Note This command does *not* automatically add constants to the C# exposure, even if the **Show Constants** option is checked. (For details, see "Toggling the Display Constants", later in this chapter.)

All methods in that class are then exposed for inclusion in the C# exposure; that is, each method check box in the **Features** pane is checked, indicating that the methods for that class will be generated in the C# class library. Exposed methods are displayed with a blue font.

You can tailor the method selection by unchecking the check box of any method that you want to exclude from the exposure.

Clearing Exposed Methods for All Classes

To remove all exposed methods from the C# exposure

- Right-click on any row in the **Classes** table and then select the **Clear Exposed Methods for All Classes** command from the popup menu that is displayed.

All exposed methods are then removed from the C# exposure; that is, each method check box in the **Features** pane for each class is unchecked. (The classes remain included in the C# exposure.)

Clearing Exposed Methods for a Selected Class

To remove all exposed methods for a selected class from the C# exposure

- Right-click on the table row in the **Classes** table and then select the **Clear Exposed Methods for Selected Class** command from the popup menu that is displayed.

All exposed methods for that class are then removed from the C# exposure; that is, each method check box in the **Features** pane for that class is unchecked. (The class remains included in the C# exposure.)

Exposing Constants for All Classes

To expose all constants in all classes included in the C# exposure

- Right-click on any row in the **Classes** table and then select the **Expose Constants for All Classes** command from the popup menu that is displayed.

Note This command does *not* automatically add methods to the C# exposure, even if the **Show Methods** option is checked. (For details, see "Toggling the Display of Methods", later in this chapter.)

All constants in the classes selected on the **Select Classes** sheet of the wizard are then exposed for inclusion in the C# exposure; that is, each constant check box in the **Features** pane is checked, indicating that the constants will be generated in the C# class library. Exposed class constants are displayed with a green font.

You can tailor the constant selection by unchecking the check box of any constant that you want to exclude from the exposure.

Exposing Constants for a Selected Class

To expose all constants for a selected class

- Right-click on the class row in the **Classes** table and then select the **Expose Constants for Selected Class** command from the popup menu that is displayed.

Note This command does *not* automatically add methods to the C# exposure, even if the **Show Methods** option is checked. (For details, see "Toggling the Display of Methods", later in this chapter.)

All constants in that class are then exposed for inclusion in the C# exposure; that is, each constant check box in the **Features** pane is checked, indicating that the constants for that class will be generated in the C# class library. Exposed class constants are displayed with a green font.

You can tailor the constant selection by unchecking the check box of any constant that you want to exclude from the exposure.

Clearing Exposed Constants for All Classes

To remove all exposed constants from the C# exposure

- Right-click on any row in the **Classes** table and then select the **Clear Exposed Constants for All Classes** command from the popup menu that is displayed.

All exposed constants are then removed from the C# exposure; that is, each constant check box in the **Features** pane for each class is unchecked. (The classes remain included in the C# exposure.)

Clearing Exposed Constants for a Selected Class

To remove all exposed constants for a selected class from the C# exposure

- Right-click on the table row in the **Classes** table and then select the **Clear Exposed Constants for Selected Class** command from the popup menu that is displayed.

All exposed constants for that class are then removed from the C# exposure; that is, each constant check box in the **Features** pane for that class is unchecked. (The class remains included in the C# exposure.)

Toggling the Display of Methods

You can include any method defined in a Jade class in a C# exposure.

To toggle the display of methods

- Right-click on any row in the **Classes** table and then select the **Show Methods** command from the popup menu that is displayed.

A check mark is displayed when methods are displayed in the **Features** pane. (When methods are first displayed, they are not selected for inclusion in the C# exposure.) All methods are then displayed in a blue font or are removed from display in the **Features** pane.

Toggling the Display of Constants

You can include any constant defined in a Jade class in a C# exposure.

To toggle the display of constants

- Right-click on any row in the **Classes** table and then select the **Show Constants** command from the popup menu that is displayed.

A check mark is displayed when constants are displayed in the **Features** pane. (When constants are first displayed, they are not selected for inclusion in the C# exposure.) All constants are then displayed in a green font or are removed from display in the **Features** pane.

Displaying Tree Lines in the Features Pane

To toggle the display of tree lines in the Features pane

- Right-click in the **Features** pane and then select the **Show Tree Lines** command from the popup menu that is displayed.

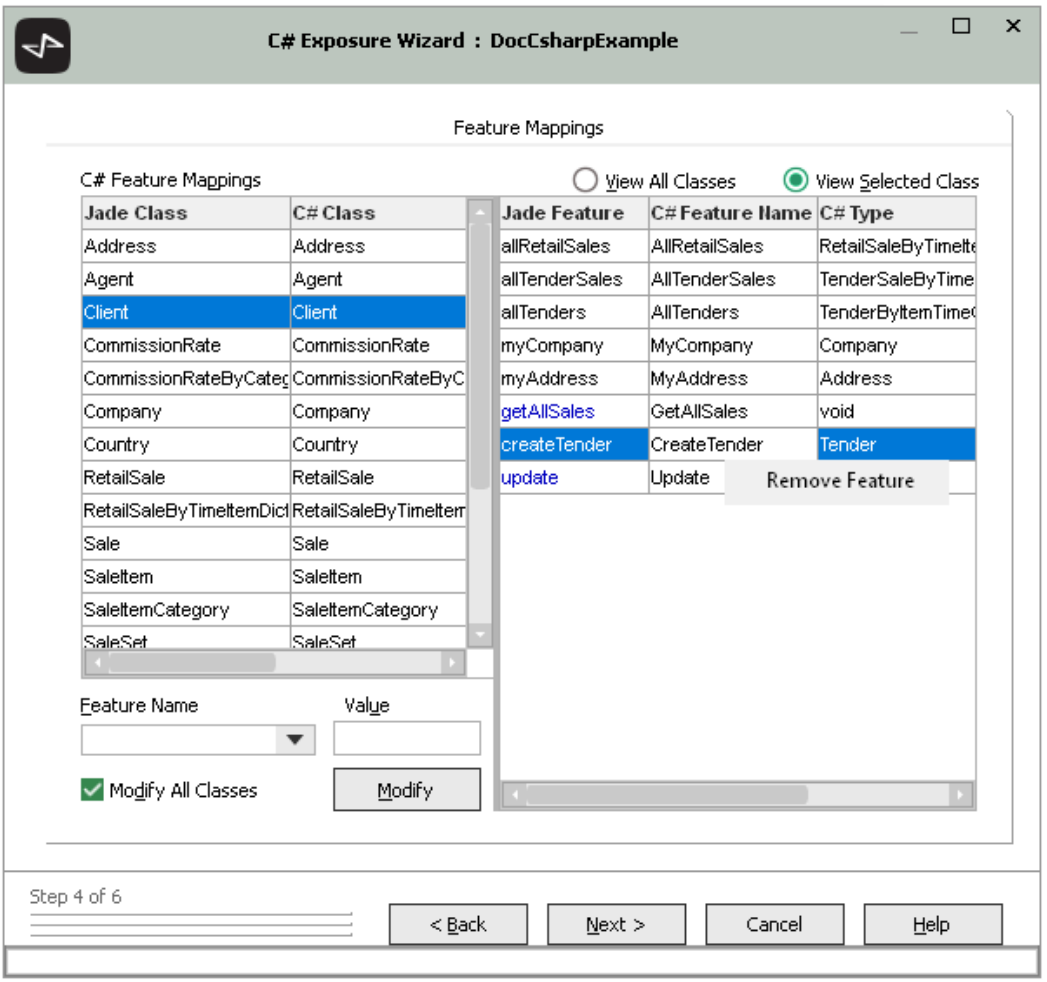
Tree lines linking hierarchy nodes are then displayed in the pane or removed from the pane. A check mark is displayed to the left of the command in the popup menu when tree lines are displayed in the **Features** pane.

Mapping C# Features

The **Feature Mappings** sheet of the C# Exposure Wizard enables you to map the features in the C# exposure. As this sheet enables you to tailor the exposed features for your selected classes, classes for which no features are exposed are not displayed.

Features in the C# exposure are grouped by class, with each alternating class in the C# exposure having an alternating background color of light yellow or white, making it easier for you to distinguish between the classes when all classes are viewed on the **Feature Mappings** sheet.

By default, features for a selected class are displayed, as shown in the following image.



If methods or constants are included in the C# exposure, these are displayed in a blue or green font, respectively. (Properties are displayed in a black font.)

To specify your feature mappings

1. If you want to change the name of a feature, click in the required cell of the **Feature Name** and then change the name to the value that you require. The name must be a valid C# name. An error message box is displayed if you do not specify a name for the C# feature or if the specified value is already defined for this class.

2. If you want to change the feature names of a class or the feature names of all classes:

   a. If you do not want the prefix or suffix to apply to all classes in the C# exposure, uncheck the **Modify All Classes** check box. (A prefix or suffix value applies to all classes by default.)

   b. Ensure that the appropriate **Add Prefix**, **Add Suffix**, **Remove Prefix**, or **Remove Suffix** value is displayed in the **Feature Name** list box.

   c. Specify the prefix or suffix value in the **Value** text box (for example, **test_**) that you want to apply to features in a specific class or to all classes in the C# exposure.

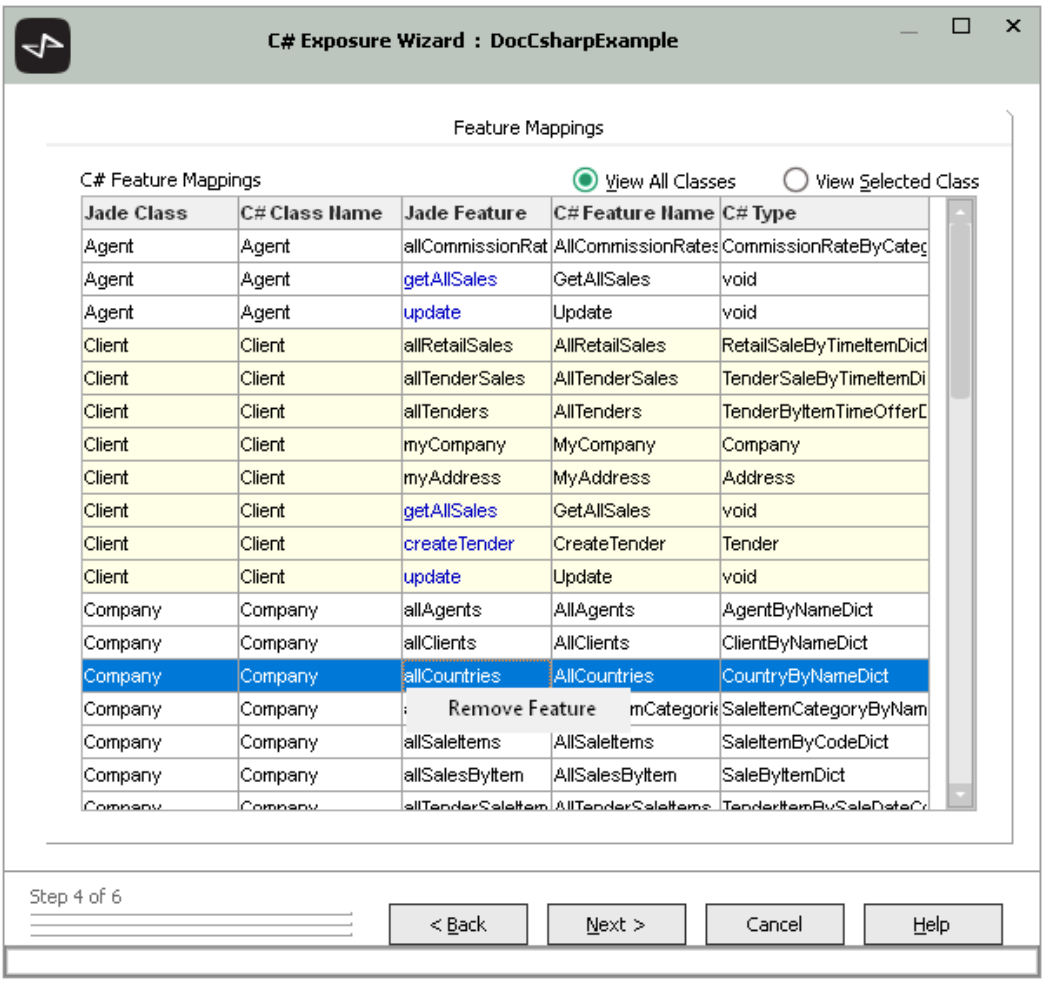
   d. Click the **Modify** button.

If the change that is being attempted would result in a duplicate feature name, a message box is displayed, informing you that the change cannot be actioned. As no update action takes place, you must change the feature name manually.

3. To remove a selected feature from the C# exposure, right-click on that row and then select the **Remove Feature** command from the popup menu that is displayed.

To display features for all classes, click the **View All Classes** option button at the upper right of the sheet, to display only the features in the class whose row is selected in the **Feature Mappings** table.

An example of the **Feature Mappings** sheet for all classes is shown the following image.



Saving Your C# Exposure

The **Save** sheet of the C# Exposure Wizard enables you to view a summary of the C# exposure definition. This sheet summarizes the entire C# exposure and enables you to save the C# exposure definition. All changes made up to this point are stored in Jade as a transient copy of the C# exposure.

» To save the C# exposure in the Jade database

- Click the **Save** button. (If you have not made any changes to an existing C# exposure, the **Next** button is displayed instead of the **Save** button.)

Alternatively, click the **Cancel** button to abandon changes made to an existing C# exposure and retain the existing definition. For a new C# exposure definition, the transient copy of the exposure is abandoned.

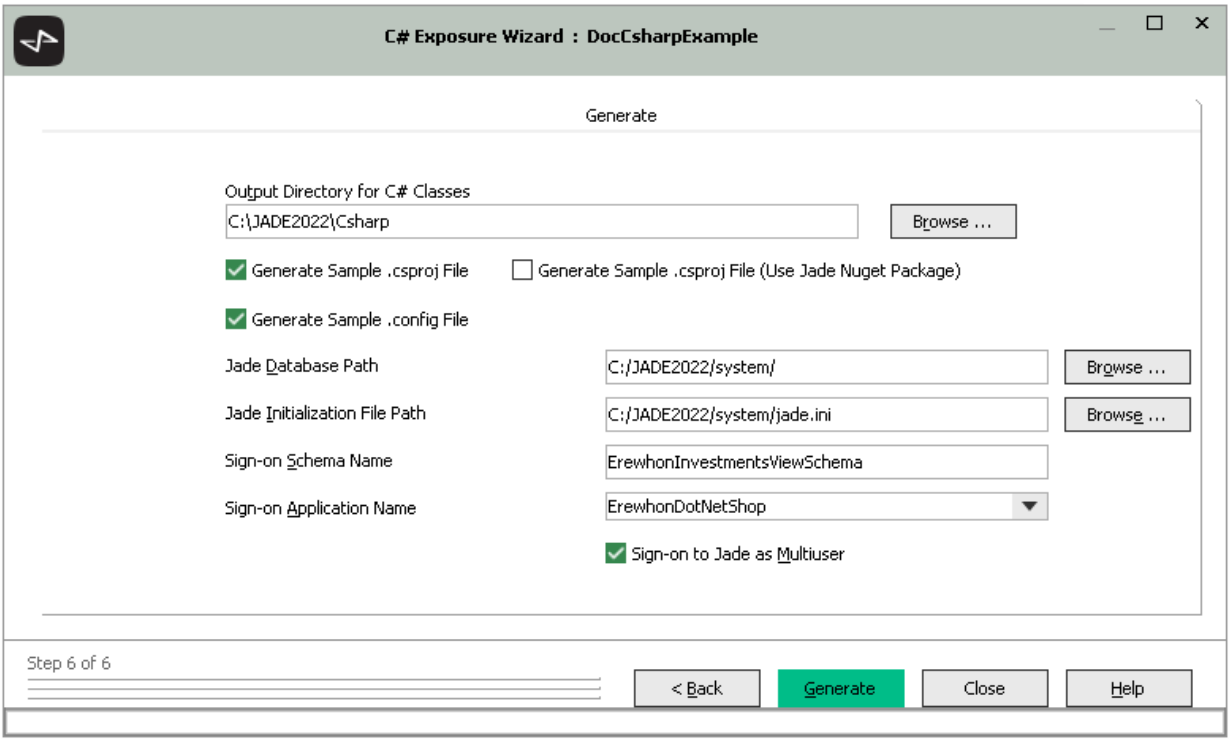
When you have saved the C# exposure, the **Generate** sheet is then displayed. For details, see the following section.

The saved C# exposure is extracted and loaded with the rest of the Jade schema.

Generating the C# Classes

The **Generate** sheet of the C# Exposure Wizard enables you to generate C# classes from the C# exposure.

An example of the **Generate** sheet is shown in the following image.



Generating the C# exposure indicates that the mapping process is complete. As this is the last step in the wizard, the **Next** button is replaced with a **Close** button.

The **Generate** sheet enables you to specify the location of generated C# files and whether to generate sample project files and sample application configuration files containing specified values.

To generate your C# class library

1. In the **Output Directory for C# Classes** text box, enter the absolute path of the directory to which the C# classes will be written. Alternatively, click the **Browse** button and then select the appropriate directory in the common Browse For Directory dialog. You must enter or select a valid output directory.

   Note In thin client mode, generated C# files are located on the presentation client.

2. Uncheck the **Generate Sample .csproj File** check box if you do not want to generate a sample **.csproj** file. You can use this project file as a starting point for creating a Microsoft Visual Studio solution.

This check box is checked by default for a new C# exposure and it is unchecked if you are modifying an existing C# exposure. When you generate a sample file for an existing C# exposure and a file with the default name already exists, the common File Save dialog is then displayed, prompting you to specify an alternative file name.

3. Uncheck the **Generate Sample .config File** check box if you do not want to generate a sample .**config** file. Configuration values are specified in the following steps of this instruction.

   This check box is checked by default for a new C# exposure and it is unchecked if you are modifying an existing C# exposure. When you generate a sample file for an existing C# exposure and a file with the default name already exists, the common File Save dialog is then displayed, prompting you to specify an alternative file name.

   When the **Generate Sample .config File** check box is checked:

   a. Specify the absolute path of the Jade database directory in the **Jade Database Path** text box or click the **Browse** button and then select the appropriate directory in the common Browse For Directory dialog.

   b. Specify the name and the absolute path of the Jade initialization file in the **Jade Initialization File Path** text box or click the **Browse** button and then select the appropriate directory and file in the common Browse For Directory dialog.

   c. Specify the value that you require for the schema to which the C# system signs on in the **Sign-on Schema Name** text box.

   d. In the **Sign-on Application Name** list box, select the non-GUI application in the schema specified in step 3.c of this instruction that you want to specify as the default application.

   Note Only applications of type **Non-GUI** are displayed in this list box.

   e. Uncheck the **Sign-on to Jade as Multiuser** check box if you want to sign on to the Jade database in single user mode only.

4. Click the **Generate** button to initiate the generating of a C# class for each exposed Jade class and optionally the .**config** and .**csproj** files.

5. When the generate process is complete (indicated at the lower-left of the wizard), click the **Close** button.

The **C#** sheet of the Exposure Browser is then displayed with your new or updated C# exposure and the C# classes generated. Any existing files with the same name are automatically overwritten. You can use the generated C# classes and the .**csproj** file with Visual Studio to create a .NET class library.

When the C# exposure is generated from Jade, the C# source includes any text defined for an exposed class, property, and method, so that it can be carried over to .NET in the exposure for inline help. The generated format is:

```
/// <summary>
/// <text-line-1>
/// <text-line-2>
/// <...>
/// </summary>
```

For details about generating C# code (get and set methods for every property in every class, but not methods and constants), see "Running a Non-GUI Client Application using jadclient", in Chapter 1 of the *Runtime Application Guide*.

Note We recommend that you do not edit the C# classes, as your changes would be lost if the class was regenerated. In addition, we recommend that each exposure is used to build its own DLL so that the .NET application can link to the generated class library.

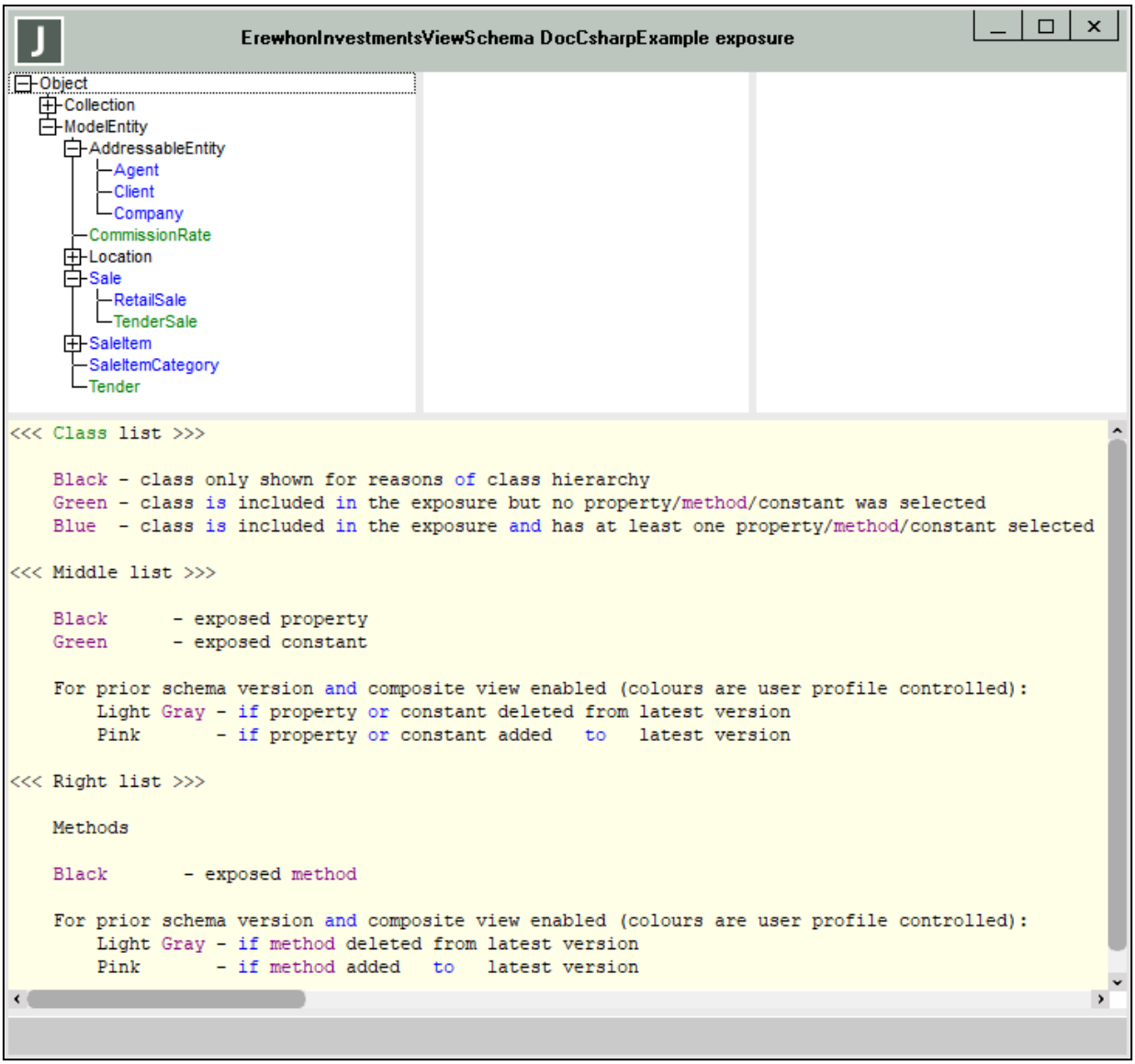
Displaying a Hierarchical Exposure Browser

You can display a hierarchical Exposure Browser that lists only the classes, properties, constants, and methods in the selected exposure.

To display a hierarchical Exposure Browser

- From the **C#** sheet of the Exposure Browser, select the **Browse** command from the Exposure menu.

The hierarchical view of your exposure is then displayed in a browser similar to the Class Browser, as shown in the following image, which is an example of the initial form of the browser.



The initial form of the hierarchy Exposure Browser displays:

- The class list in the upper left pane, containing a hierarchical list of classes in the exposure. Classes shown in:
  - Black are classes not in the exposure but are superclasses of the classes that are in the exposure
  - Green are included in the exposure but they have no property, constant, or method exposed
  - Blue are those classes in the exposure that have at least one property, constant, or method exposed

  Click on a class, to list the properties, constants, and methods exposed for that class, and to display the usual class information in the editor pane.
- The list of properties and constants in the upper middle pane that are exposed for the selected class. Exposed:
  - Properties are displayed using black text
  - Constants are displayed using green text, and they are listed after the properties names in the same list

  Click on a property or constant to display the usual browser information for the property or constant in the editor pane. For a C# exposure, the external name and type are also displayed.

  Right-click on a property to display a popup menu that contains the **References**, **Update References**, and **Read References** commands. Right-click on a constant to display a popup menu that contains the **References** command.
- The methods list in the upper right pane, containing the methods that are exposed for the selected class.

  Click on a method to display the method source in the editor pane. You cannot edit the source in this window.

  Right-click on a method to display a popup menu that contains the **References** command.
- An editor pane, containing a description of the currently selected entity. This text is read-only, and cannot be edited.

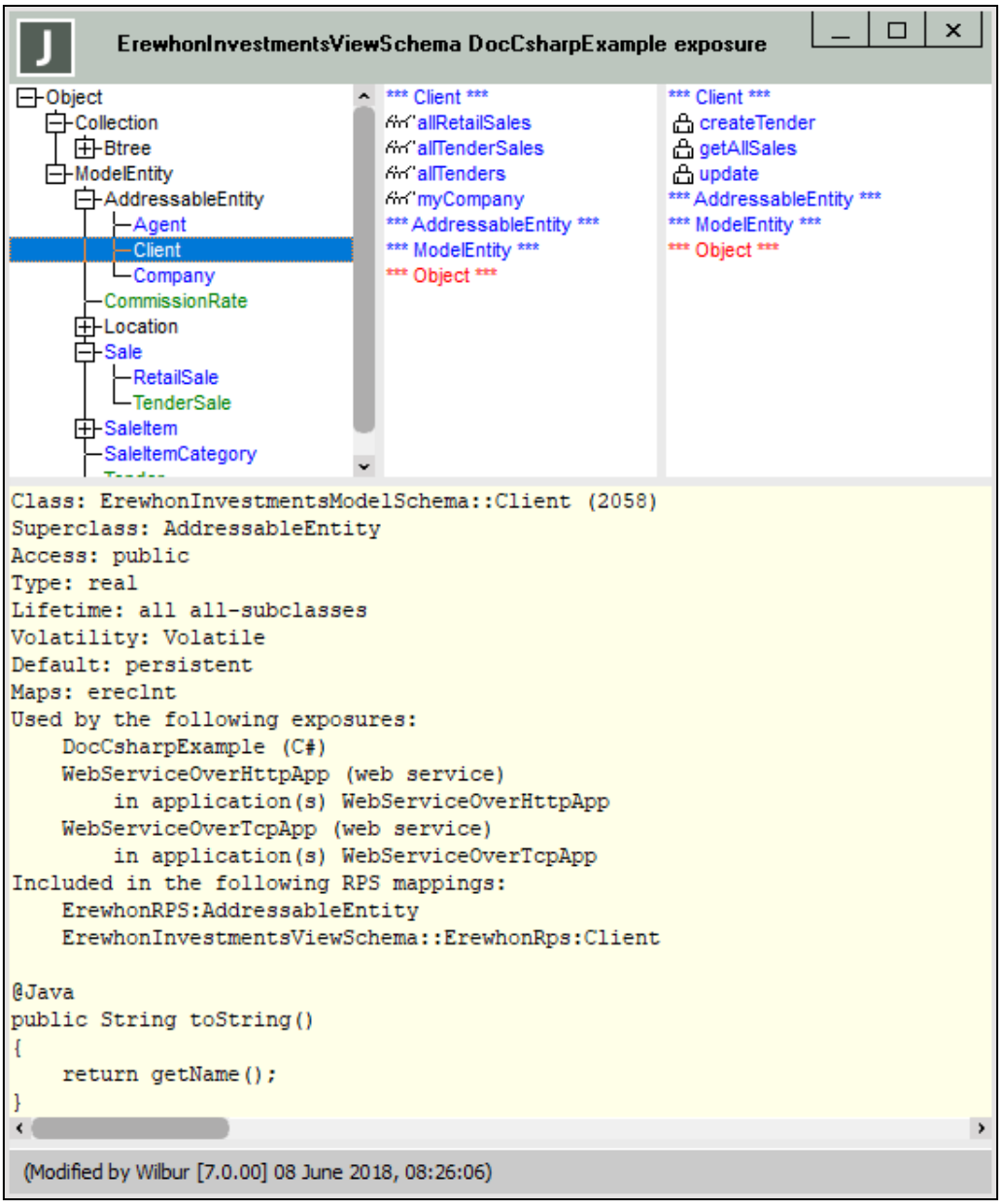
Note Other than your Jade color preferences, the form does not offer other display options supported by the hierarchy Class Browser or the options that are specified on the **Browser** sheet of the Preferences dialog.

When you press F4 to search for a class by name, the search results return only the classes in the exposure when this form has focus.

To display a composite view of entities in a versioned schema, select the **Show Composite View** command in the View menu. For details, see "Toggling the Display of a Composite View of the Hierarchy Exposure Browser".

Applies to Version: 2016.0.01 and higher

An example of entities displayed in the hierarchy Exposure Browser is shown in the following image.



Toggling the Display of a Composite View of the Hierarchy Exposure Browser

When a schema is versioned, the hierarchical Exposure Browser enables you to toggle the display of a composite view.

» To toggle the display of a composite view in a hierarchical Exposure Browser

- Select the **Show Composite View** command from the View menu. A check mark is displayed at the left of the menu command when a composite view is selected.

Note This command is enabled only when the schema is versioned.

When the **Show Composite View** command is checked, the current schema is a versioned schema, and the selected schema is the *current* schema version, the properties, constants, and methods lists display entities that:

- Are versioned, including entries for both the current version and the latest version
- Have been deleted or removed from the exposure in the latest version
- Have been added to the exposure in the latest version

When the **Show Composite View** command is checked, the current schema is a versioned schema, and the selected schema is the *latest* schema version, the properties, constants, and methods lists display:

- Entities that are versioned, including entries for both the current version and the latest version

The added and deleted items are not displayed, because it is confusing as to which version of the schema the entries were deleted from or added to.

Note The composite view uses standard color options that you control using the Preferences dialog.

Applies to Version: 2016.0.01 and higher

Changing a C# Exposure Definition

You can modify an existing C# exposure definition in the current schema.

» To change a C# exposure definition

- From the **C#** sheet of the Exposure Browser, select the **Change** command from the Exposure menu.

The C# Exposure Wizard is then displayed. For details about the C# Exposure Wizard steps, see "Adding a C# Exposure Definition", earlier in this chapter. When you are changing an existing C# exposure definition, the **Exposure name** text box in the **Define Exposure** sheet displays the name of your selected C# exposure definition.

Removing a C# Exposure Definition

From the **C#** sheet of the Exposure Browser, the **Remove** command from the Exposure menu enables you to remove (delete) the C# exposure definition that is currently selected.

» To remove a C# exposure definition

1. On the **C#** sheet of the Exposure Browser, select the C# exposure definition that you want to remove.
2. Select the **Remove** command from the Exposure menu.
3. A message box is then displayed, to enable you to confirm that you want to remove the specified C# exposure definition.
4. Click the **OK** button to confirm that the selected C# exposure definition is to be removed. Alternatively, click the **Cancel** button to abandon the deletion.

The **C#** sheet of the Exposure Browser is then updated to reflect the removal of the selected C# exposure definition. (There may be a momentary delay while this updating occurs.)

Extracting a C# Exposure

You can extract a C# exposure as part of the schema in which it is defined, or you can extract only the C# exposure itself.

To extract a C# exposure only

1. On the **C#** sheet of the Exposure Browser, select the C# exposure that you want to extract.
2. Select the **Extract** command from the Exposure menu.

   The Extract dialog is then displayed, to enable you to select the options that you require. For details about using the Extract dialog, see "Extracting Your Schema", in Chapter 10.

The file name defaults to the name of the current C# exposure, with a .**scm** suffix. The location defaults to your Jade working directory; for example:

```
s:\jade\test\bin\DemoCsharpExposure.scm
```

For details about:

- Extracting the C# exposure as part of the schema in which it is defined, see "Extracting Your Schema", in Chapter 10.
- Using the non-GUI client executable to automate the generation and extraction of a C# exposure, "Automating C# Exposure Generation and Extraction", in the following topic.

Finding an Existing C# Exposure

You can search for an existing C# exposure in the current schema by name.

To find a C# exposure

1. Select the **Find** command from the Exposure menu or right-click on the **C#** sheet of the Exposure Browser and then select the **Find** command from the popup menu.

   The Find Exposure dialog is then displayed, listing the current exposures in alphabetic order.

2. In the **Find** text box, specify text to identify the exposure you want to locate.

The list is filtered when you specify text in the **Find** text box, so that the displayed exposures are only those entries that contain the specified text. The text is case-insensitive or if the specified text is all uppercase, the entries with the same Pascal case are located; for example, **DCE** matches **DocCsharpExample**. If all text entered after the first character is uppercase, the list includes all entries in which the first character of the entity name matches the first entered character and the entity name has uppercase characters that match the order of the remaining entered uppercase text.

3. Select an entry in the **Select Required Entry** list and then click the **OK** button.

The C# exposure is then selected in the Exposure Browser.

Applies to Version: 2020.0.01 and higher

Automating C# Exposure Generation and Extraction

You can use the **jadclient** non-GUI client application to automate the generation and extraction of an existing C# exposure, passing command line arguments after the **startAppParameters** parameter to specify your generate and extract requirements. For details about:

- The **jadclient** non-GUI client application, see "Running a Non-GUI Client Application using jadclient", in Chapter 1 of the *Runtime Application Guide*.

  "Running a Non-GUI Client Application using jadclient" also covers using the **ExtractAllExposures** application from the **jadclient** command prompt to generate C# code for all classes in a schema.

- Generating an exposure using the **C# Exposure** wizard in the Jade Platform development environment, see "Generating the C# Classes", earlier in this chapter.
- Extracting a C# exposure from the Jade Platform development environment, see "Extracting a C# Exposure", earlier in this chapter.

To generate and extract an existing C# exposure as a non-GUI application, specify the following parameters in the **jadclient** program command line. The parameters following **startAppParameters** must be specified in the following order. (Parameters within brackets (**[ ]**) are optional.)

```
jadclient schema=JadeSchema
          app=CSharpGenerator
          path=database-path
          ini=Jade-initialization-file-path-and name
          startAppParameters
          schemaName=top-most-schema-in-which-classes-defined
          exposureName=existing-exposure-name
          outputFolderName=absolute-path-and-name-of-output-directory
         [shouldGenerateProjFile=true]
         [shouldGenerateAppConfigFile=true
          dataSourceConfigParam=absolute-path-of-the-database-control-file
          configFileConfigParam=name-and-path-of-Jade-initialization-file
          singleUserConfigParam=true|false
          schemaConfigParam=name-of-user-defined-schema-in-the-database
         [applicationConfigParam=an-application-defined-in-schemaName-parameter]]
```

The parameters that you can specify after the **startAppParameters** parameter are listed in the following table in the order in which they must be specified.

Parameter	Description
schemaName	Name of the highest schema in the hierarchy whose classes are to be included in the C# exposure generation
exposureName	Name your existing C# exposure
outputFolderName	Name and absolute path of the directory to which the C# classes will be written
[shouldGenerateProjFile]	**False**, if you do not want a sample .**csproj** file generated
[shouldGenerateAppConfigFile	**False**, if you do not want a sample .**config** file generated
dataSourceConfigParam	Absolute path of the folder containing the database control file (**_control.dat** path, when the **shouldGenerateAppConfigFile** parameter is specified with a value of **true**
configFileConfigParam	Name and absolute path of the Jade initialization file, when the **shouldGenerateAppConfigFile** parameter is specified with a value of **true**
singleUserConfigParam	**True** or **false** (the default value), to indicate if you want to sign on to the Jade database in single user mode or multiuser mode, respectively, when the **shouldGenerateAppConfigFile** parameter is specified with a value of **true**
schemaConfigParam	Name of the user-defined schema in the database
[applicationConfigParam]]	Optionally specifies the name of the default application in the database specified in the **schemaName** parameter (defaults to **RootSchemaApp**), when the **shouldGenerateAppConfigFile** parameter is specified with a value of **true**

The following is an example of the **jadclient** command line that generates and extracts a C# exposure.

```
jadclient schema=JadeSchema app=CSharpGenerator path=c:\jade\system
ini=c:\jade\system\jade.ini startAppParameters
schemaName=ErewhonInvestmentsModelSchema exposureName=DocCsharpExample
outputFolderName=c:\jade\csharp shouldGenerateProjFile=true
shouldGenerateAppConfigFile=true dataSourceConfigParam=c:\jade\system
configFileConfigParam=c:\jade\system\jade.ini singleUserConfigParam=false
schemaConfigParam=DocCsharpDemoSchema applicationConfigParam=CsharpConnectionApp
```

When specifying C# generate and extract parameters:

- Enclose command line arguments that contain spaces in double (**""**) or single (**'**) quotation marks.
- The **jadclient** program treats processing arguments enclosed in double (**""**) or single (**'**) quotation marks after the **startAppParameters** parameter as single-string entries in the **HugeStringArray**. The handling of strings in this **HugeStringArray** is application-specific. For example, path**= "program files"** is treated as a two-string entry and **"path= program files"** is treated as a one-string entry. How these entries are handled is determined by your application.
- The parameters are case-sensitive (that is, **multiple** is a valid argument but **Multiple** is not).

Note You cannot use the **jadclient** program to automate the generation and extraction of C# files from a deployed environment, as the internal system-only **JadeSchema** schema from which batch generate and extract actions are run is not present in deployed databases.

Applies to Version: 2018.0.01 and higher

Appendix A Glossary

[

[Relational] AttributeColumn Map

Part of a ClassTable map that defines how Jade attributes map to relational attributes (table columns).

[Relational] ClassTable Map

Defines how instances of a Jade class are mapped to rows in one or more relational tables.

[Relational] Database Map

Defines a mapping between a Jade system model and a single relational database.

A

AbsoluteQueueName

String in the generic message exchange module that fully identifies a specific queue. It includes the transport type, host address, Queue Manager name, and queue name. (Also known as FullQueueName.)

abstract class

Class that cannot be instantiated but which defines the behavior for its subclasses.

abstract method

Method in an abstract class whose implementation is deferred to a subclass.

ActiveX client

Program or piece of code that accesses the functionality and content of an ActiveX or OLE object (for example, The Jade Platform, Visual Basic, or Microsoft Word). (Also known as an Automation Controller.)

ActiveX component

Physical file that contains classes which are definitions of objects; for example, a .dll, .exe, or .ocx file. (Formerly known as OLE server.)

ActiveX control

User interface COM object that implements a number of interfaces that support its use on forms. (Formerly known as Object Linking and Embedding (OCX) external control.)

Enables existing third-party functions (for example, highly specialized controls) to be used within an application.

ActiveX interface
Set of properties that you can set or get, and a set of methods that can be called.

ActiveX object
Instance of a class that exposes properties, methods, and events to ActiveX clients. Supports the Component Object Model (See COM).

after-image
Contents of an object after a change.

Any primitive type
Can contain an object reference of any class or interface, or a value of any primitive type.

Apache
Apache HTTP Server that enables connection from a Web browser to Jade applications.

application
End-user runtime system, defining a collection of forms and other runtime information.

Application Programming Interface (API)
Enables code written in Smalltalk, Visual Basic, or C++ to coexist with the Jade language and interact with the Jade Object Manager.

application server
Executes Jade application logic in Jade thin client mode. Communicates with the Jade database on the server node and one or many presentation (or thin) clients.

archive journal set
Set of journal files that are not currently in use. Also known as offline or archived journal set.

array
Ordered collection of like objects in which the member objects are referenced by their position in the collection.

asynchronous messaging
In the generic message exchange module, occurs when the sending program proceeds with its own processing without waiting for a reply to its message.

ATCG (Automated Test Code Generator)
Records and replays GUI actions in Jade applications.

attribute
One of the characteristics (or features) of an object. Does not possess identity, rather its individuality is determined by the individual object to which it applies.

automation
COM-based technology that provides interoperability among ActiveX components. Ability of a client to drive or direct a COM object by calling methods or setting properties using one or more interfaces of that object.

B

before-image
Contents of an object before a change.

Binary primitive type
Container for arbitrary binary data.

blob
Binary large object. When the maximum length of a Binary property is not specified, the property value is separated from the parent object in the database, to simplify space allocation.

body
Part of the HTTP request that contains data, usually a serialized object.

Boolean primitive type
One of the logical truth values represented by the standard Jade language identifiers true and false.

breakpoint
Assists code analysis by interrupting its execution. The Debugger stops before executing a line of code containing a breakpoint.

browse button
Button of fixed size and color that can be clicked. Normally interpreted to indicate first, prior, next, or back.

button

Command button to begin, interrupt, or end a process. When selected, a command button appears as though it were pushed in.

C

Camel case

Phrase in which the first letter of the first word is lowercase and each word or abbreviation in the middle of the phrase starts with a capital letter with no intervening spaces or punctuation; for example, "this is an example" becomes "thisIsAnExample". (See also Pascal case.)

caret

Insertion point in a text editor, indicated by the vertical bar (|). (See also cursor.) The caret character is the wedge-shaped symbol (^) that is usually located above the 6 on the keyboard.

CDC (Change Data Capture)

The Jade Platform event stream that captures database state changes into a local data store and uses an Event Stream Producer (ESP) to serialize events into JSON format. Set of design patterns used to determine (and track) the data that has changed so that action can be taken using the changed data.

changeset

In source control, a collection of changes to files that are committed to the repository in a single operation.

Character primitive type

Defines a variable as a single ANSI or Unicode character.

characteristic

Term used in the OMG and ODMG object model for a feature (see feature).

check box

Control that gives a True/False or Yes/No option. Used in groups to display multiple choices from which the user can select one or more options.

class

Grouping of "similar" objects, including a definition of the data those objects contain (properties), and the actions they can perform (methods). Encapsulates structure and operations into a cohesive software unit, which hides the implementation details while

exposing only the interface to the class; for example, Employee would be a class in a human resources system.

class hierarchy
The "family tree" of classes, with the most general class (Object) at the root, and progressively more-specialized classes at each level of the tree.

client
The code or process that invokes an operation on an object; that is, requests a service. An object takes a client role when it uses the services of another object, either by operating upon it or by referencing its state.

clone
Local copy of a remote repository in a distributed version control system (DVCS).

clsid
Component Object Model (COM) class identifier.

coclass (component object class)
Specific implementation of a Component Object Model (COM), and corresponds to a class in The Jade Platform.

collection
Basic structure used to store multiple object references or primitive type values.

column-mapping method
Non-updating Jade method that has no parameters, returns a primitive type or a non-exclusive object reference, and is not updating.

COM (Component Object Model)
Language-independent specification for the way in which objects can interact.

combo box
Control that gives the user the choice of typing in the text box portion or selecting an item from the list portion of the control.

commit-coherent database
Database in which there are no active transactions. Backup of a secondary that when recovered, stops tracking and is in a commit-coherent (consistent) state.

component
Representation of a Jade class in an OpenAPI specification plus optional descriptions for the class and any exposed properties of that class. Typically, only classes

referenced by the REST API are exposed.

condition
Restricted form of a method that returns a Boolean result.

connection pool
Encompasses the group of client connections associated with the transport group in a Jade multiple worker TCP/IP connection environment.

consistent database
See commit-coherent database.

constraint
Used in a condition to maintain automatic inverse references when the condition is satisfied.

constructor
Method that is executed every time a new instance of a class is created.

containerization
Process of packaging an application along with its required libraries, frameworks, and configuration files together so that it can be run efficiently in various computing environments efficiently; that is, it is the encapsulation of an application and its required environment.

control
Specialized child window that is resident on top of a form or another control.

Control class
Abstract subclass of the Window class that inherits all properties and methods of that class.

CORBA (Common Object Request Broker Architecture)
Enables objects to communicate regardless of the language in which they are written or operating system on which they are running.

CUA (Common User Access)
Standards for user interfaces.

current journal set
Set of journal files that are currently in use. Also known as online or active journal set.

current version
Original version of a metaschema entity that represents the existing state of user objects. User applications and JadeScript methods use the current version of a class.

cursor
Mouse pointer. (See also caret.)

D

Data Pump application
Server application used by a Relational Population Service (RPS) node to implement incremental object replication into a target relational database.

database
Set of managed files containing both the system specification and system data.

database file compaction
Form of file reorganization in which objects are cloned but not mutated, to reclaim free space and eliminate file system fragmentation.

database role
In an SDS environment, the role assigned to the database. Valid roles are Primary Database Role and Secondary Database Role.

Date primitive type
Represents a specific day since the start of the Julian period. Defines the protocol for comparing and manipulating dates.

DCOM (Distributed Component Object Model)
Language-independent specification for the way in which objects can interact across network boundaries.

DDB
Forms definition file that contains form, control, ActiveX and .NET external component libraries, relational view, external database, skin, HTML document, and exported package definitions in .ddb format. (See also the schema file.)

DDX
Forms definition file that contains form, control, ActiveX and .NET external component libraries, relational view, external database, skin, HTML document, and exported package definitions in .ddx format. (See also the schema file.) File format in which all entities are identified by name and by their position in the XML object hierarchy.

Debugger
Enables you to observe the behavior of your code at run time by stepping through code, monitoring and changing the values of variables used in the code, or monitoring the flow of code in an application.

Decimal primitive type
Defines a variable with a specific decimal currency and number format; for example, a bank balance.

definition file
Schema file that contains form, control, ActiveX and .NET external component libraries, relational view, external database, skin, HTML document, and exported package definitions. (See also the schema file.)

delegate claim
Token validation is performed by a specified delegate method.

delta
Change control mechanism for checking out and testing changes to methods. Changes that appear successful can be checked in; less-successful changes can be discarded.

delta image
Differences between the before-images and after-images.

descriptive element
Property of the OpenAPI specification intended to provide documentation to human readers of the specification. Not typically relevant to generation of client proxy classes.

deserialize
Converts a serialized object from its standardized text representation back into an object.

design-time property
Property of a control that is displayed in the Properties dialog of the Painter. A value assigned to a design-time property can be saved, and used as the default value when a runtime instance of the control is created.

destructor
Method that is executed every time an instance of a specific class is deleted.

development environment
Environment written in the Jade language but open to other user interfaces. Contains a predefined set of classes that is the Jade framework.

DevOps
Set of practices that combines software development and IT operations.

direct web service
Web service that connects Jade systems directly over the TCP/IP communication protocol.

dispatch interface
Used to access named automation properties and methods of an object from a script.

DLL (Dynamic Link Library)
External method (or routine) that can be written in any language that can create a DLL. DLL methods can then be executed as standard Jade methods from within the Jade Platform.

docker container
Lightweight, standalone, executable package that includes everything needed to run an application; for example, code, run time, system tools, system libraries, and settings.

DOM (Document Object Model)
Interface that specifies how objects in a Web page are represented.

DTD (Document Type Definition)
States the tags and attributes used to describe content in SGML, XML, and HTML documents.

dual update mode
Stage in schema migration when updates are applied to user objects in the current .dat file and to the mutated user objects in the .reo file.

dynamic binding
See polymorphism.

E

emoji
Surrogate pair characters supported by the Unicode version of Jade.

encapsulation
Conceptual "packaging" within an object of all data and behavior relevant to that object. Helps to reduce external dependencies on the implementation of an object, thus providing security and robustness.

endpoint
In the context of REST, the first part of a URI that describes the location of the REST service.

ESG (Event Schema Generator)
Generates the Event Stream Producer (ESP).

ESP (Event Stream Producer)
Captures state changes from database journals, converts them to events, and writes to stream.

ETL (Extract, Transform, and Load)
Process that extracts data from databases, applications, and systems, transforms it to meet your requirements, and then loads it into target systems (including data warehouses, data marts, analytical applications, and so on).

event (1)
Action recognized by an object (for example, clicking the mouse or pressing a key) for which you can write code to respond. Record of a state change in the system.

event (2)
Record of a state change in the system.

event interface
Callback interface attached to an object. Used by script engines to get notifications of events thrown by objects.

event stream
Sequence of events that flow from publishers to subscribers.

exception handling
Provides a mechanism that enables you to construct Jade applications that are tolerant of exceptions that occur because of abnormal conditions for both Jade and external methods.

exclusive lock
Stops all other processes from locking an object. Jade automatically implies an exclusive lock when an object is updated. By default, objects that are being updated are locked for the duration of the transaction.

exposure definition
Collection of classes, methods, properties, and constants that are made accessible to external programs through an exposure implementation.

expression
Rule for computing a value. Consists of one or more operands combined by means of language-defined operators.

extensibility
Ability to extend a system in a safe, reliable way. Object-oriented features such as subclassing and method reimplementation are powerful aids to extensibility. Jade is extensible in that you can add new class types to the system.

external function
Method (or routine) that is not necessarily associated with any specific class.

external interface
Interface to Jade from an external source, including OCX, OLE, and ODBC.

external method
Method (or routine) for Jade classes. Can be written in any language that can create a library.

F

feature
Term for something that says something about objects or does something to objects. Features include properties, conditions, and methods (or operations). Properties include attributes and references.

file reorganization
Stage in schema migration when objects are copied to a new .reo file. The objects are mutated, if necessary, to match the latest version of the class definition.

file synchronization
Stage in schema migration when transactions that occurred during file reorganization are applied to objects in the target .reo file.

final

Class that cannot be subclassed or a method that cannot be reimplemented in a subclass.

folder

Special container control that has one or more sheets. A sheet is the same as a group box, in that it holds a series of painted controls. Only one sheet is visible at any time, but tabs for each sheet are displayed at the top of the folder. These tabs can be clicked to make a selected sheet visible.

form

Window that acts as a container for controls that display information and that permit user input. Can be standalone (the default), MDI frame containing other child forms, or MDI child form.

frame

Control that provides a way of grouping controls in a container, aligning the frame positionally or by size within its container, and for the use of three-dimensional effects Frame text appears in the area bounded by three-dimensional (3D) borders. If the borders are too large, the text area of the frame may not exist, and no text will be displayed. Can be a container for groups of controls.

full replica mode

The mode of operation whereby an RPS node retains a full copy of the primary database, regardless of the mapped extent. In this mode, tables in the relational database can be repopulated without impacting the primary database. An RPS node running in full replica mode can also be used for disaster recovery, as it can acquire the primary database role in a takeover situation.

FullQueueName

See AbsoluteQueueName.

G

get

See receive.

Git

Web-based distributed version control system to track changes in source files and coordinate work on those files among a team of people.

group box
Control that provides a way of grouping controls in a container. Can be a container for groups of controls.

GUI (Graphical User Interface)
Windows, menus, dialogs, and controls through which the user interacts with the system.

GUID (Globally Unique Identifier)
128-bit integer number used to identify resources.

H

header
Part of the HTTP request that contains metadata such as the content type of the data and any authorization information.

HMAC (Hash-based Message Authentication Code encryption algorithm)
Validates the signature of a JSON Web Token.

HTML (HyperText Markup Language)
Set of codes used to create documents for the Internet.

HTTP (HyperText Transfer Protocol)
Underlying protocol used by the Internet to define how messages are formatted and transmitted and the actions that Web servers and browsers should take in response to commands. REST requests are all HTTP requests, so the terms are often used interchangeably.

I

identity
"Property" of an object that distinguishes it from all other objects. In Jade, objects possess a unique object identifier (OID).

IIS (Internet Information Server)
Microsoft Web server.

inheritance
Mechanism whereby a subclass exhibits, by default, all the characteristics of its superclass. Those characteristics may then be extended or specialized in whatever way is appropriate to the subclass.

instance
Specific member of the group of objects that a class represents. For example, an instance of class Employee might be a specific employee named John Jones.

instance method
Method whose receiver at run time is an instance of the class, primitive, or interface type.

instance variable
Variable found in all instances of a class; components of an object. The term, which derives from Smalltalk, generally implies an actual field in the data structure representing an object, and is therefore one of the possible implementations of an object's attributes.

instantiation
Creation of an instance of a class by establishing its initial state and assigning a new object identifier.

instruction
Forms the body of your method, and defines the actions to be carried out as a sequence of actions. (Can also be referred to as keyword.) Specifies one corresponding action. Executed sequentially, one after the other, and not simultaneously.

Integer primitive type
Represents the set of positive and negative whole numbers. Any value of the Integer type is therefore a whole number.

Inter-Node Communications (INC)
Module providing peer-to-peer communications between participating nodes in a Synchronized Database Environment (SDE).

inter-object conversion
Stage in schema migration when relationships between objects in the .reo file are established, including population or repopulation of dictionaries.

interface
See Jade interface or external interface.

inverse reference
End point of a relationship.

J

Jade client
Jade Object Manager application with its own API that enables other languages to use the Jade object manager database. Maintains a cache of local objects.

Jade database
Maintains the physical database files, containing control information used by the database, schema definition, and user data.

Jade interface
Collection of method declarations that define common behavior among unrelated classes without forcing them to be coupled together by class inheritance. Set of methods that are guaranteed to be available on any implementing class.

Jade language
Purpose-built Jade development language. An interpretive object-oriented language.

Jade object engine
Contains an object-oriented database, providing storage, cache management, and dynamic binding. Provides object model functionality, high performance, a seamless interface to the object model, and extensibility of schema and storage media.

Jade Platform
Completely integrated environment for the development of object-oriented distributed processing systems.

Jade runtime environment
Uses the Jade language interpreter to handle execution of methods at run time. (See also Runtime-only environment.)

Jade server
Arbitrates between multiple client processes making requests and handles locks and notifications.

Jade specification
Set of class definitions stored in a specific Jade Platform development environment.

Jade system model
Single Jade specification or object model that potentially spans multiple Jade schemas.

Jade thin client
Provides a smaller operational "footprint" on the user workstation, by moving most of the processing requirements from the presentation client to the application server.

JadeJWTModel class
Abstract superclass of JSON Web Token classes that enable you to create, parse, and validate JSON Web Tokens.

JadeScript
System class that enables you to execute and debug methods independently of your Jade application.

JOM (Jade Object Manager)
Object engine that lies at the heart of the Jade Platform.

journal (1)
Complete set of audit records that capture details of changes to the database or to the database state. Also known as audit trail, audit log, recovery log, transaction journal, and transaction log.

journal (2)
File comprising part of the journal.

journal file number
Number, sequentially incrementing by 1 for each journal file.

journal serial number
Number, sequentially incrementing by 1 for each audit record within the entire journal. Also known as audit serial number (ASN).

journal synchronization mode
Write synchronization mode of journals from an SDS database server node to secondary databases.

JSON (JavaScript Object Notation)
Standard data-interchange format used for serializing objects before transmission over the web.

JWT (JSON Web Token)
Open standard tracked by RFC 7519, to define a compact and self-contained way to securely transmit information between parties represented as a JSON object.

K

keyword
Another term for instruction. (See instruction.)

L

label
Control that displays text that cannot be directly changed by the user. You can write code that changes a label in response to events at run time.

LAN (Local Area Network)
Group of networked workstations within a relatively small geographical area.

latest version
New version of a meta schema entity created as a result of a development change that has been made or loaded but has yet to be brought to life.

list box
Control that displays a list of items from which the user can select one or more items.

load style
Parameter to a schema load determining into which schema context changes are loaded and whether non-structural versioning is allowed.

local repository
Also known as the master source control system repository.

LSN (Log Sequence Number)
Actual location of a journal record, comprising the file number and the byte offset within the file.

M

mainframe hosts
Connection can be made to a mainframe host using a TCP/IP or named pipe transport provider, from the supplied Connection class. Jade can interface to existing mainframe applications on IBM, UNIX, or Unisys hosts.

mapped extent

Set of objects in the production database included by all class mappings in a database map that are not excluded by filter conditions.

mapped extent mode

Mode of operation whereby an RPS node retains a copy of the mapped extent, even after user data objects have been replicated to the relational database. In this mode, tables in the relational database can be repopulated without impacting the primary database.

mapping method

Has the same name as a property, and is invoked automatically each time the value of the property is accessed or updated in a method.

MDI (Multiple Document Interface)

Enables the display of multiple documents at the same time, with each document displayed in its own window. MDI applications often have a Window menu item with submenus for switching between windows or documents.

messages

Means by which objects communicate with each other, requesting services and initiating actions (methods); for example, obtaining a customer balance. In the generic message exchange module, a data packet that includes a set of properties (the header) and a payload.

messaging

Program-to-program communication in which data is sent in messages rather than by calling each other directly.

metaclass

Class of a class. In Jade, classes are objects, but as every object is an instance of some class, class objects are instances of a special type of class called a metaclass. A class whose instances are themselves classes.

metaschema

Objects describing system entities such as schemas, classes, properties, and methods.

method

Implementation of an operation for a specific class and signature. Represents the behavior of an object and constitutes its procedural interface.

method view
Grouping of methods from any class in any schema into a named workspace that bookmarks a workflow.

multi-value claim
Any one of the specified values must be present in the JSON Web Token.

N

node
Installation of Jade that dynamically supports the client role and the server role on the same workstation. A client node initiates a request and a server node processes the request. Each node can take on client or server roles.

nodes and processes
In a Relational Population Service (RPS), a primary system is comprised of primary nodes. A secondary system is comprised of secondary nodes. Secondary processes execute in a secondary system. In addition, special processes called replay processes are created in secondary servers to reflect and represent the original updating processes on the primary database. These replay processes are associated with a single secondary node called a replay node.

non-structural versioning
Versioning to ensure that applications can run unchanged; for example, a method change loaded into the latest schema context leaves the current version unchanged.

notification
Facility to provide concurrent access to data and synchronization of objects in response to changes in the state of the system.

O

object
Real or abstract business entity that exhibits well-defined behavior (for example, a specific customer).

object-orientation
Ability to model and construct information systems in terms of self-contained objects.

object mutation
Schema migration operation where an object's structure is converted into a form compatible with the class definition in the latest schema version.

ODBC (Open Database Connectivity)
Enables SQL statements to access a relational view of Jade user data or a relational database to be transformed to an external database schema.

OID - Blob/Slob
Unique identifier of Blob or Slob, comprising:
0.parent-instance-number.parent-class-number.
subclass-level.property-ordinal-number:edition; for example, 0.6985.2557.2.1:3.

OID - Object
Unique object identifier, comprising: class-number.instance-number:edition; for example, 2557.6985:3.

OLE control (Object Linking and Embedding control)
Allows attachment of objects that are edited or controlled by other applications; for example, a Word document or video clip can be stored in Jade but editing or playing the object invokes the controlling application. Standard Windows mechanism for transferring and sharing information between applications.

OLE server
Supports objects that can be embedded in a document. (See ActiveX component.)

operation
(1) Represents the behavior of objects. Service (provided by an object) that can be requested. The set of operations defined for a class specifies the interface to the objects that are instances of the specified class. (2) Representation of a REST method within an OpenAPI specification. It is contained within a path and keyed by an HTTP verb.

option button
Control that displays an option that can be turned on or off. Only one of a group of option buttons can be turned on. Usually used as part of a group to display multiple options from which the user can select only one.

OTLS (Opportunistic Transport Layer Security)
Allows encrypted connections with Microsoft Exchange servers from a TcpIpConnection class.

P

package
Set of classes, properties, methods, and constants built in an exporting schema and exported for use in importing schemas.

Pascal case
Phrase in which the first letter of every word in a compound word is uppercase (for example, "longfunctionname" becomes "LongFunctionName"). See also Camel case.

patch control
Assigning a numeric patch version number to changes to schema entities to enable you to track and extract changes by patch version.

path
Location of a resource in a REST API that maps to one or more REST methods and is represented in the OpenAPI specification; for example, the path/customer/name could be fulfilled by a getCustomer(name : String) or a deleteCustomer(name : String) method, depending on the verb specified in the request.

payload
User-supplied data block in the generic message exchange module. In some transports, considered to be just another property in Microsoft MQ (MSMQ). Transports can have payload length restrictions.

PCRE (Perl Compatible Regular Expressions)
Regular expression dialect.

persistent object
Object retained in the database between invocations of the application.

picture box
Displays a graphic from a bitmap, icon, cursor, or metafile.

Point primitive type
Used to represent a point in two-dimensional space. Encapsulates two integer values: the x (horizontal) and y (vertical) coordinates.

polymorphism
Ability for an object to behave in different ways, depending on the runtime classes of the objects involved. For example, calculateInterest operates differently for Savings and Mortgage accounts.

presentation client
Client node in the Jade thin client mode of operation. Communicates with one or more application server processes.

primary database role
Database with a primary database role is the only database in a Synchronized Database Environment (SDE) in which user applications can perform persistent updates.

primary database server
Database server node that manages the Jade database that currently has the primary role.

primary system
Jade system comprising nodes attached to the primary database server in an SDE, including the primary database server node itself. User applications in a primary system can directly update objects in the primary database.

primitive type
Has a defined null value, which can be tested for by using the null language identifier. Properties defined as primitive types represent a value, and not a reference to an object.

process
Activation of a single thread of control.

production database
Source Jade database replicated by the Relational Population Service (RPS). Also known as the primary database.

property
Characteristic of an object. Properties are used to encapsulate the state of an object. Can be an attribute or a reference.

pseudo type
Class for which an instance is never created, but which can be specified as a parameter type and return value in methods when the actual class may vary, depending on the object that is executing the method. Provides parametric polymorphism when the type returned by a method or the method parameters are dependent on the type of the object executing the method.

put
See send.

Q

QM (Queue Manager)
In the generic message exchange module, a non-application process that is responsible for receiving messages from senders (local and remote) and storing them until they are retrieved by an application. Can be responsible for inter-queue manager messaging, the persistent storage of queue definitions, generation and sending of reports (discards, delivery, and retrieval confirmations), and initiating an application when a message is placed into a queue (IBM WebSphere MQ, or WMQ, triggers). WMQ allows multiple QMs for each host. MSMQ allows one QM only for each host.

queue
In the generic message exchange module, entity to which messages are sent and from which messages are received.

queuing
Program communication through queues in the generic message exchange module. Programs communicating through queues do not have to execute concurrently.

R

RDB (Relational Database)
Single database that can be spread across several tables.

RDBMS (Relational Database Management System)
Stores data in the form of related tables.

Real primitive type
Represents a floating point number. Has a set of values that is a subset of real numbers. These values can be represented in floating-point notation with a fixed number of digits.

receive
Removing a message from a generic message queue. (Also known as get.)

receiver
Current instance of a method.

recovery
(1) Restart recovery, to automatically recover after a crash. (2) Archival recovery, to roll forward through database journals after restoring a backup.

rectangular format
Content copied by rectangular selection (see also stream format).

reference
End point in a bi-directional relationship. The other side (or end point) of the relationship is referred to as an inverse reference. An inverse may be explicit, in which the inverse reference is named, or implicit, when the inverse reference remains unnamed.

Regex
Regular expression library. See also PCRE.

relational attribute map
Part of a class mapping that defines how Jade attributes map to relational attributes (that is, to table columns).

relational class mapping
Defines how instances of a Jade class are mapped to rows in one or more relational tables.

relational database map
Defines a mapping between a Jade system model and a single relational database.

relationship
Denotes semantic connections among classes. Jade supports binary relationships; that is, relationships between two classes. The cardinality of relationships may be one-to-one, one-to-many, or many-to-many.

release deployment
Task of deploying a new or patched release of a Jade application in the field, including consideration of issues such as managing change dependencies and reorganizing user data.

remote repository
Common repository that all team members use to save their changes. It may or may not be stored on a code-hosting service online.

replayable database reorganization
Reorganization that is audited in the journals to support restart in case of interruption, roll-forward recovery, and replay on secondary databases.

reserve lock
Available for situations where you intend to update an object but you need to minimize the length of time the object is locked with an exclusive lock. When placed on an object, forces other processes attempting to acquire an exclusive lock or reserve lock on that object to wait until the reserve lock is relinquished.

resource
Object (or other entity such as image or multimedia file) against which REST requests can be made.

resource location
Second part of a URI that describes the location of a resource within a REST service.

REST (Representational State Transfer)
Architectural style for web services where objects are treated as “Resources” and HTTP verbs are used to manipulate those resources.

REST class
Jade class that is a subclass of JadeRestService and referenced in a REST web service application. The OpenAPI Specification describes the REST methods of this class, as well as any classes referenced by those REST methods.

REST method
Method of the REST class that begins with an HTTP verb and can be accessed by users of the REST API.

root class
Uppermost definition of a class.

RootSchema
Provides essential system classes (for example, the Collection classes and the File, Exception, and Form classes), which can then be accessed from all subschemas. The top of the schema hierarchy.

RPS (Relational Population Service)
Specialized secondary database server node that implements incremental object replication into a target relational database.

RPS data store
Native Jade database maintained by an RPS server node as temporary storage or to persist a mapped extent.

RPS node
Jade database server node that directly connects to a target RDBMS in order to replicate Jade objects to the relational database.

RSA (Rivest–Shamir–Adleman encryption algorithm)
Validates the signature of a JSON Web Token.

runtime-only environment
Installation of a Jade application in which the Jade Platform development environment is not present. (Also known as Jade deployment.) Allows a smaller base database to be used, but limits the development activities that can be performed on-site.

S

SAX (Simple API for XML)
Interface that specifies how objects in a Web page are represented.

schema
Complete logical description of the classes (together with their properties and methods) that model the application domain. Can contain one or many applications, each providing a different "view" of the object model.

schema context
Version of a schema (latest or current) selected in the Jade Platform development environment or for a schema load. See also version.

schema evolution
Changing the definitions of a system by adding, deleting, or changing entities such as schemas, classes, or properties.

schema file
Contains the schema metadata definition. (See also definition file.)

schema instantiation
See schema migration.

schema migration
Bringing the changes introduced by schema evolution to life. (Formerly known as schema instantiation.)

scroll bar
Control that provides easy navigation through a long list of items or a large amount of information. Can also provide an analog representation of current position. You can use a scroll bar as an input device or as an indicator of speed or quantity.

SDE (Synchronized Database Environment)
One or more connected database server nodes, including RPS nodes, that are providers or subscribers to the Synchronized Database Service (SDS). An SDE consists of a primary database (where updating occurs) and one or more secondary databases, which are transactionally consistent copies of the primary database. An SDE includes both SDS and RPS nodes.

SDS (Synchronized Database Service)
Database server node that manages a database with a secondary database role.

secondary database role
Database with a secondary database role is synchronized with a primary database by replaying journals automatically shipped from the primary database.

secondary database server
Database server node that manages a database with a secondary database role.

secondary system
Jade system comprising nodes attached to the primary database server.

self
Used in a method to denote the object that invoked the method.

Semantic Version Number
Defined as three integers delimited by periods; for example, 314.15.926.

send
Inserting a message into a generic message queue. (Also known as put.)

sender
Method that sends a specified message.

serialize
Convert an object into a standardized text representation, usually before transmission over the web.

server
Process implementing one or more operations on one or more objects. A server object is an object that is operated on by other objects; an object that provides certain services.

set
Unordered collection of objects. An object cannot be referenced in a set more than once.

SHA (Secure Hash Algorithm)
Family of cryptographic hash algorithms with the property that they are computationally infeasible; for example, when encrypting passwords, and hashing data and certificates.

shadow method
Generated by Jade and used by the REST service to get the required claim annotations associated with the REST API method.

shared lock
Lock acquired for read-only intent. Multiple users can acquire a shared lock. Prevents others from obtaining an exclusive lock.

sheet
See form.

signature
Defines the formal parameters of a specified operation, including their order, types, and passing mode (usage), and any return type.

single-value claim
Exact specified value must be present in the JSON Web Token.

slob
String large object. When the maximum length of a String property is not specified, the property value is separated from the parent object in the database, to simplify space allocation.

SOAP (Simple Object Access Protocol)
Lightweight XML-based messaging protocol used to encode Web service request and response message information before sending them over a network.

SSO (Single Sign-on)
Single sign-on is an authentication method that allows users to sign in to multiple independent software systems using one set of credentials.

status line
Special frame control that aligns itself to the bottom and width of its container by default, providing a status line for the container.

stream format
Content copied by using single or multiple carets (see also rectangular format).

String primitive type
Defines a String-type variable. A character string contains zero or more characters, and a null string is a string that has a zero length.

structural versioning
Versioning to make the reorganization phases aware which objects require updating.

stub
(1) Implementation of a package interface that supplies all of the classes, methods, and properties of the package but where each method is a stub that contains none of the functionality. (2) Method shell without any code automatically generated in implementing classes when a new method is added to a currently implemented interface.

subclass
Specialization of a class. Has the same characteristics of the parent classes (known as its superclasses). May define additional elements (properties), and may exhibit additional or different behavior (methods); for example, the Manager subclass, a subclass of the Employee class, that has special responsibilities.

subschema
Inherits all classes, methods, and properties that are defined in its superschemas.

subschema copy class
Copy of a class in a subschema of the schema in which the root class is defined. A subschema copy class is added to a subschema whenever a superschema class is subclassed in the subschema or has methods added to it in the subschema.

subschema copy final
Method that cannot be reimplemented in a subschema copy class.

subschema final
Class or method that can be extended or reimplemented in its local schema but not in a subschema. When applied to a class, there are no restrictions on the class in its local schema, but the class cannot be subclassed in subschemas. When applied to a method, there are no restrictions on the method in its local schema, but the method cannot be reimplemented anywhere in a subschema.

subschema hidden
Entity or feature available only in the local schema; that is, it is not available for use in any subschemas.

subscriber
Object invoking the beginNotification, beginClassNotification, or beginClassesNotification method.

superclass
Class from which another class inherits attributes and methods.

surrogate pair characters
16-bit Unicode code characters that include emojis and mathematical symbols.

synchronous messaging
Occurs in the generic message exchange module when the sending program waits for a reply to its message before continuing its own processing.

system
Database server and its associated client nodes.

system class
Standard class containing properties and methods to encapsulate the behavior of objects in your Jade applications.

T

table
Control that enables you to display entries in a table using rows and columns. Also enables you to have many different sheets in the table, of which only one is visible at any time.

TAF (Transaction Agent Framework)
Set of classes that work together to perform persistent storage of objects.

target
Class instance subscribed in a notification.

target database
Instance of a relational database connected to an RPS node.

text box
Control that displays information entered in the Jade Platform development environment, entered by the user, or assigned to the control by code at run time. Also called an edit field or edit control.

Time primitive type
Declares a variable representing the time of day since midnight to the nearest millisecond.

TimeStamp primitive type
Used to store the variable as type Timestamp; that is, the date and time.

TLS (Transport Layer Security)
Protocol that provides authentication, privacy and data integrity between two communicating computer applications.

transaction-id
Number, sequentially incrementing by 1 for each beginTransaction.

transient object
Temporary object that exists only for the duration of a session.

transition phase
Final phase of a schema migration where instances of versioned classes are no longer available to applications until the instantiation completes. Must be carried out offline.

transport
Specific messaging implementation in the generic message exchange module.

transport group
Encompasses the client connection pool, the worker pool, the listen connection, and so on, in a Jade multiple worker TCP/IP connection environment.

tuple
Data object containing two or more other objects as elements. (Analogous to a relational database record.)

type
Characterizes the behavior that can be applied to an instance or value. In Jade, types are classes, primitive types, and Jade interfaces.

type library
File or component within another file that contains type information about exposed objects.

type method
Method declared on a class, primitive, or interface type without having to have an instance of the type. At run time, the receiver is the type on which the method is declared.

U

UDR (Unstructured Data Resource)
Generic term for an instance outside the Jade database of unstructured data of arbitrary length.

UIA (Microsoft UI Automation)
An API that allows you to access and manipulate the user interface elements of another application

unversioning
Removal of the latest version of schemas to discard pending changes before those changes have been instantiated.

updating method
Method that can modify properties in the object to which it is sent.

URI (Uniform Resource Identifier)
Generic term for all types of names and addresses that refer to objects (for example, a URL) on the Internet. Commonly used term when talking about REST.

URL (Uniform Resource Locator)
Provides the Internet address of a file.

UTC (Coordinated Universal Time)
Primary time standard by which the world regulates clocks and time.

UUID (Universally Unique Identifier)
128-bit label used for information in computer systems.

V

VBX (Visual Basic custom control)
Industry-standard controls and extensions specifically for 16-bit environments.

version
Instance of a structural change to a class (for example, adding, deleting, or changing a property) and other related structures. Jade uses these versions, for example, to reorganize the database and developers can use them to make structural changes to classes even when they are in use by running applications. (See also schema context.)

versioning
Creation of meta schema objects to represent both states of the system (current version and latest version). Enables schema evolution to occur.

W

WAN (Wide Area Network)
Group of networked workstations or LANs covering a large geographical area.

Web server
Microsoft Internet Information Server (IIS) or Apache HyperText Transfer Protocol (HTTP) Server.

Window class
Abstract superclasses of all Form and Control classes. Provides the ability to define properties and methods that apply to all forms and controls.

worker
Single-threaded Jade application, running on the same Jade node as the original Jade application that spawned it.

worker pool
Encompasses the group of Jade processes that are linked to the transport group, in a Jade multiple worker TCP/IP connection environment.

Workspace
Editor pane that enables you to write a Jade method that does not to form part of your application.

WPF (Windows Presentation Foundation)

User Interface (UI) framework that creates desktop client applications for Windows.

WSDL (Web Services Description Language)

XML-formatted language that describes the capabilities of a web service as collections of communication end points capable of exchanging messages.

X

XML (Extensible Markup Language)

Web service architecture that enables you to create your own customized tags for web documents and also used for serializing objects before transmission over the web, albeit less commonly than JSON.