



User Guide

Together[®] ControlCenter[™]
Version 6.0

©2002 TogetherSoft Corporation. All rights reserved. Patents pending. Together® is a registered trademark of TogetherSoft Corporation. Together ControlCenter, TogetherSoft, Model-Build-Deploy Platform, and Improving the ways people work together, are trademarks of TogetherSoft Corporation. Java, Java2, Javadoc, JSP, Solaris, J2EE, and EJB, are all registered trademarks of Sun Microsystems, Inc. Linux is a trademark of Linus Torvalds. Visual Basic 6, Visual Basic.Net, and Visual C#, are all registered trademarks of Microsoft Corporation. The Coad Letter is a registered trademark of Object International, Inc. Oracle9i is a trademark of Oracle Corporation. Other trademarks or service marks referenced herein are property of the respective owners.

TABLE OF CONTENTS

Preface 37

Audience 37

Together Documentation Set 37

Finding out more 38

 Third-Party Books 38

 On the Web 39

 Workshops, Mentoring, and Consulting 39

Getting Help 39

 TogetherSoft corporate site 39

 Together community site 39

GETTING STARTED 41

Introducing Together 42

Getting started with a sample project 42

 Opening a project 43

 Opening a diagram 44

 Observing correlation between model and code 44

 Activating special features 45

Navigating Together's main window 46

 Arranging panes, inspector, and Designer toolbox 46

 View or hide a pane 46

 Resize or expand panes to full screen 47

 Undocking and moving panes 48

 Hiding pane title bars 48

 The main menu, toolbar, and status bar 48

 Main menu 48

 Main toolbar 49

 Status bar 50

Explorer pane 51

 Directory tab 52

- Server tab 52
- Model tab 52
 - Opening diagrams and source code files 53
 - Model, Favorites, and Diagram tab toolbar 54
- Favorites tab 54
- Diagrams tab 55
- Modules tab 56
- Components tab 56

- Special Features tabs 57
 - User Interface Builder tab 57
 - Test tab 57
 - XML tab 58
- Explorer pane operations 58
 - Right-click menus 58
 - Quick project access 58
 - Copying and pasting 58
 - Finding the locations of nodes, files, and folders 59
- Designer pane 59
 - Opening and closing diagrams 59
 - Designer pane toolbar 60
 - Designer tools 60
- Editor pane 61
- Message pane 62
- Property Inspectors 63
 - Overview of Inspectors 63
 - Inspector tabset 64
 - Properties tab 64
 - Hyperlink tab 64
 - View tab 65
 - Description tab 66
 - Javadoc tab 66
 - Requirements tab 66
 - Bean/C++ Properties tab 66
- Workspaces 67
 - Accessing different workspaces 67
 - Managing workspaces 67
 - Saving workspaces 68
 - Role-based workspaces 68
 - User roles 68

Changing the configured role 69

Roles and view management 69

Setting Your Personal Preferences 71

Setting-up workspaces 71

Accessing different workspaces 71

Managing workspaces 72

Saving workspaces 73

Role-based workspaces 73

User roles 73

Changing the configured role 74

Using view management 74

Controlling the level of detail shown in diagrams 75

Controlling how members display in diagrams 75

Showing and hiding subpackage contents in diagrams 75

Controlling how Java Beans/C++ properties display in class diagrams 75

Showing and hiding referenced classes in diagrams 76

Showing dependencies between classes and interfaces 76

Controlling the display of sequence diagrams 76

Banning destinations in class diagrams 76

Showing or hiding aggregations of diagram and EJB elements 77

Configuring Options 78

Activating and deactivating modules 78

Overview of configuration levels 79

Setting options in default mode 79

Setting options in advanced mode 80

Viewing and editing options 82

Values indicated by checkboxes 83

Accessing context-sensitive help for nodes and options 83

Resizing the Options dialog 83

Reference guide to options 83

Default and project level options 84

Diagram level options 89

Common Configuration Tasks 89

How to make Association links display a directional arrow 89

How to change the default source file header for the generated code 90

How to customize the default settings for C++ 90

How to create and use custom snippets for source code and text 90

- How to configure Stereotypes 91
- How to customize Inspector properties 91
- How to hide and show elements 91
- How to set options to control the formatting of your source code 91
- How to enable mouse-wheel support (Windows only) 92
- How to set up Together and projects to interact with version control 92

CREATING AND MANAGING PROJECTS 93

Working with Projects 94

Understanding Project Basics 94

- Primary root directory 94
- Project file 95
- The <default> diagram 95

Creating new projects 95

- Preparing to create a project 96
 - New projects without existing code 96
 - New projects with existing code 96
- Using the New Project Expert 96
- Using the New Project dialog 98

Editing project properties 99

- Adding Resources 101
- Removing Resources 102
- Setting Resource Options 103

Running multiple project instances 104

Setting-up version control for projects 104

Setting up large projects 105

- Creating views with referenced content 105
- Performance tuning 105

Reverse Engineering 107

Reverse engineering techniques 107

- Creating a new project with existing code 107
- Incorporating code into an existing project 109

Reverse engineering archive files 109

Importing and Exporting Information 111

Overview of import and export operations 111

- Importing and exporting JDBC database information 112
 - Activating database import and export 112
 - Generating DDL (exporting to DDL) 112
 - Importing database meta-information 114
- Generating IDL 114
 - Exporting IDL from class diagrams 114
 - Round-trip IDL support 115
- Importing and exporting XMI models 115
 - Importing XMI models 115
 - Exporting XMI models 116
 - User configurable translations 116

Importing Database Information 118

- Importing-Exporting JDBC Database Information 118
- Importing database meta-information 119

MODELING WITH TOGETHER 120

Introduction to Modeling 121

- Overview of modeling 121
- Diagrams supported in Together 122

Working with Diagrams 124

- Creating diagrams in projects 124
 - Using the main menu or toolbar 125
 - Using the Hyperlinking feature 125
 - Cloning diagrams 126
 - Renaming diagrams 126
- Configuring diagram options 126
 - Options for improving graphics 126
 - 3D look 127
 - Anti-aliasing 127
- Drawing diagram elements 127
 - Undo/Redo 128
 - Using the grid 128
 - Drawing node elements 128
 - Adding multiple elements 129
 - Drawing relationship links 129

Drawing links with bending points	130
Drawing a link to self	131
Drawing dependency links	131
Filtering out autodependency links	131
Working with node elements and links	132
Selecting elements	132
Right-click menus	132
Moving elements and drag-drop copying	132
Full drag-and-drop support	133
Copying and cloning elements	133
Resizing node elements	133
Converting bidirectional links	134
Labeling links	134
Rerouting links	135
Viewing diagrams and managing diagram elements	136
Viewing	136
Zooming	136
Using the automated layout features	136
Creating your own manual layout	137
Tips and tricks	137
Editing diagrams	137
Opening and closing diagrams	138
Editing properties	138
Opening the Inspector	139
Modifying properties	139
140	
Applying changes	140
In-place editing	140
Hyperlinking diagrams	141
Why use hyperlinking?	141
How to create hyperlinks	142
Hyperlinking to a new diagram	142
Hyperlinking to an existing diagram or diagram element	142
Hyperlinking to a URL or file	143
Viewing hyperlinks	143
Browsing hyperlinked resources	143
Removing hyperlinks	144
Annotating diagrams	144
Using Notes	144

The Note Link	144
Inspector documentation tabs	145
Standalone Design Elements	145
Creating SDEs	145
Using SDEs	146
Saving and copying diagram images	146
Copy - Paste within Together	146
Copy Image	146
Save Image	147
Printing diagrams and source code	147
Setting Print options	147
How to print diagrams	147
How to print text	148
Using auto-layout for printing	148
Printing generated documentation	148
Troubleshooting	148
Tips and tricks	149

UML Modeling 150

Notation and stereotypes of UML diagrams	150
Use case diagrams	151
Diagram elements	151
Creating browse-through sequences of use case diagrams	152
Showing other diagrams in a use case diagram	153
Adding an extension point	153
Class Diagrams	154
Diagram elements	154
Setting-up shortcuts	156
Showing Java Beans	156
Working with package diagrams	157
Viewing packages and their content	157
Opening packages	157
Renaming a package	157
Moving elements into packages	157
Moving nodes from packages	157
Showing project content from different packages	157
Dependency links	158
Deleting a Package	158

- Showing classes on search paths 158
- Defining inner classes 159
- Importing other diagrams 159
- Showing Association, Aggregation, and Composition 159
- Editing Members and Properties 160
 - Adding and editing members 160
 - Adding and editing properties 161
 - Rearranging the order of Attributes and Operations 161
 - Setting compartment controls 161
- Sequence and Collaboration Diagrams 162
 - Diagram elements 163
 - Creating sequence and collaboration diagrams using the expert 165
 - Using the Options dialog to control a sequence diagram's generation 165
 - Converting between sequence and collaboration diagrams 166
 - Refining sequence and collaboration diagrams 166
 - Techniques for sequence and collaboration diagrams 166
 - Creating a Message-to-self 166
 - Creating a Message link that calls an operation 166
 - Reordering Message links 167
 - Techniques specific to sequence diagrams 167
 - Adjusting the default lifeline of objects 167
 - Adjusting the size of object lifelines 167
 - Changing the order of a sequence diagram 167
 - Marking a destroyed object with an "X" 167
 - Indicating periods of time 168
 - Nesting messages 168
 - Specifying properties of message links 168
 - Techniques specific to collaboration diagrams 168
 - Setting-up shortcuts for sequence diagrams 168
 - Using Sequence automation to analyze patterns 169
 - Analysis of the Singleton pattern 169
 - Analysis of the composite pattern 170
 - Generating implementation source code 171
 - Creating project and class 171
 - Generating a sequence diagram from a class 171
 - Creating source-generating elements in the sequence diagram 171
 - Creating message sends 172
 - Generating implementation code 172
- Statechart Diagrams 173
 - Diagram elements 173
 - Drawing a self-transition 175

- Creating internal transitions 175
- Specifying entry/exit actions for a state 175
- Creating nested substates 175
- Showing multiple transition sources or targets 176
- Tips for modeling complex states 176
- Activity Diagrams 177
 - Diagram elements 178
 - Working with activity diagrams 179
- Component Diagrams 179
 - Working with component diagrams 181
- Deployment Diagrams 182
 - Diagram elements 182
 - Working with deployment diagrams 183

UML Extension Diagrams 185

- Entity relationship diagrams 185
 - Notation 186
 - Logical and physical diagram view 187
 - Diagram elements 188
 - Entities 188
 - Attributes 188
 - Relationship links 189
- Business Process Diagrams 190
 - Diagram elements 191
- Robustness analysis diagrams 192
 - Diagram elements 193
 - Key elements and properties 193

Real-time Modeling 195

- Introduction to Together's real-time module 195
 - Capabilities of the real-time module 195
 - Getting started 196
 - Real-time development process 196
 - Gathering system requirements 196
 - Deciding system architecture 196
 - Analyzing subsystems 197
 - Designing subsystems 197
 - Implementing subsystems 197
- Special real-time diagrams 197

Use case diagrams	198
Use case stereotypes	198
Activity	199
Exception	199
Use case links	199
Extend	199
Actors	199
System context diagrams	199
System context diagram elements	200
Real-time inspector properties	201
System requirements	201
Creating a new system requirement	202
Properties of functional requirements	202
Properties of quality of service requirements	203
System architecture diagrams	203
Subsystems	205
Subsystem Properties	205
Responsibilities	205
Statechart diagrams	206
Event sheet diagrams	206
Interaction diagrams	208
Interaction diagram elements	209
Object groups	209
Asynchronous messages	209
simple asynchronous	209
sender waiting reply	210
receiver polling reply	210
time stamp	211
process delay link	211
Together real-time audits	211
Simulation	212
Preparing a simulation	212
Event properties	212
The ops simulation unit	213
Rates of occurrence	213
Interprocess messages	213
Preemption	214
Priority	214
Running a Simulation	214
Simulation controls	214
Simulation chart settings	214
XML export	215

Simulation results 216

XML Modeling 217

XML Structure Diagrams 217

Content of XML Structure diagram 218

Creating XML Structure Diagram 219

Creating design elements of XML Structure diagram 219

Format of XML Structure Diagram 220

Working with DTD-specific components 221

XML Modeling Step by Step 222

Creating XML Structure diagram 222

Creating elements, links, and attributes 222

Creating and using complex type and data type 223

Creating re-usable (global) attributes 225

Creating groups 225

Import and Export Operations 226

DTD/XSD Import-Export 226

XML Serialization 227

PROGRAMMING WITH TOGETHER 229

Working with Code 230

Editing 230

Configuring the Editor 231

Using the Editor 232

Edit menu commands 234

Editor right-click menu commands 234

Bookmarks 234

Global Bookmarks 235

Setting and removing global bookmarks 235

Viewing, editing, and classifying global bookmarks 235

Editing global bookmark descriptions 236

Reordering bookmarks 236

Classifying global bookmarks 236

Deleting bookmarks 237

Navigating with global bookmarks 237

Local bookmarks 237

Setting and removing local bookmarks 237

Navigating with the local bookmarks 238

Breakpoints 238

- Setting and removing breakpoints 238
- Working with breakpoints 238
- Advanced Editor Features 239
 - Code Completion 239
 - Code Sense 239
 - Example 1 240
 - Example 2 240
 - Advanced Code Sense 240
 - Type casting 240
 - Instantiating new objects 241
 - Suggesting commonly used names 241
 - Import Assistant 241
 - Snippets 241
 - Defining Snippets 242
 - Deleting Snippets 242
 - Using Snippets 242
 - Browse Symbol 242
 - Adding library classes to the sourcepath 242
 - Context Help 243
 - Creating a help database 243
 - Using context help 244
 - Integrated XML Editor 244
 - Working with the XML Editor 245
 - Developing XML file structure on the XML tab 246
 - Developing XML file structure using the table view 247
 - Editing XML, JSP, and HTML files with XML support disabled 248
 - Code Sense in JSP Editor 248
 - Viewing HTML files 249
 - Tag Library Helper 249
 - Tips and Tricks 249
 - Rectangular Blocks 249
 - Split Pane 249
 - Showing - hiding the Editor pane 250
 - Using the Editor with an open project 251
 - Using the Editor with no open project 251
 - External Editor 251
 - Configuring the external editor 251
 - Using the external editor 252
- Search Facilities 252
 - Find and Replace 252
 - Find and replace in files 253
 - Search on diagrams 254

- Query-based search 254
- Go to line 256
- Search for Usages 257
- Compile - Make - Run 258
 - Using Compile and Make from Together 258
 - Executing the compile and make tools 258
 - Configuring and using the standard compiler 259
 - Compiler output 260
 - Cross-compilation 260
 - Configuring and using an alternative compiler 261
 - Running programs in Together 261
 - Run/Debug Configurations 261
 - Makefile Generation 262
- Debugging 263
 - Using the Integrated Debugger 263
 - Starting a Debugger session 263
 - Run/Debug Tab 263
 - Controlling program execution 264
 - Breakpoints 265
 - Setting breakpoints 266
 - Modifying breakpoint properties 267
 - Evaluating and Modifying Variables 268
 - Displaying structured context 268
 - Evaluating arrays 268
 - Evaluating and modifying objects 268
 - Watches, Threads and Frames 269
 - Using Watches 269
 - Change Display Range 270
 - Change Values 270
 - Change display format 270
 - Using Threads and Frames 270
 - Monitors 270
 - Monitors in deadlock and non-deadlock state 271
 - Attaching to a Remote Process 271

Using the UI Builder 272

- Overview of the UI Builder 272
- Activating the UI Builder 273
 - Deactivating the UI Builder 274
- Setting UI Builder options 274
 - Choosing a profile 274

Customizing a profile	274
Profile options	274
Creating visually editable UI classes	275
What is a visually editable class?	275
Creating UI classes	275
Using the UI Designer	277
The UI Designer view	277
The Menu Designer view	277
Adding components from the Toolbox	279
Editing component properties	279
Exposure levels of properties	279
Property editors	280
Changing the layout manager for UI components	281
The layout property	281
Using layout setup	281
Supported layout managers	282
Placing components into containers with different layouts	282
Where to learn more about layout managers	285
Other tips and tricks for working on UI components	285
Selecting multiple components	285
Changing the look and feel	285
Changing the order of components	285
Changing alignment, spacing, and distribution	286
Creating event handlers for UI components	287
Designing Menus	288
Creating a new menu bar component	288
Setting a menu bar into the container frame	288
Understanding the code	289
Creating a new popup menu component	289
Displaying a popup menu at runtime	290
Defining menus visually	290
Defining new menu items	291
Menu definition syntax and syntax helpers	291
Using the menu definition syntax helpers	292
Defining a checkbox menu item	292
Changing the default state of a checkbox	293
Defining a radio-button menu item	293
Changing the default state of a radio-button	293
Creating nested menus	294
Inserting separators	294
Creating separators using in-place editing	294

Creating separators using menu item's right-click menu	294
Creating separators using the Toolbox	295
Moving menu items	295
Defining menu item properties	295
Disabling and hiding menu via the right-click menu	295
Adding icons to menus	296
Icon properties	296
Defining menu colors and font	297
Defining menu items non-visually	297
Renaming menu component identifiers visually	298
Renaming component identifiers in source code	298
UI Builder Audits and Metrics	299
UI Audits	299
UI Metrics	299
Generating user interface documentation	300
Customizing the Toolbox component palette	300
Creating a component archive	300
Invoking the Toolbox customization dialog	301
Customizing toolbox categories	301
Reordering existing component categories	301
Creating new component categories	301
Removing component categories	302
Restoring the default component categories	302
Adding custom bean components	302
Adding components to a category	302
Removing components	303
Changing component display names	303
Changing component icons	303

Using Version Control 304

Multi-user development	304
Overview of version control support	305
Using Together with a Version Control System	305
Getting started with version control	305
Configuring Together for version control	306
Enabling version control support	306
Choosing which VCS to use	306
Configuring system-specific version control options	308
Setting up CVS options	308
Setting up CVS Local or LAN-based repository	308
Setting up CVS Pserver connection	309

- Setting up direct connection to the server 309
 - Setting up an SCC-compliant versioning system 309
 - Coroutine classes 310
 - Setting SCC Version Control options 310
 - Switching among different SCC providers 310
 - Tuning SCC support for Visual SourceSafe 311
- Enabling version control for projects 311
 - New projects 311
 - Existing projects 312
 - Together files to include in version control 313
- Interacting with version control 313
 - Examining differences 314
 - External visual diff tools 314
- Product-specific VCS notes 314
 - PVCS command line tools 314
 - PVCS Dimensions 314
 - Problems and workarounds 315
 - Configuring PVCS Dimensions to work with Together's IDE 315
 - Continuous/CM 316
 - Known issues 316
 - Perforce 317
 - Known issues 317
 - Problems and workarounds 317
 - PVCS Version Manager 317
 - Known issues 317
 - StarTeam 318
 - Known issues 318
- Using Together with ClearCase 318
 - Setting up the interface 319
 - ClearCase options 319
 - Basic Version Control 320
 - Add 321
 - Update 321
 - Check-in 321
 - Uncheckout 321
 - Checkout 321
 - System 321
 - Advanced Version Control 322
 - History 322
 - Remove 322
 - Difference 323
 - Details 323

Refresh 323
Reporting Results and Errors 323
Unified Change Management 324

GENERATING DOCUMENTATION 326

Generating Documentation for Together Projects 327

Generating documentation for an open project 327
 Documentation generation parameters 328
 RTF documentation options 329
Automating documentation generation 330
 Launching the documentation generator from the command line 330
 Generating HTML and RTF documentation from the command line 331
 Documentation generation options 331
 RTF options 331
 HTML documentation options 332

Designing Custom Documentation Templates 334

Template Organization 334
 Body sections and the DocGen engine 335
 Body section types 336
Metamodel types 336
Creating new templates 338
 Setting template properties 339
Designing the template body 340
 Creating, arranging, and resizing sections 340
 Setting section properties 341
 Output Styles 341
 Other options 341
 Enabling Conditions 341
Accessing model elements through iterator sections 342
 Element iterators 342
 Element iterator scopes 343
 Sorting 344
 Element property iterators 344
 Folder sections 345
Reusing templates and stock sections 345

- Calls to stock sections 345
 - Creating calls to stock sections 345
 - Creating and editing stock sections 346
- Calls to template sections 346

Documentation Template Controls 348

- Using static sections, headers, and footers 348
 - Creating and deleting static sections, headers, and footers 349
 - Setting area properties 350
- Creating controls and setting control properties 350
 - Labels, images, and panels 350
 - Labels 350
 - Images 351
 - Panels 351
 - Data controls 352
 - Data Control Sources 352
 - Displaying custom properties 353
 - Formula and text controls 353
 - Formula controls 353
 - Include text controls 354
- Managing the display of output 354
 - Moving and resizing controls 354
 - Text format settings 355
 - Aligning controls 355
- Hyperlinking controls to element documentation 356
 - Making a target from a static section, footer, or header 356
 - Linking a control to a target 356

Creating Multi-Frame HTML Documentation 358

- Multi-frame HTML document template basics 358
 - Frameset document template organization 359
 - Creating frameset templates 359
- Defining the frameset structure 359
 - Specifying FRAMESET properties 360
 - Specifying Frame properties 360
- Designing the frameset template body 361
 - Frameset template body organization 361
 - Frameset template processing 361
 - Call to template section properties 362

- Naming the generated document 362
 - Naming the output directory 362
- Creating hypertext links (advanced) 363
 - Assigning a target frame to a link reference 364
 - Targeting an element's "specific" documentation 364
 - Image mapping diagram elements 365
 - Creating compound link references 365
 - JavaDoc link references 366
 - Converting JDRefs into hyperlinks 366
 - Converting {@link} tags 366
 - Converting the value of an element's property 366

Documentation Generator and Template Designer Reference 369

- Documentation generator variables 369
- Documentation generator functions 372

CONTEMPORARY SOFTWARE PRACTICES 384

Refactoring 385

- Enabling Refactoring 385
- Showing code structure 386
- Moving classes, interfaces, attributes, and operations 388
 - Moving classes and interfaces 388
 - Moving attributes and operations in the class hierarchy 388
- Renaming 390
- Encapsulating attributes 391
- Extracting interfaces, superclasses, and operations 392
 - Extracting interfaces and superclasses 392
 - Extracting operations 393
- Summary of refactoring commands 396

Using the Testing Framework 397

- Overview of the testing framework 397
- Before you begin 398
 - Activating the testing framework module 399
 - Accessing the sample test project CashSales 399
- Creating a test project 400

Configuring options	402
Testing framework options	403
Options for visual steps	403
Options for test suites	403
Options for JUnit steps	404
Options for building unit tests	404
Support for JUnit	404
Support for JUnitX	404
Support for Cactus	404
Support for HttpUnit	405
Developing unit tests for source code	405
Setting-up filter options for the test builder	405
Generating stubs for test cases	406
Writing test cases	406
Creating a test suite	407
Importing JUnit tests	407
Creating and configuring collection suites	407
Using templates to create test cases	408
Elements of the unit test builder configuration file	408
Using the XML editor	409
Working with unit tests	409
Running JUnit tests	409
Running a collection of tests	410
Generating a report in HTML	410
Developing visual tests for a user interface	410
Creating a test collection	410
Recording a visual script	411
Setting options for recording a visual script	413
Editing a visual test	414
Working with visual tests	415
Running visual tests	415
Creating an assertion	417
Setting-up the bootstrap loader for a test server	418
System Requirements	419
Installing the bootstrap loader	419
Setting run configuration options	419
Testing your application with Together	420

- Running a stand-alone test server 420
- Troubleshooting 421
- Network security 422
- Known problems and limitations 422

Templates and Patterns 423

Code templates 423

- Template properties 424
 - Using template macros 425
- Default properties in templates 425
- Browsing the available code templates 425
- Editing code templates 426
 - Applying source formatting 427
- Custom code templates 427
 - Working with the Code Template Expert 427
 - Creating custom code templates manually 429
 - Groups of templates 429
 - Displaying custom template names 430
 - User-defined macros 430
 - Creating templates from diagram elements 431

Patterns 431

- Concept of Patterns 431
 - Working with various languages 432
 - Elements generated by patterns 432
 - Behavior of a pattern 433
 - Creating patterns 433
 - Providing pattern description 435

Using templates and patterns 435

- Creating classes and links by patterns 435
- Creating members by pattern 436
 - Choosing a pattern for a Member 437
- Refactoring with patterns 437

Audits and Metrics 438

Overview of audits and metrics analysis 438

- How should metrics be used? 439

Running audits and metrics in Together 440

- Available audits and metrics 440
 - How to perform metrics analysis 441

Metrics for audit violations	441
How to perform audits	441
Output for audits and metrics	442
Sorting results	442
Updating results	442
Exporting results	443
Creating linked HTML reports for metrics	443
Saving and loading metrics results	444
Comparing metrics results	444
Graphic output of metrics results	446
Kiviat graph	446
Bar graph	447
Distribution graph	448
Printing results	450
Printing results from the table	450
Printing graphic results	450
Automatic correction of audit violations	450
Creating and using saved sets of metrics or audits	451
Language-specific metrics and audits	452
Metrics supported for Visual Basic .NET	457
Running audits and metrics from the command line	457
Usage	457
Options	458
Additional information sources	459

J2EE DEVELOPMENT 460

J2EE Support 461

Supported J2EE technologies	461
J2EE-supported diagrams	462
EJB Assembler diagram	462
Web Application diagram	462
Enterprise Application diagram	463
Taglib diagram	463
Application Client diagram	463
Resource Adapter diagram	464
References support	464
Security support	465

Designing and Developing EJB Modules 466

Visual assembly of EJB applications for deployment 466

Creating EJB Modules 467

 Creating a new EJB Assembler diagram 468

 Creating a new EJB module using the Object Gallery 468

 Visual design elements 469

 Key elements and properties 471

 Notation 471

Creating and developing EJBs 471

 Creating a project using existing EJB code 471

 Creating “one-click” EJBs 472

 Creating EJBs using the Object Gallery 473

 Local interfaces 475

 Customizing the default code for EJBs 477

 Configuring EJBs using EJB Inspectors 478

 Mandatory subtabs 478

 Optional subtabs 478

 Sharing home and remote interfaces 479

 Creating a second implementation class 480

 Deleting implementation classes with shared interfaces 481

Entity beans 481

 Entity bean Inspector 482

 General subtab 482

 General properties of the entity EJB 483

 Primary key 483

 References subtab 484

 EJB references 484

 EJB local references 485

 Resource references 485

 Resource-environment references 486

 Environment references 486

 Security role references 486

 Deployment properties subtab 486

 External access options 487

 Database options 487

 General options 488

 Additional options 488

 Fields subtab 488

 Methods subtab 488

 Create methods 489

 Finder methods 489

- Business methods 489
 - Home methods 490
 - Select methods 490
- Relationships subtab 490
- Creating a bean-managed entity EJB 491
- Creating a container-managed entity EJB 492
- Session beans 494
 - Session bean Inspector 494
 - General subtab 494
 - General properties of the session EJB 495
 - Stateless or stateful session bean 496
 - References subtab 496
 - Deployment properties subtab 496
 - External access options: 497
 - General options: 497
 - Additional options: 497
 - Methods subtab 497
 - Create methods 498
 - Business methods 499
 - Creating a stateless session EJB 499
 - Creating a stateful session EJB 500
 - Using a session EJB as a Web service 501
- Message-driven beans 501
 - EJB implementation class, with: 501
- Message-driven bean Inspector 501
 - References subtab 501
 - Deployment properties subtab 503
 - External access options: 503
 - General options: 504
 - Additional options: 504
- Creating EJB shortcuts 504
 - An EJB shortcut 504
 - How EJBs are displayed in diagrams 505
 - Showing and hiding EJB elements 506
- Verification and Correction of EJBs 506
 - Verification overview 506
 - Using verification and correction 507
- How an EJB Assembler diagram relates to the J2EE Deployment Expert 507
- Tips for the EJB Assembler diagram 508

Working with Container-Transaction Elements and References 510

- What is a container-transaction element? 510
- Creating container-transaction elements 511
- Deploying container-transactions 513
- EJB References Support 513
 - Creating and using an EJB Properties element 513
 - Creating EJB references 515
 - The EJB References Inspector 517
 - Creating EJB local references 517
 - Working with EJB references 518
 - Sample: How to provide linking between two EJBs 519
 - Creating EJB resource references 520
 - Creating resource-environment references 521
 - EJB Security Reference Support 521
 - Creating EJB environment references 521
 - Deploying EJB references 525

Designing and Developing Web Applications 526

- Visual assembly of Web applications for deployment 526
 - Creating a new Web Application diagram 527
 - Creating a new Web application diagram using the Object Gallery 527
 - Visual design elements 529
 - Key elements and properties 530
 - Notation 530
 - Properties of a Web Application diagram 531
 - General subtab 531
 - Module name 532
 - Display name 532
 - Welcome File List 532
 - Additional options 532
 - Session time-out 532
 - Context Root 533
 - Servlet Context Parameters 533
 - Mime type mapping 533
 - Login Config subtab 533
 - Authentication method 534
 - Realm name 534
 - Form login page 534
 - Form error page 534
 - App event listeners subtab 534

Working with a Web Application diagram	535
Creating EJB shortcuts	536
Creating Container Transactions, EJB Properties, and References	536
Developing and designing applets	537
Creating an applet	537
Running and debugging an applet	538
Deploying an applet	540
Developing and designing servlets	540
Creating a servlet	540
Creating a new servlet using the Object Gallery	541
Running and debugging a servlet	543
Deploying a servlet	543
Developing and designing JSPs	544
Creating a JSP	544
Creating a JSP collection	545
Creating new JSPs using the Object Gallery	545
Debugging JSPs	546
Debugging JSPs under Tomcat	547
Deploying JSPs	547
Designing and developing Tag libraries	548
Taglibs	548
Taglib diagram	548
Main elements	548
Properties of the diagram	549
Properties of the elements	549
Creating Tag libraries	549
Using tag libraries	550
Tag Library Helper	550
Creating and mapping filters	551
Deploying filters	552
Developing and designing listeners	552
Event types	552
Creating a listener	553
Deploying listeners	554
How a Web Application diagram relates to the J2EE Deployment Expert	556
Tips for Web Application diagrams	556

Designing and Developing Enterprise Applications 558

Visual assembly of Enterprise Applications for deployment 558

Creating and drawing Enterprise Application diagrams 559

 Creating a new Enterprise Application diagram 559

 Creating an enterprise application using the Object Gallery 559

Visual design elements 560

 Creating diagram shortcuts 561

 Importing existing archive files 562

How an Enterprise Application diagram relates to the J2EE Deployment Expert 562

Designing and Developing Application Clients 564

Overview of Application Client diagram 564

Creating an Application Client 565

 Creating a new Application Client diagram 565

 Creating a new application client using the Object Gallery 565

 Visual design elements 566

 Creating “one-click” application clients 567

Using an Application Client diagram 567

Designing and Developing Resource Adapters 569

Integration Problems and Connector Architecture 569

Creating a Resource Adapter diagram 570

 Visual design elements 570

 Creating a Resource Adapter using a pattern 571

Using a Resource Adapter diagram 573

J2EE Platform Security Support 574

Security roles 574

Security References support 574

Process overview 575

Security support in EJB modules 576

 Declarative security role 576

 Programmatic security role 577

Security support in Web applications 579

 Using a Security Role design element for supporting security 579

 Authentication and authorization in Web applications 580

Security support in enterprise applications 583

Deploying J2EE 584

Deployment process 584

Supported application servers 585

Generic server options 585

Requirements for deployment 586

Starting application servers from within Together 586

Using the J2EE Deployment Expert 587

- Diagrams for deployment 587

- Specifying parameters 588

- Running the J2EE Deployment Expert 588

Server-specific information 590

- Deploying to IBM WebSphere Application Server 590

 - Setting project properties and environment, deployment steps 591

 - Process options 591

 - Deployment steps for IBM WebSphere 3.5 594

 - Common Properties page 594

 - EJB Deployment Properties page 595

 - Servlet Deployment Properties page 597

 - Client Properties page 599

 - Simple JSP Client Generation page 599

 - Command Line File with Instructions for Deployment page 600

 - Verify/Correct Sources page 601

 - Deployment steps for IBM WebSphere 4.0 602

 - Common Properties page 602

 - Deployment Properties page 602

 - Advanced Single Edition Properties page 603

 - Verify/Correct Sources page 604

 - Simple JSP Client Generation page 604

 - Command Line File with Instructions for Deployment page 605

- Deploying to BEA WebLogic Application Server 605

 - Starting BEA WebLogic Application Server from within Together 605

 - Setting project properties and environment, deployment steps 606

 - The J2EE Deployment Expert for BEA WebLogic Application Server 607

 - Process options 607

 - Common Properties page 610

 - Verify/Correct Sources page 611

 - Run-time Deploy page 611

 - Simple JSP Client Generation page 612

 - For BEA WebLogic 5.1 612

 - For BEA WebLogic 6.1 613

 - Command Line File with Instructions for Deployment page 614

Making transitions between application servers 614

- Transitions between specifications 614
 - Together verification and correction of EJBs 615
 - Manual correction of EJBs 615
 - How to move from EJB 2.0 to EJB 1.0 616
 - How to move from EJB 1.0 to EJB 1.1 616
- Transitions between application servers 617
 - Changes in the deployment descriptors 617
 - Changes in the client code 617
 - Changes connected with the EJB specification 617
 - Changes in the properties of the application server 617
- Developing and deploying clients for various servers 617
 - Correct findByPrimaryKey method 618
 - Correct localhost 618
 - Correct getInitialContext method 619
 - Correct lookup 619

WEB SERVICES 620

Designing and Developing Web Services 621

- Supported application servers 621
- Creating a Web service for deployment to a specific application server 622
 - Apache SOAP 622
 - Properties 623
 - Context 623
 - Other properties 623
 - Server-specific properties 624
 - Web Service ID 624
 - Message 625
 - CheckMustUnderstands 625
 - Scope 625
 - Type mapping properties 625
 - Mapping class name 626
 - Serializer 627
 - Deserializer 627
 - Result of type mapping on a Class diagram 627
 - Exposed methods 627
 - Expose 628
 - BEA WebLogic 6.1 629
 - RPC-style Web service 629
 - Message-style Web service 630
 - Web Service name 631

- Action 631
 - Destination JNDI name 631
 - Destination type 631
 - Connection factory JNDI name 631
 - URI 631
- IBM WebSphere AES 4.0 632
- Creating Web services by patterns 632
- Generating a WSDL file from a Web service 633
 - Generating a WSDL file using an existing Web service 633
- Generating a Web service client 634
 - Generating a proxy Web service client using a WSDL file 634
 - Generating a proxy Web service client for Apache SOAP by pattern 635
 - Properties of the pattern for an Apache SOAP proxy client 635
 - Generating a Web service client for BEA WebLogic by pattern 636
 - Properties of the pattern for a WebLogic client 636
- Generating a WSDL file and a WSDL client using UDDI Browser 637

Deploying a Web Service 638

- Deployment properties 638
- Running the Web Services Deployment Expert 638
- Web Services Deployment Expert for Apache SOAP 639
 - Process options 640
 - Add libraries required for deployment to the current project's Search/Classpath 640
 - Compile classes from the currently selected diagram 640
 - Generate Deployment Descriptor(s) 640
 - Clear temporary folder 641
 - Common Properties 641
 - Apache SOAP home directory 641
 - Run-time Registering Web Services 641
 - Server host name 642
 - Server port number 642
- Web Services Deployment Expert for BEA WebLogic 6.1 642
 - Process options 642
 - Compile classes from the currently selected diagram 643
 - Generate EJB Deployment Descriptor 643
 - Open XML editor for the generated Deployment Descriptor and build.xml 643
 - Pack modules for deployment 643
 - Assemble Web Services 643
 - Register Web Services on the server 643

Clear temporary folder...	643
Common Properties	644
Web Service context	644
Run-time Registering Web Services	644
Target server name	644
Server host name	644
Server port number	645
Web Service context	645
Protocol	645
Server root directory	645
Web Services Deployment Expert for IBM WebSphere AES 4.0	645
Process options	645
Common Properties	646
Deployment Properties	646
Node name	646
Server name	646
Server configuration file	646

UDDI Browser 648

Functionality of the UDDI Browser	648
Using the UDDI Browser	648
Searching for Web objects	649
Search Options pane	649
Search button	650
UDDI Browser Options	650
Properties subtab on the Search tab	652
WSDL subtab on the Search tab	652
Properties of Web businesses	652
Properties of Web services	652
Saving WSDL files	653
Generating a WSDL client	653
Publishing Web services	653
Logging on to the operator site	654
Publishing Web services on the operator site	654
Creating Templates	657

EXTENDING TOGETHER 660

Advanced Customizations 661

Customizing the user interface	661

Creating custom diagram types 662

Step 1: Creating icons and icon references for the UI 662

- Editing the Together.bat file 662

- Referencing images in the config file 663

Step 2: Creating the diagram configuration file 663

Step 3: Defining tree-view icons 663

Step 4: Defining toolbar icons 664

Step 5: Defining viewmaps 665

- Defining graphical presentation 665

- Defining the shape of an element 665

- Defining the element's options 666

 - Enable display of the element's unique name 666

 - Add the element's name to the compartment 666

 - Enable in-place editing 666

 - Define label properties 666

 - Define links 667

- Sample configuration file 667

Customizing Property Inspectors 668

Adding custom pages and fields to the Inspector 668

- API-based Inspector customization 668

 - More API documentation 669

- Customizing Inspector by means of the Custom Properties Module 670

- User-defined inspector customization with the Inspector Property Builder 670

Customizing View Management *Show* options 671

- Changing the display text of a *Show* option 671

- Removing a *Show* option from the Options dialog 672

- Adding a *Show* option to the Options dialog 672

Together Open API 674

Overview of the Together Open API 674

Understanding modules 675

- Locating modules 676

- Registering a module 676

- Starting modules 677

- Examples for Hello World 677

Extending Together using modules 678

- Types of modules 678

- Interfaces implemented by the modules 679

- Viewing modules 679

- Running modules 679
- Guidelines for developing modules 680
 - Applying naming conventions 680
 - Documenting the module 681
 - Order of Tags 682
 - Deploying the module 682
- Tutorial: Writing and deploying modules 682
 - Source code for the module 683
 - Declaring a module 684
 - Declaring a module in a Manifest.mf file 684
 - Declaring a module in a .def file 687
 - Specifying whether a module is a startup module 687
 - Defining a visible name for the module 687
 - Specifying the location of the Modules tab 687
 - Adding the module name to menus for classes, interfaces, and members 687
 - Adding the module name to menus of elements with specific shapetype 688
 - Rules for the manifest file 688
 - Compiling the module 689
 - Storing the compiled class 689
 - Evaluating the results 689
 - Troubleshooting your module 690
- Accessing more API documentation 691

Language Support 692

- Language Support 692
 - Switching languages on or off 693
 - Essential features 693
 - Language-specific configuration files 695
- Language-specific inspectors 695
 - Visual Basic inspector 695
 - C# inspector 696
 - CORBA IDL inspector 697
 - C++ inspector 697
- Using Together with C++ 698
 - Limitations on C++ projects 698
 - Defining C++ project structure 699
 - Setting the search path in large C++ projects 699

- Skipping standard includes 699
- Using the preinclude file 699
 - Managing include directives 699
 - Managing C++ macro definitions 700
 - Writing macro definitions 700
 - Default library support 701
 - Setting wrapper macro options for default libraries 702
- Configuring Together for C++ 702
 - Configuring C++ header and implementation file extensions 702
 - Specifying file extensions for code generated by Together 703
- Configuring C++ compile and make utilities 703
- Known problems and recent corrections 704
 - Deep parsing 704
 - Recognizing member operation definitions 705
 - Processing of enumerations, typedefs and global symbols 705
- Documentation of C++ member definitions 705
 - Using the definition documentation feature 706

Commands, Macros, and Keyboard Shortcuts 707

- Command Line Parameters 707
 - Basic command line syntax 708
 - Using the Windows launcher 708
 - Invoking the Together main class 709
 - Command-line examples 709
 - Standalone formatter 709
 - Documentation generation 710
 - Running modules at start-up 710
- Parameters for the Together.exe Launcher 710
 - Parameters for virtual machine preferences 710
- System Macros 711
 - Referencing configuration properties as macros 714
- Template Macros 715
- Keyboard shortcuts 715
 - Main Menu 716
 - Designer shortcuts 717
 - Editor/Compiler/Debugger shortcuts 720
 - Miscellaneous 720

PREFACE

The *User Guide* contains instructional and conceptual information for using Together[®] to build enterprise level applications. The guide assumes that you have installed Together on your computer. If not, refer to the *Installation and Administrator's Guide*.

Audience

The information in this guide is intended primarily for developers using Together to build applications. The documentation assumes that you have experience with the following:

- The basics of the Unified Modeling Language[™] (UML[™]).
- Your organization's process for designing and modeling object-oriented systems or components.
- Your target programming language(s).
- Fundamentals of development technologies such as Enterprise JavaBeans[™] (if you will use Together to develop and deploy an EJB[™], JSP[™], etc.)
- Your external tools and systems such as IDE's, version control systems, database management systems (DBMS), application servers, etc.
- How to install a Java[™] VM and launch a Java application with it

Together Documentation Set

The documentation set for Together consists of the items listed in Table 1.

Table 1 Together documentation set

Item	Description	Location
<i>Together 6.0 Readme</i>	Late-breaking information including: • Licensing notes • System requirements • Installing and starting Together • Known problems • Significant fixes	Your Together installation under \$TGH\$/readme.html
<i>Installation and Administrator's Guide</i>	Information most relevant to the Together administrator, including: • System requirements • Installing and configuring Together • Setting-up and using Together's floating license server	• PDF located as follows: • Your Together installation under \$TGH\$/doc/installGuide.pdf • http://www.togethersoft.com/downloads

Table 1 Together documentation set (continued)

Item	Description	Location
<i>User Guide</i>	Information most relevant to the user, including: • Introduction to Together • Setting personal preferences and options • Detailed instructions for using Together features • Advanced customizations • Reference information including commands, macros, and keyboard shortcuts	• PDF located as follows: • Your Together installation under \$TGH\$/doc/userGuide.pdf • http://www.togethersoft.com/downloads
Online Help	Information most relevant to the user, including: • Main window components • Brief instructions for using Together features with references to the <i>User Guide</i> for more details • Dialog help	Together's Help menu
Documentation for integrations with third-party products	Instructions for using Together integrations and plug-ins.	http://www.togethersoft.com/developers/integrations
API documentation	Technical reference for the Together API generated by Javadoc; Use in conjunction with API information in the <i>User Guide</i> .	• Together's Help menu • Your Together installation under \$TGH\$/doc/api/index.html
Practical guides	A collection of interactive tutorials that provide an introduction to Together for the new user.	http://www.togethersoft.com/services/practical_guides/

Finding out more

If you are just getting started with UML, object technology, or distributed application development, the following books and resources can help get you started.

Third-Party Books

UML Distilled: Applying the Standard Object Modeling Language by Martin Fowler. Addison-Wesley, 1997. ISBN: 0201325632

The Unified Modeling Language User Guide by Booch, Rumbaugh, and Jacobson. Addison-Wesley, 1998. ISBN 0-201-57168-4

Java Modeling in Color with UML: Enterprise Components and Process by Peter Coad, Jeff De Luca, and Eric Lefebvre. Prentice Hall, June 1999. ISBN: 013011510X

Java Design: Building Better Apps & Applets (2nd Ed.) by Peter Coad and Mark Mayfield. Prentice Hall, 1997. ISBN 0-13-271149-4 (pbk).

Java Enterprise in a Nutshell by David Flanagan, Jim Farley, William Crawford, and Kris Magnusson. O'Reilly & Associates, 1999. ISBN 1-56592-483-5

On the Web

Object Management Group: <http://uml.shl.com/>

Cetus Links site: <http://www.cetus-links.org>

Dev-x Developer Exchange: <http://www.devx.com/>

Workshops, Mentoring, and Consulting

If your organization needs help making a smooth transition into the world of object and component software technologies, TogetherSoft Corporation offers a wide range of workshops, seminars, mentoring, and consulting services to teams worldwide. For more information, visit the Services area at the TogetherSoft website at www.togethersoft.com/services/.

Getting Help

The TogetherSoft web site offers a comprehensive range of company, product, and technical information.

TogetherSoft corporate site

The TogetherSoft corporate site is located at <http://www.togethersoft.com> and provides company and product information such as:

- What features come with Together products (www.togethersoft.com/together)
- Together installation, startup & readme notes (www.togethersoft.com/readme)
- Worldwide Together technical support centers (www.togethersoft.com/support)
- Downloads: products, docs, JVMs, etc. (www.togethersoft.com/downloads)
- Integrations and plug-ins (www.togethersoft.com/developers/integrations)
- Newsletters: info, subscribe, unsubscribe (www.togethersoft.com/subscribe)

Together community site

The Together community site is located at <http://www.togethercommunity.com> and provides technical information, discussions, open-source code, and more including:

- FAQs
- Discussion forums and e-mail lists.
- Technical articles and presentations on modeling, Java, using Together, and more.
- Open source and binaries for modules and patterns.

- Latest Together Application Help online in HTML format.

PART

1

GETTING STARTED

- Chapter 1, “Introducing Together”
- Chapter 2, “Setting Your Personal Preferences”
- Chapter 3, “Configuring Options”

INTRODUCING TOGETHER

From its origins as the first modeling tool to fully synchronize modeling diagrams and source code, Together ControlCenter has grown to include all of the features needed to build enterprise level applications, allowing the entire development team to collaborate using common language, diagrams, and software. Together provides business people, developers, and project managers a single platform with a consistent but customizable user interface for all of their work throughout the entire software development cycle.

This chapter provides an introduction to the basic features of Together. It includes the following topics:

- “Getting started with a sample project” on page 42.
- “Navigating Together’s main window” on page 46.
- “Explorer pane” on page 51.
- “Designer pane” on page 59.
- “Editor pane” on page 61.
- “Message pane” on page 62.
- “Property Inspectors” on page 63.
- “Workspaces” on page 67.

Getting started with a sample project

Almost all work in Together is done in the context of projects. Together ships with a collection of sample projects in your Together installation under `$TGH$/samples`. There are sample projects for every supported language as well as projects for some special features.

The sample Java *CashSales* project is the basis for the tutorial steps in this section.

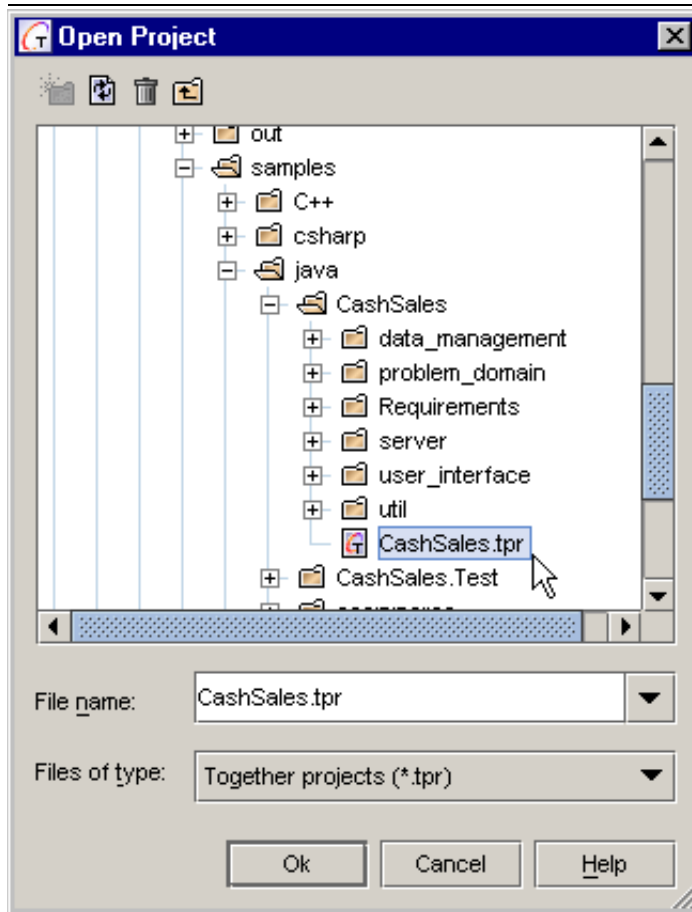
Opening a project

When you start Together for the first time, it opens with no project. There are several ways of opening an existing project or creating a new one.

To open the sample Java CashSales project from the main menu:

1. Select **File | Open Project**.
2. Pick the desired project from the resulting dialog box. Together project file names end in `.tpr`. (See Figure 1.) For the CashSales project, select `$TGH$/samples/java/CashSales/CashSales.tpr`.
3. Click **OK** to quit the dialog and open the project.

Figure 1 Opening the sample Java CashSales project.



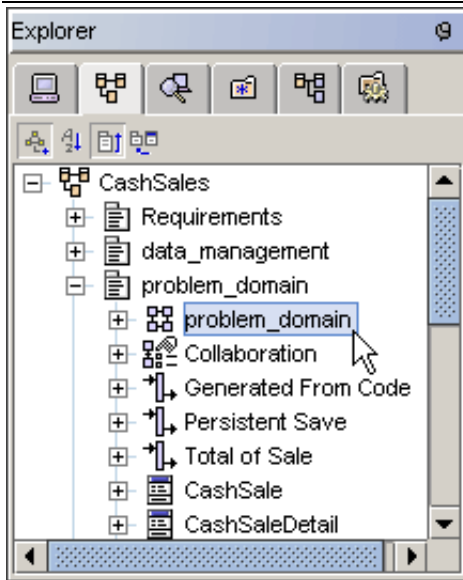
Opening a diagram

By default, the Explorer pane occupies the upper left side of the main window. You can open diagrams and source code files from the Explorer pane.

To open the `problem_domain` diagram:

1. Click the Model tab () at the top of the Explorer pane.
2. Expand the model tree to find the diagram: **CashSales | `problem_domain`**.
3. Right click the **`problem_domain`** diagram. (See Figure 2.)
4. Select **Open in New Tab** from the menu. The diagram opens in the Designer pane.

Figure 2 Selecting the **`problem_domain`** diagram from the Explorer.



Observing correlation between model and code

You can open source code files from the Explorer and from the Designer.

To open `CashSales.java` from the Designer:

1. Find the `CashSales` class node in the `problem_domain` diagram in the Designer.
2. Click **CashSales** once to select it. `CashSales.java` automatically opens in the Editor.

When you click on a member of a class node in the Designer, the declaration of that member highlights in the Editor.

Together always keeps model and code in sync. If you change the model, the code changes. If you change the code, the model changes.

To observe how changing a diagram also changes code:

1. Double click the attribute named `SALES_NEW` in the *CashSales* node of the *problem_domain* diagram.
2. Enter a new name and new type for the highlighted attribute.
3. Notice how the declaration of the attribute changes in the Editor.
4. Undo the change by clicking the undo button () on the main toolbar or typing Ctrl+Z.

To observe how changing code also changes diagrams:

1. Find the declaration of `SALES_NEW` in the file `CashSales.java`.
2. Rename `SALES_NEW` to `XXX`.
3. Move the cursor out of the Editor to change focus. (The source code automatically updates; there is no need to save it.)
4. Observe the new name `XXX` in the *CashSales* node in the diagram.
5. Undo the change to restore the sample project to its original state.

Activating special features

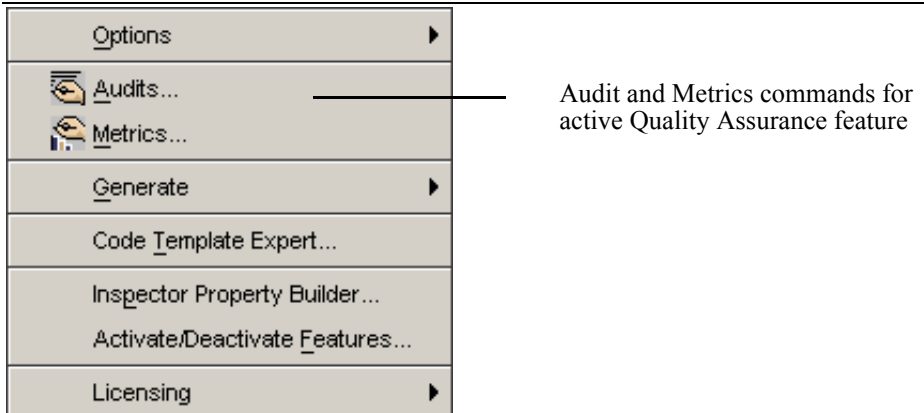
Together has an extensive collection of special activatable features, including audits and metrics, real-time modeling, the testing framework, and many others. Some of the features are activated by default. You must activate others that you want to use.

To activate the Quality Assurance feature for generating audits and metrics:

1. Select **Tools | Activate/Deactivate Features** from the main menu.
2. In the resulting dialog box, bring the **Together Features** tabbed page to the front.
3. Click the **Quality Assurance** checkbox on.
4. Click **OK** to close the dialog box and turn on activation.

Figure 3 shows the Tools menu with its new *Audits* and *Metrics* commands.

Figure 3 Tools menu with Quality Assurance activated



Navigating Together's main window

Together's main window is divided into four major panes.

1. Explorer: for file system and project navigation.
2. Designer: for creating UML and other kinds of model diagrams as well as for building graphical user interfaces. The Designer has a *toolbox* for its GUI construction tools.
3. Editor: for viewing and editing source code files and other text files.
4. Message pane: for system messages, special tasks, and results of some feature operations.

The *focus pane*, with the light blue title bar, is the site of the most recent activity.

Together *elements* are individual components of the project and the user interface. Elements can be diagrams, files, diagram elements such as nodes or links, names, error messages, and so on. Right clicking on a Together element displays a menu of commands for that element. These *right-click menus* vary according to the type of element. Figure 4 shows the right-click menu for a note on a diagram in the Designer pane, which is undocked from the main window.

Many elements have *Inspectors* for accessing the elements' properties. You can display the Inspector of an element by selecting Properties from its right-click menu (see Figure 4).

Arranging panes, inspector, and Designer toolbox

You can rearrange the windowing and layout of the panes, inspector, and the Designer toolbox. You can hide panes, resize them, and expand them to full screen. You can also undock panes from the main window or move them from one side of the main window to another.

View or hide a pane

You can view or hide any available pane, the Designer toolbox, and the inspector.

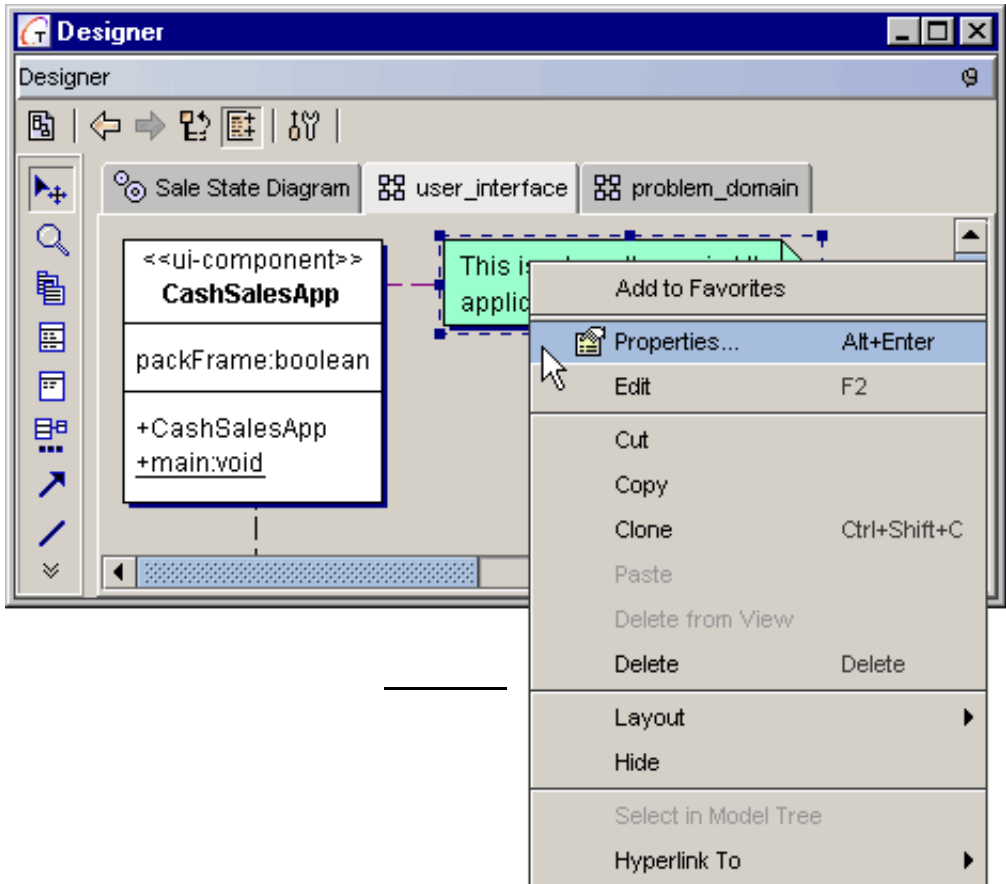
To view or hide a pane, toolbox, or Inspector:

- Click the corresponding view/hide button on the main toolbar (see “Main toolbar” on page 49), *or*
- Select **View | Main Panes | <name>** on the main menu, *or*
- Type the keyboard shortcut for the pane. The keyboard shortcuts for the different panes are listed on the **View | Main Pane** cascading menu.

Buttons and names of panes that are not available are shown in gray on the main toolbar and menu.

The Designer pane and toolbox are available only when there is an open project. The inspector is available only if the currently selected element has an inspector. (Files in the Editor and tabs in the Explorer do not have inspectors. However, diagrams in the Designer pane do have inspectors.)

Figure 4 Accessing an inspector from a right-click menu



Resize or expand panes to full screen

To resize panes, drag their separating bars.

To expand a docked pane to full screen:

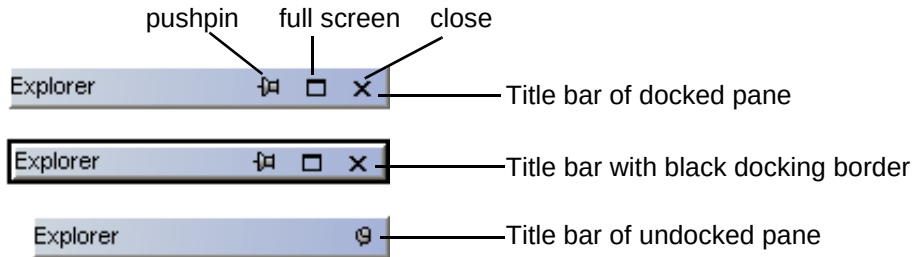
- Click its full screen expander box on the title bar (see Figure 5), or
- Select **Full Screen** from the right-click menu of its title bar, or
- Bring the pane into focus by clicking it *and*
 - click the **Full Screen** button on the main toolbar, or
 - select **View | Full Screen** from the main menu, or
 - press **F12**.

Repeat the actions listed above to return the pane to its previous unexpanded state.

Undocking and moving panes

Each pane in the main window has a push pin in the upper right corner of its title bar. (See Figure 5.) Clicking the push pin undocks the pane from the main window. Dragging the title bar off the pane also undocks it.

Figure 5 Title bar variations



To redock the pane back on the main window, click the push pin on the title bar of the undocked window. Alternatively, drag the title bar off the pane and onto the main window, releasing it when the border turns black. (See Figure 5.)

You can move the Explorer, Inspector, toolbox, or Message panes from their original docked positions to either side of the main window. Simply drag the title bar as described above, releasing it when the border turns black.

Hiding pane title bars

Even the title bars of panes, toolbox, and Inspector have right-click menus. The items on the right-click menu include those on the title bar itself; they change when switching between docked and undocked.

The last item on the right-click menu of each title bar is **Hide Title**. When you select it, the title bar becomes a narrow line at the top of the pane.

The main menu, toolbar, and status bar

The main menu and toolbar are at the top of the main window. The status bar is at the bottom.

Main menu

The main menu consists of a collection of menus, shown in Table 2.

Table 2 Main menu items

Menu	Meaning
File	Commands for interacting with the operating system. Use the File menu for the usual file commands as well as special commands such as Import/Export.
Edit	Editing commands plus “infinite” undo/redo for almost all operations. The Edit commands apply to both the Designer and Editor.

Table 2 Main menu items (continued)

Menu	Meaning
Search	Search and replace commands that apply to the Designer and Editor. Use the Search menu to find and replace text strings in source files, diagrams, and across multiple files in a project. You can also create custom queries for searching.
View	View options for the main window. Use the View menu to hide or show any of the panes. You can also maximize the currently selected pane. Note the keyboard shortcuts displayed on the menu.
Project	Commands and properties for the currently open project. Use Project to generate documentation, compile (Java), format source code, set bookmarks, and set project properties, including version control.
Run	Running and debugging projects. Use the Run menu to set breakpoints and control execution in the debugger.
Deploy	J2EE and Web Services deployment experts.
Selection	Commands relevant to the currently selected element. Selection replicates the right-click menu of the currently selected element. The menu content changes dynamically to reflect commands appropriate to the selection.
Tools	Options (default, project, diagram), feature activation and special feature commands, code template experts, and licensing. The Tools menu varies greatly according to which features are activated.
Help	Hypertext online documentation

The menus vary according to the workspace, the status of the open project and project options, the activated features, and the currently selected element.

Some of the items on the individual menus have cascading submenus. Many commands on the menus have keyboard shortcuts. For a complete list of keyboard shortcuts, see “Commands, Macros, and Keyboard Shortcuts” on page 707.

Main toolbar

Main menu commands that are frequently used have corresponding buttons on the main toolbar.

Table 3 Main toolbar buttons

Button	Name	Meaning
	New object	Displays the Object Gallery.
	New project	Closes the current project and displays the new project dialog.
	Open/Save/Print	Usual file operations.
	Undo/Redo	Available for most operations in the Editor and Designer.
	Cut/Copy/Paste	Usual edit operations.
	Find	Find in current file in Editor. (Search/Find across multiple files is available on the main menu.)

Table 3 Main toolbar buttons (continued)

Button	Name	Meaning
	View Explorer	Toggle to view or hide Explorer pane.
	View Editor	Toggle to view or hide Explorer pane.
	View Designer	Toggle to view or hide Explorer pane.
	Designer toolbox	Display Designer toolbox in a separate pane.
	Message pane	Toggle to view or hide Explorer pane. Message pane.
	Full screen	Toggle to maximize the main window focus pane.
	Inspector	View properties Inspector for current selection (toggle). It is available only when the selected element has an inspector.
	Make/Build	Make and build the current Java project.
	Run	Run the current Java project.
	Debug/Attach to Remote Process	Launch the integrated debugger/debug remote process. (Not available in all products.)
	Help	Launch online help system.

The rightmost item on the main toolbar is a menu for selecting the current workspace. (See “Workspaces” on page 67 for detailed information on workspaces.)

You can undock the main toolbar from the main window by dragging the “handle” at the left edge. Click the close button on the undocked toolbar to dock it.

Status bar

The status bar for the main window varies according to which panes are docked and in view. The full status bar shows the following items, left to right:

- Messages (button): Show or hide the Message pane; indicate when there is a new message.
- General info: This cell displays information about elements in diagrams as you move the mouse over them.
- Progress: This bar shows the progress of internal processes that you invoke while working in Together.
- Editor information: The remaining cells display information related to the Editor pane:
 - Changed file indicator
 - Insert/Overwrite mode
 - Current line number
 - Current character column position

Explorer pane

The Explorer pane enables navigation of your system, provides detail views of your project, and gives easy access to system and extension modules and reusable components. On the Explorer pane you can:

- View the physical and logical structure of your project.
- Navigate within the project as well as within your physical directory system.
- View project resources such as source files, classes and members, diagrams, and modules. Open diagrams and files for editing.
- View and run Together system and feature building blocks or any custom ones you develop.
- View test plans for Together's activatable Testing Framework feature.
- View and modify the properties of user interface components from Together's activatable UI builder.
- View the structure of an XML file and modify its elements.

The Explorer pane is organized into multiple tabs, as shown in Table 4. Some tabs display only with an open project or when special features are activated. By selecting View | Explorer Tabs from the main menu, you can toggle the display of three tabs: Server Explorer, Diagrams Explorer, and Favorites.

Table 4 Explorer pane tabs

Icon	Name	Meaning
	Directory	Physical structure of the open project and the file system. Does not require an open project.
	Model	Logical view of the project's model elements. Requires an open project.
	Server	Servers relevant to your Together installation. Does not require an open project; toggle on or off with View Explorer Tabs Server Explorer.
	Favorites	Frequently accessed ("favorite") model elements and files. Requires an open project; toggle on or off with View Explorer Tabs Favorites.
	Diagram	Listing of project diagrams by type. Requires an open project; toggle on or off with View Explorer Tabs Diagram Explorer.
	Components	Reusable model components. Requires an open project in which "include components" is checked on.
	Modules	Custom building blocks. Does not require an open project.
	UI Builder	Displays hierarchy of user interface components. Displays only when the Designer pane is in UI Design view.
	XML Editor	Displays structure of XML files. Requires an open project and activation of XML Support feature.
	Test	Testing Framework instructions and results. Requires an open project and activation of Testing Framework feature.

Directory tab

The Directory tab presents views of the contents of your physical system and enables you to navigate both inside and outside of a project's physical structure.

You can open text type files in the Editor pane that are independent of your current project or when no project is open. You can open a project file. When a project is open, you can also open a diagram file (in the Designer pane).

The Directory tab shows an icon beside each file, including these:

-  Edit file icon. Double click source code and text files to open in the Editor.
-  Together project icon. Double click to open the project in Together.
-  Generic file icon. Together cannot open these files.

When you launch Together, the Directory tab displays your available drives/directories and the following additional directories from your Together installation:

- Samples: contains example projects.
- User Projects: an empty directory where you can place your first experimental projects or real projects for quicker access from the Directory tab. In the latter case, you should create the Together project under the myprojects directory.
- Templates: contains numerous templates of classes, members, and links for the supported languages.

When you open a project, the Directory tab displays an additional directory at the top:

- Current Project: the physical files that comprise your project as well as files relevant to your project.

Server tab

The Server tab enables you to navigate through the application and database servers relevant to your project and to your Together installation. If the Testing Framework is activated, the Server tab shows Test Suites as well.

Model tab

The Model tab becomes visible when you open a Together project. It displays a logical view of the elements that comprise the project's model. (See Figure 6.)

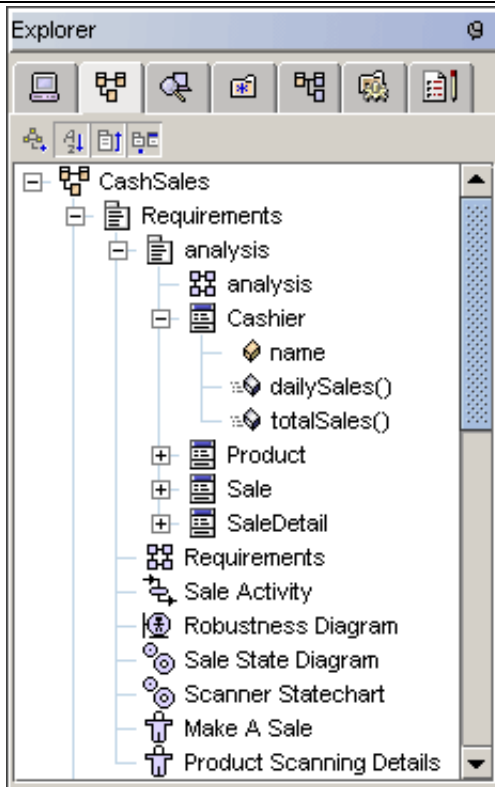
The Model tab displays the root-level packages that comprise your project, enabling you to logically navigate the contents to see the subpackages, diagrams, and diagram elements in each one.

NOTE: The Model tab's view is not strictly hierarchical in the same sense as a file system explorer because the project's root-level packages can physically reside anywhere on your system. The Model tab displays secondary root packages in relation to the primary root (the package containing the Together project file).

For example, you might create a project file in the directory C:\Project1, and

specify D:\mysources as the secondary root directory for the project. In this case, mysources displays as a package node under the primary root node (Project1). From that point on, the contents of mysources displays hierarchically.

Figure 6 Explorer Model tab



Opening diagrams and source code files

From the Model tab, you can open diagrams in the Designer and source code files in the Editor.

To open a diagram from the Model tab:

1. Double click the diagram or select **Open in New Tab** from the diagram's right-click menu to open the diagram without closing the diagram at the front, *or*
2. Select **Open** from the diagram's right-click menu to replace the diagram at the front if the Designer is already open. (You can set the project or default options to open in a new tab instead of replacing the front diagram.) *or*
3. Choose **Select in Diagram** from the right-click menu of a diagram element (class, interface, operation, attribute) to open the containing diagram if necessary and to highlight the element on the diagram.

NOTE: When you open a diagram from the Model tab by selecting a key element such as a class, member, or use case (depending on the diagram type) on the Model tab, the element is simultaneously selected in the diagram.

To open a source code from the Model tab:

1. Double click the class node in the Model tab or selecting **Edit** from its right-click menu. (Double clicking a class or interface member highlights the declaration of the member as well.) -- or,
2. Choose **Select on Diagram** from the right-click menu of a class node or member to open the corresponding diagram in the Designer and highlight the node or member.

When the Model tab is at the front of the Explorer, the right-click menu of a class or interface in the Designer pane has a *Select in Model Tree* command. This command highlights the node in the Model tab tree-view.

Model, Favorites, and Diagram tab toolbar

The Model tab, Favorites tab, and Diagram tab display a small toolbar (see Figure 6). The toolbar buttons control the presentation of information in the tree-view. From left to right, these buttons are:

- Expandable diagram nodes: controls whether or not you can expand nodes on the tab to show their contents.
- Sort node: controls whether or not nodes in the tree-view are sorted alphabetically.
- Packages first: controls whether or not packages display first in the tree-view before other content.
- Show fully qualified names: shows the fully qualified class and interface names when displayed in the Favorites tab.

Favorites tab

The Favorites tab can display the nodes from the model that you want to access without navigating the logical project hierarchy in the Model tab or in a diagram. Nodes can be diagrams, classes, or interfaces. The Favorites tab is particularly useful for large projects. The rightmost button on the Favorites tab toolbar is a toggle for showing the fully qualified names of classes and interfaces.

You can add an element to the Favorites from the Designer as well as from the Model tab of the Explorer.

To add an element to Favorites:

1. On the Model tab or in the Designer, find and select the element to be added to Favorites.
2. Right-click the element and choose **Add to Favorites** from the right-click menu.

You open elements in Favorites in the same way you would open them from the Model tab.

To remove an element from Favorites:

1. Select the element in the Favorites tab.
2. Right-click and choose **Remove from Favorites** from the right-click menu.

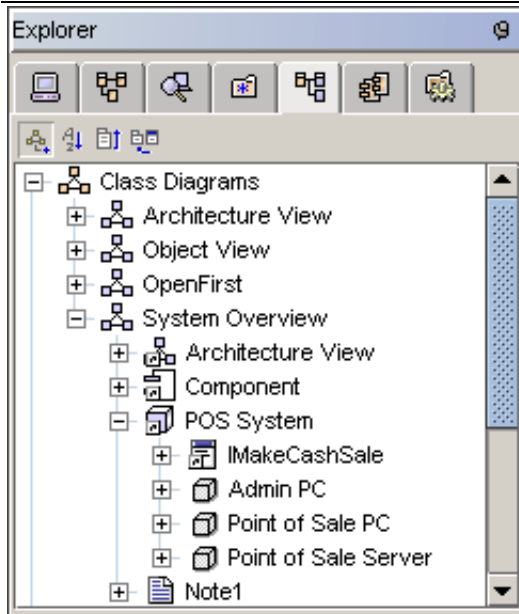
Diagrams tab

The Diagrams tab displays a list of all types of diagrams available in Together. The diagrams are listed according to their types. You can open the diagrams and source code files from the Favorites tab the same way as in the Model tab.

When Expandable Diagram Nodes (from the Diagram toolbar) is on, you can expand each diagram to see its elements. Elements on diagrams that are not actually contained in the diagrams appear as *shortcuts*. For example, if a class diagram displays classes that reside outside the class diagram's package, the nodes for these classes have shortcut symbols in both the tree-view and the diagram. Class diagrams can show classes and interfaces from packages that reside on the classpath or other search paths as defined in the project's properties.

Figure 7 shows the Diagram tab and several shortcuts: Component, POS System, and IMakeCashSale. Shortcut icons have the curled arrow in the lower right corner.

Figure 7 Diagram tab showing shortcuts



The Diagrams tab is available whenever there is an open project. You can leave it visible or hide it from view.

To hide the Diagrams tab:

1. Select **Tools | Options | Default (or Project) Level** from the main menu.
2. In the resulting Options dialog box, select **General** without expanding it.
3. Uncheck **Show Diagrams** tab.

Modules tab

Together is highly extensible. Using Java and the Together API, you can develop your own modules to do such things as create custom metrics or documentation and generate custom outputs based on model information. Indeed, many of Together's own features are implemented as building-block modules. These modules appear in the system folder on the Modules tab.

The Modules tab provides quick access to system, sample, and any added-in building blocks that are supplied with Together or developed or added yourself. You can view the available building blocks by navigating through the Modules tab folders. You can run any compiled or source files from the right-click menu of individual nodes. If you develop your own modular building blocks or acquire third-party building-block extensions, you can install them so that they display in and run from this tab.

The Modules tab uses the following special file icons:

Table 5 Module tab icons

Icon	Meaning
	Java source file for module. Can be compiled from "Run" when a compiler is configured.
	Compiled Java module.
	TCL script. "Run" executes the script in interpreted mode.

See Chapter 39, "Together Open API" for more information on working with modules.

Components tab

The Components tab enables you to access and reuse component models. It shows the `$TOGETHER_HOME$/modules/components` directory of your installation. This directory includes the "Coad Modeling Components," which are enterprise component models in color that you can reuse or modify. You can add your own components to the Components tab by placing them in directories under the components directory.

To display the Components tab with an open project:

1. Select **Project | Properties** from the main menu.
2. Check **Include Components** on.

You can copy classes and packages from the Components tab to your class diagrams, thus creating new source files and packages in your project. You can copy them to any package in your project as well. Classes and packages copied in this way appear on the Model tab in the appropriate package but do not appear in any class diagrams until you open the diagram and the source files are reverse engineered.

"Explorer pane operations" describes how to copy between Explorer tabs and diagrams.

Special Features tabs

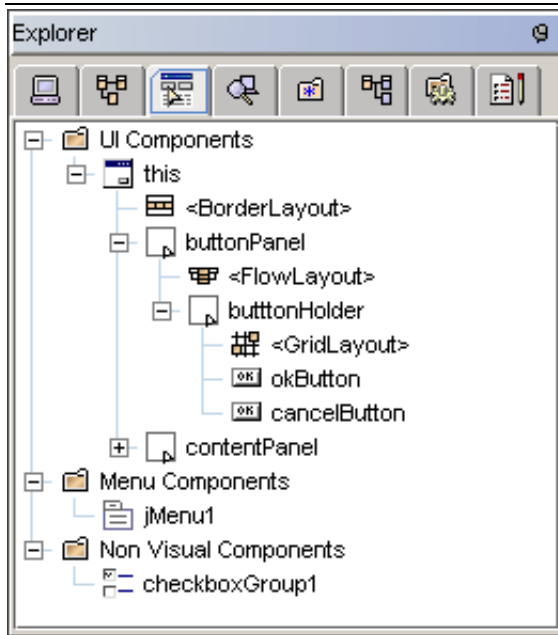
The Explorer displays an additional tab for some features that are activated.

User Interface Builder tab

The UI Builder tab shows a hierarchy of the user interface components in the Designer pane. It displays only when the Designer pane is in UI Designer view or Menu Designer view. (“Designer pane” on page 59 has a discussion of Designer pane views.)

Figure 8 shows the UI Builder tab for the sample project.

Figure 8 UI Builder tab



See Chapter 15, “Using the UI Builder” for a full discussion of how to use the UI builder.

Test tab

The Test tab is present only when a project is open and the Testing Framework feature is activated. It displays a list of unit tests, test suites, and test results.

See Chapter 23, “Using the Testing Framework” for a full discussion of how to use the Test tab and the Testing Framework.

XML tab

The XML tab is present only when a project is open, the XML Support feature is activated, and an XML file is open in the Editor. The XML tab is an “explorer” for the XML file. You can use it to examine the elements of the XML file as well as add new ones.

Chapter 13, “XML Modeling” offers a full discussion of how to use the XML tab and the XML editor.

Explorer pane operations

There are several operations that are common across two or more Explorer pane tabs.

Right-click menus

The right-click menus on various types of nodes on Explorer tabs enable actions such as opening source files, performing clipboard operations, and checking files into version control. Explorer nodes for classes, use cases, and other main diagram elements have the same right-click menus as the elements in the diagram. You can perform the same operation from either place with identical results.

Quick project access

When you create project directories under the myprojects directory in your Together installation, they display under the myprojects folder of the Directory tab. You can quickly navigate to projects and open them from there.

To customize the default location for new projects:

1. Select **Tools | Options | Default Level** from the main menu.
2. Expand **General** (the top item on the left).
3. Click on **New Project** without expanding it.
4. Enter the default location in the textfield on the right.
5. Click **OK** to close the dialog box and save the setting.

Copying and pasting

You can copy and paste classes, packages, diagrams, and elements of diagrams using the Explorer. Items that can be copied have a Copy command on their right-click menus. You can copy between the Component and Model tabs or from the Explorer to an open diagram.

To copy something from the Explorer:

1. Select the Explorer tab containing the source.
2. Select the items you want to copy.

NOTE: You can select several items at once by using Ctrl+click.

3. Choose **Copy** from the right-click menu.

4. Select **Paste** from the right-click menu of the destination, which can be a package in the Explorer or a diagram in the Designer.

It is also possible to clone diagrams, elements, and class members. Cloning creates an exact copy of the cloned item in the same package or node element and gives it a default name, which you can edit.

The instructions for cloning are identical to those for copy, except select **Clone** from the right-click menu.

NOTE: You can also clone node elements in the Designer pane using their right-click menus.

Finding the locations of nodes, files, and folders

On the Directory, Model, Modules, Diagrams, and Components tabs, you can navigate to the desired location without actually scrolling the tree-view. With the appropriate tab active, type the name of the desired folder. As you enter the characters, the highlight moves to the appropriate node of the tree-view.

There is no need to type in complete names of nodes. As soon as the required node is reached, type a delimiter (slash, backslash, or dot) to complete the name and expand the node; then continue typing the name of the required nested node. Keep in mind that the entry is case sensitive.

Designer pane

The Designer pane is visible only when there is an open project. The three views of the Designer correspond to the different kinds of design that it supports:

- **Diagram Design:** for creating UML and other kinds of diagrams that make up the project model.
- **UI Design:** for building graphical user interfaces of Java projects.
- **Menu Design:** for building menus in graphical user interfaces.

The Diagram Design view is always available when there is an open project. The UI Design and Menu Design views are available only when there is an open project and the UI Builder feature has been activated.

Opening and closing diagrams

You can open a diagram from the current project by using the Explorer pane. (See “Model tab” in the Explorer pane section.) You can also open a diagram from a source code file in the Editor by choosing **Select in Diagram** from the right-click menu.

To close a diagram:

- Right-click on the diagram background and select **Close**, or
- Enter **Ctrl+F4**, or
- Right click the tab and select **Close** or **Close All**.

The Designer shows a tab for each open diagram. The tab displays an icon for the UML diagram type and the name of the diagram. The *focus diagram* is the one at the front. To switch between open diagrams, click the tab of the one you want to view.

Designer pane toolbar

The Designer pane toolbar lies across the top of the pane. The toolbar enables you to:

- navigate between different diagrams
- create new diagrams
- manage the appearance of the diagrams
- create and arrange new graphical user interface components

The Designer toolbar changes when you switch from creating models to building user interfaces and menus.

TIP: You can hide the toolbar on the Designer or the Editor by right-clicking the title bar and selecting **Hide Toolbar**.

Table 6 Diagram toolbar icons

Icon	Name	Meaning
	New diagram.	Displays the Object Gallery dialog box. From there you can select the new diagram type, name, location, and description.
	Back	Returns to the diagram from which this was first reached.
	Forward	Returns to go to the diagram first reached from this one.
	Parent	Returns to parent diagram.
	Expandable compartments	Makes the compartments for class and interface nodes expandable.
	Diagram view management	Displays view management options for showing or hiding different types of diagram content.
	UI Designer	Goes to the UI Designer (a user interface component must be selected).
	Menu Designer	Goes to the Menu Designer (a menu component must be selected).
	Diagram Designer	Goes to the Diagram Designer.

Designer tools

When the Designer is in Diagram Design view, the tools for constructing diagram elements lie on a toolbar on the left side of the pane. You use the tools to draw elements on the Designer pane to represent the structural and behavioral elements and interactions of your model.

The Diagram Design tools vary according to the type of focus diagram. These tools include principal elements defined by the UML for the focus diagram type as well as Together-specific enhancements (such as *Class by Pattern* in the class diagram). Mouse-over tool tips identify the toolbar icons. *Note* and *Note Link* elements are common to all diagram types.

When the Designer is in UI Design or Menu Design views, it has a *toolbox* that contains the available user interface components for AWT and JFC Swing libraries.

NOTE: You can undock the Designer toolbox in the same way as undocking a pane or inspector.

Editor pane

Together's Editor rivals the best stand-alone editors with features like code-completion, bookmarks, symbol browsing, pane splitting, color and indentation schemes, and keyboard customization. The Editor is syntax-savvy, highlighting reserved words in the target programming language of the current project. It can detect syntax errors in Java code as well.

Table 7 shows the toolbar icons for commonly performed Editor tasks

Table 7 Editor toolbar icons

Icon	Meaning
	Expand all lines of code. (This is a toggle.)
	Code sense. (See Chapter 14, "Working with Code" for more information.)
	Advanced code sense.
	Parameters tool tip.
	Comment or uncomment selected lines.
	Surround selected lines with a control structure (if, for, try-catch, and so on).
	Override or implement method.
	Expand snippet.
	Browse symbol.
	Go to next declaration.
	Go to previous declaration.
	Jump to next snippet tag.
	Last change.
	Go to line.

Table 7 Editor toolbar icons (continued)

Icon	Meaning
	Super method.
	Run test. Available only when Testing Framework is activated.
	Context-sensitive help.

You can open multiple files in the Editor, placing each file in its own tabbed page. When Together first starts, the Editor displays a new untitled document. If you select a diagram element on the Designer pane that has source code, the source code file opens and replaces the untitled tab.

You can open any text-based file in the editor using File | Open on the main menu. Alternatively, you can navigate to the desired file in the Explorer and choose Edit from its right-click menu.

The Editor pane has a right-click menu with commonly-needed commands. The collection of commands depends on whether the Editor pane is used with or without an open project as well as which Together features have been activated.

From the Editor pane's right-click menu, you can configure the editor settings (Text Editor Options), control breakpoints and bookmarks, perform clipboard operations, refactor code, format the source code, and invoke external tools.

Chapter 14, "Working with Code" offers a full discussion of how to use the Editor.

Message pane

The Message pane displays pages that:

- Provide system information. The Messages page shows a queue of system messages. Use the right-click menu to save or copy messages, navigate to problem spots described by a message line, or clear the message queue.
- Enable you to perform tasks. These include debug code, navigate to problem spots, and others.
- Displays the results of operations. Operations include generating metrics, real-time simulations, and others.

Messages are organized into tabbed pages, with a separate tab for each type of task, operation, or source of information.

You can show or hide the Message pane by using the main toolbar or the equivalent keyboard shortcut, Ctrl+Alt+M. When the pane is hidden, an icon in the first cell of the status bar indicates the presence of messages in the message queue, as shown in Table 8.

Table 8 Status bar icons for the Message pane

Icon	Meaning
	Message pane is open.
	Message pane is closed, no new messages.

Table 8 Status bar icons for the Message pane

Icon	Meaning
	Message pane is closed, new messages in queue.

To remove a message from the queue in the Message pane:

1. If the Message pane is closed, open it.
2. Right click the message and select **Remove** (to remove a single message) or **Remove All** (to remove all messages).

The Message pane right-click menu also enables you to sort system messages (by time or type), filter them, and save them to a file.

To configure the maximum number of messages in the Message pane:

1. Select **Tools | Options | Default Level** from the main menu.
2. Click **General** at the top of the list, but do not expand it.
3. Enter the maximum number in the **Messages Maximum Count** field.

NOTE: You can also set the Message pane to open automatically when a new error is detected.

4. Click **OK** to close the dialog and make the changes.

Property Inspectors

Properties inspectors enable you to view and change the properties of many Together elements. Inspectors are organized into tabbed pages whose content depends on the type of element.

To display the properties of an element:

1. Select the element or diagram in the Diagram pane or in the Explorer.
2. Do one of the following:
 - Right-click and choose **Properties** from the right-click menu, *or*
 - Use a keyboard shortcut (Alt+Enter or Alt+double-click), *or*
 - Click the **Properties** icon on the main toolbar, *or*
 - Choose **Selection | Properties** on the main menu.

Overview of Inspectors

The properties that Inspectors enable you to change vary greatly according to the type of element. For example, with a class Inspector, you can:

- Edit the class properties including name, stereotype, and superclass.
- Change the appearance of the class node on the diagram.
- Create and navigate hyperlinks between diagrams or diagram elements and other diagrams, elements on other diagrams, and external files or URLs.

- Edit comments in source code; add comments to non source-generating diagram elements.
- Add Javadoc comments such as `@author` and `@version`.
- Edit requirements information.
- Edit the bean properties.

The property editors for the Inspector fields depend on the value types. Table 9 shows the icons that the Inspectors display for some kinds of fields.

Table 9 Inspector property editor icons

Icon	Name	Meaning
	(Asterisk)	Indicates a field that can take multiple values.
	File/Path Chooser	Opens a selection dialog box.
	Add/Remove Value	Opens a dialog box that displays the list of current property values and enables you to add or remove them.
	Edit	Enables you to add or modify textual values
	Select Color	Opens a dialog box where you can choose foreground and background colors. Used in view adjustment fields.

To cancel or delete a new entry:

- Press **Esc** to cancel entries in text fields if you have not yet pressed Enter to complete your changes.
- If a value is already entered, use the **Undo/Redo** buttons on the main toolbar, or **Ctrl+Z**. This is valid for text fields, comboboxes, and checkboxes.
- To delete a value from a text field or a combobox, select it in the Inspector and press **Delete**.

Inspector tabset

The composition of the Inspector's tabset changes depending on the selected element. This section describes the basic function of each tab of an ordinary class inspector, as shown in Figure 9.

Properties tab

The Properties tab shows the properties of the currently selected element in the focus diagram or the diagram itself. You can view and modify values of properties.

Some properties can have multiple values. Multiple values are comma-delimited. Fields that enable multiple values are marked with an asterisk.

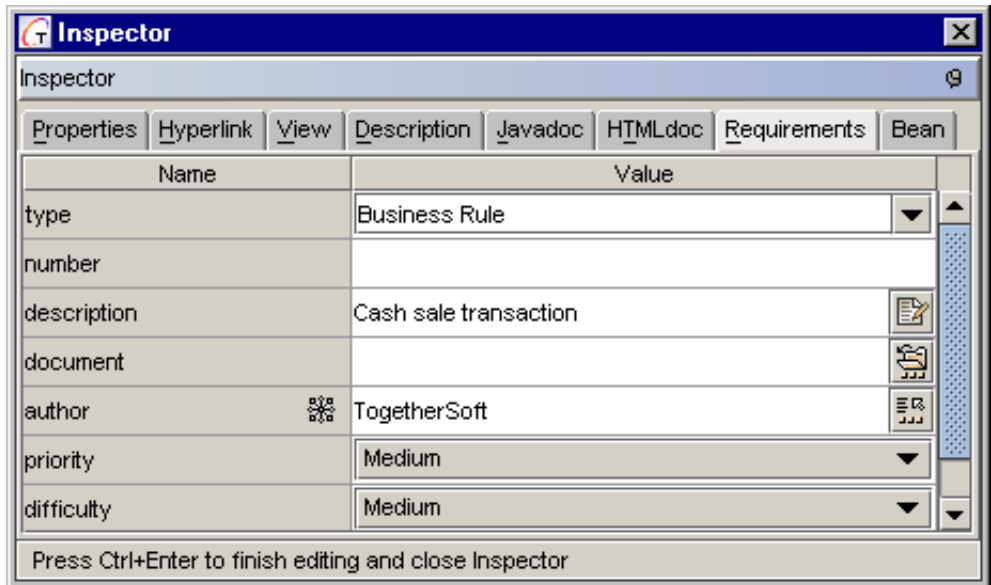
Hyperlink tab

The Hyperlink tab enables you to create hyperlinks to different types of artifacts and browse directly to them. You can create hyperlinks from the focus diagram or from a selected element on the diagram to:

- A diagram or diagram element anywhere in the project.
- A file that is external to your project.
- A URL on your company's intranet or on the Internet.

You create, view, remove, and browse hyperlinks with the Hyperlink tab's right-click menu.

Figure 9 Properties inspector for a class.



View tab

The View tab enables you to set the foreground and background colors of the selected element. When you start working with Together diagrams, the foreground and background colors are the default Windows colors.

To change the background color of a selected node:

1. Click the **Select Color** button in the background field.
2. In the resulting dialog box:
 - Choose the color from the standard Swing scale of colors using the Swatches tab, *or*
 - Define a custom color using the RGB (Red-Green-Blue) tab. In this case, you can choose the background color by moving sliders.
3. To restore the default color, click **Set Default**.

Description tab

For diagram elements that are on class or sequence diagrams, the Description tab displays source code comments. For example, if you select an interface, source code comments in the corresponding source file display on the Description tab where you can edit them.

For elements on other diagrams, you can enter comments for the element that are stored with the diagram file.

TIP: Avoid using ‘*/’ characters in the tags.

Javadoc tab

Inspectors for source-generating elements display the Javadoc tab. You can enter a description and specify values for Javadoc tags applicable to the selected element. These values are used when you generate Javadoc using the Documentation Generation feature.

Filling in the Javadoc fields automatically generates appropriate tags in the Javadoc tags in the source code. @author and @see tags allow multiple values. In this case a separate tag is generated for each value in the Inspector field. Checking the box @deprecated creates an empty tag.

Requirements tab

You can track various requirements properties including type, priority, and difficulty for diagrams and individual elements. You can specify a requirements document for the diagram or elements.

NOTE: A hyperlink to the document is not created when you specify the requirements document. Use the Hyperlinks tab to create such links.

Bean/C++ Properties tab

This tab is added to the Inspector when *Recognize Java Beans/C++ Properties* options are checked on. For classes, this tab displays JavaBean/EJB properties and events on the following lower tabs:

- General: general JavaBean attributes.
- Properties: bean properties such as getter, setter, bound, and constrained.
- Events: bean event sets.

To turn on *Recognize Java Beans/C++ Properties*:

1. Select **Tools | Options | Default Level** or **Tools | Options | Project Level** from the main menu.
2. Expand **View Management**.
3. Select **Java Beans/C++ Properties**.
4. Check the properties listed on the right side that are appropriate for your project.
5. Click **OK** to save and quit the dialog.

Workspaces

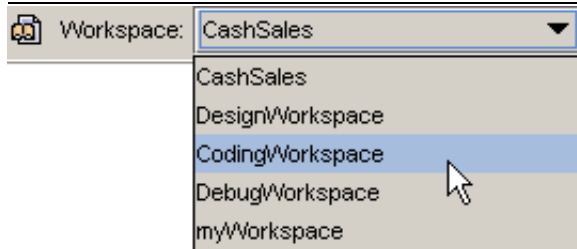
Together enables you to define, save, and recall different arrangements of main window panes. These saved, reusable pane arrangements are called *workspaces*.

Accessing different workspaces

Workspaces are project-specific. Every project has a *default workspace*, which Together creates based on the user's role. (See "Saving workspaces" below.) Together names the default workspace with the same name as the project.

You can access different workspaces from the rightmost item on the main toolbar. The dropdown list shows all of the workspaces available for that project under the current user role. See Figure 10.

Figure 10 Accessing workspaces from the main toolbar



Managing workspaces

You can create new workspaces, delete workspaces, and rename workspaces.

To create a new workspace:

1. Arrange the main window and panes for the new workspace layout.
2. Select **View | Workspaces | Create Workspace** from the main menu.
3. Enter a name for the new workspace in the resulting dialog box.
4. Click **OK** to quit the dialog and save the result.

To rename or delete an existing workspace:

1. Select **View | Workspaces | Manage Workspaces** from the main menu.
2. Select the workspace that you want to rename or delete from the list on the resulting dialog box.
3. To delete the workspace, click **Remove**.
To rename the workspace, enter the new name in the **Workspace name** field. Figure 11 illustrates changing the name of *myWorkspace* to *mySpecialWorkspace*.
4. Click **OK** to quit the dialog and complete the action.

NOTE: You cannot delete the default workspace. You can, however, rename it.

Saving workspaces

You can rearrange the panes and window layout for any workspace. Whether Together saves a rearrangement depends on the workspace save option.

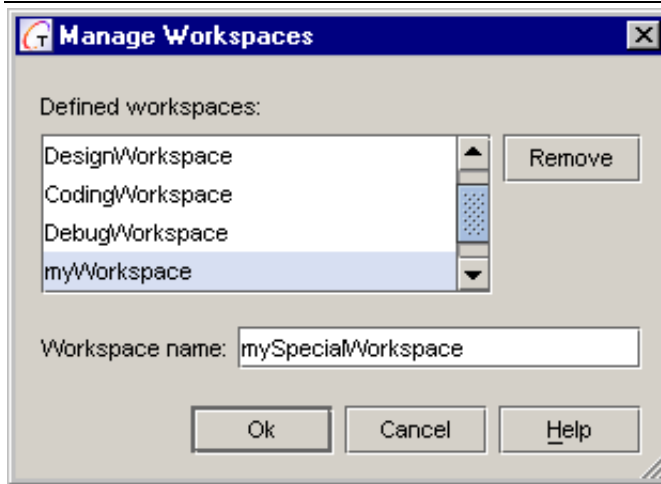
To set the save workspace option:

1. Select **Tools | Options | Default Level** from the main menu.
2. Select **General** from the option list, but do not expand it.
3. In the right frame, check **Save workspace before leaving**.
4. Click **OK** to close the dialog box and save the settings.

With the save option checked on, the current state of a workspace is automatically saved when you leave it. If the option is not checked on, any changes to the workspace are lost when you leave it.

If the save workspace option is turned off, you can still manually save the workshop. Select **View | Workspaces | Save Workspace** from the main menu before leaving.

Figure 11 Changing a workspace name



Role-based workspaces

A *role* is a pre-defined configuration of the user interface that helps you work in Together from a specific point of view. After you select a role, Together automatically provides ready access to only the relevant elements of the user interface, showing only the panes, toolbars, and menus that support the chosen role.

NOTE: Roles are available only in Together ControlCenter.

User roles

There are four different user roles:

1. *Business Modeler*: Designer central, Editor upon demand.
2. *Designer*: both Designer and Editor central. The user interface provides ready access to all operations up to the point of compilation.
3. *Developer*: both Designer and Editor central. The user interface provides ready access to all operations.
4. *Programmer*: Editor central. Provides ready access to all programming operations, including compile, debug, assemble, deploy, and run.

The Business Modeler role is intended for domain experts and analysts. Under this role, the Explorer shows the Directories, Model, Favorites, and Diagram tabs. The Directories tab shows only the Current Project, Samples, and Users projects. You must use the *New Project Expert* or the *Object Gallery* to create a new project. The Tools menu does not contain *Activate/Deactivate Features*; EJB buttons do not appear on the class diagram toolbar. Right-click menus are also streamlined for the business modeler's tasks.

The Designer role is intended for those who are designers or who are both analysts and designers.

The Developer role is intended for people who are involved in all phases of application development. This includes those who are analysts, designers, and programmers as well as those who are designers and programmers.

The Programmer role is intended for those who are engaged only in programming. In the Programmer role, Together opens with a closed Designer pane. It is still possible to open the Designer if required.

NOTE: You can create different workspaces to arrange the panes and main window layout within any role.

Changing the configured role

You must select a role during installation. However, it is possible to change roles at any time. The changes take effect the next time you restart Together.

To change your role:

1. Select **Tools | Options | Default Level** from the main menu.
2. Select **General** in the left frame, but do not expand it.
3. Select the role from the dropdown list in **Role after restart** on the right frame.

NOTE: The *Descriptions* field for *Role after restart* has complete descriptions of the default settings and detail levels.

4. Click **OK** to close the Default Options dialog box and save the change.

Advanced users can also modify the default role-based configurations by editing the `workspace.config` file.

Roles and view management

Together's view management feature allows you to control the type of data displayed in different views of a model. Different members of your project team can view aspects of the model that are relevant to their roles. For example, if you are the domain expert of a

project, you can model so that you do not see implementation details or other information irrelevant to your tasks.

You can access the view management feature from **Tools** on the main menu. Choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**). Alternatively, click the view management button on the horizontal Designer pane toolbar. The available options include:

- Level of detail: Default, Analysis, Design, or Implementation. The default setting is assigned according to the current role.
- Member format: UML or Language.
- Show subpackage contents.
- Recognize JavaBeans and Recognize C++ Properties:

NOTE: This option appears for the Default Level and Project Level only. It does not appear under the Diagram Level options.

If the option *Recognize JavaBeans* is checked, the Bean tab appears on the Properties Inspector of the classes. There you can add bean properties, getters and setters, and event sets. The Show attributes and accessors option controls whether bean properties and events show up on the class icon.

- Show or hide referenced classes in a diagram.
- Show dependencies between classes and interfaces.

NOTE: Recognizing dependencies can result in slow performance and large diagrams.

- Controlling the display of sequence diagrams.
- Ban destinations in class diagrams: to keep the diagrams less crowded.
- Show or hide aggregations of diagram and EJB elements.

SETTING YOUR PERSONAL PREFERENCES

This chapter explains how to set-up preferences for workspaces and view management. This chapter includes the following topics:

- “Setting-up workspaces” on page 71
- “Using view management” on page 74

Setting-up workspaces

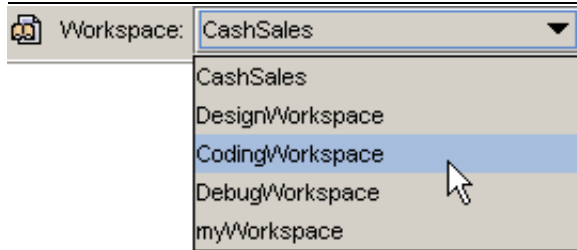
Together enables you to define, save, and recall different arrangements of main window panes. These saved, reusable pane arrangements are called *workspaces*.

Accessing different workspaces

Workspaces are project-specific. Every project has a *default workspace*, which Together creates based on the user’s role. (See “Saving workspaces” below.) Together names the default workspace with the same name as the project.

You can access different workspaces from rightmost item on the main toolbar. The dropdown list shows all of the workspaces available for that project under the current user role. See Figure 12.

Figure 12 Accessing workspaces from the main toolbar



Managing workspaces

You can create new workspaces, delete workspaces, and rename workspaces.

To create a new workspace:

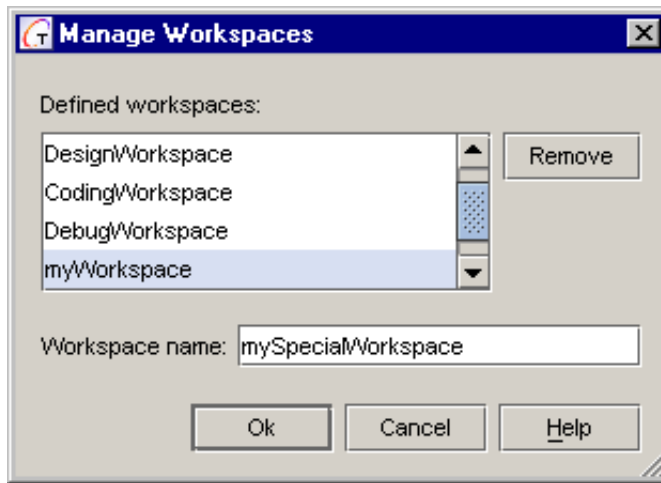
1. Arrange the main window and panes for the new workspace layout.
2. Select **View | Workspaces | Create Workspace** from the main menu.
3. Enter a name for the new workspace in the resulting dialog box.
4. Click **OK** to quit the dialog and save the result.

To rename or delete an existing workspace:

1. Select **View | Workspaces | Manage Workspaces** from the main menu.
2. Select the workspace that you want to rename or delete from the list on the resulting dialog box.
3. To delete the workspace, click **Remove**.
To rename the workspace, enter the new name in the **Workspace name** field. Figure 13 illustrates changing the name of *myWorkspace* to *mySpecialWorkspace*.
4. Click **OK** to quite the dialog and complete the action.

NOTE: You cannot delete the default workspace. You can, however, rename it.

Figure 13 Changing a workspace name



Saving workspaces

You can rearrange the panes and window layout for any workspace. Whether Together saves a rearrangement depends on the workspace save option.

To set the save workspace option:

1. Select **Tools | Options | Default Level** from the main menu.
2. Select **General** from the option list, but do not expand it.
3. In the right frame, check **Save workspace before leaving**.
4. Click **OK** to close the dialog box and save the settings.

With the save option checked on, the current state of a workspace is automatically saved when you leave it. If the option is not checked on, any changes to the workspace are lost when you leave it.

If the save workspace option is turned off, you can still manually save the workshop. Select **View | Workspaces | Save Workspace** from the main menu before leaving.

Role-based workspaces

A *role* is a pre-defined configuration of the user interface that helps you work in Together from a specific point of view. After you select a role, Together automatically provides ready access to only the relevant elements of the user interface, showing only the panes, toolbars, and menus that support the chosen role.

NOTE: Roles are available only in Together ControlCenter.

User roles

There are four different user roles:

1. *Business Modeler*: Designer central, Editor upon demand. The Business Modeler role is intended for domain experts and analysts.
Under the Business Modeler role, the Explorer shows the Directories, Model, Favorites, and Diagram tab. The Directories tab shows only the Current Project, Samples, and Users projects. You must use the *New Project Expert* or the *Object Gallery* to create a new project. *Activate/Deactivate Features* is removed from the Tools menu. EJB buttons do not appear on the class diagram toolbar. Right-click menus are also streamlined for the business modeler's tasks: source formatting, make/rebuild QA features, and the Choose Pattern command for members and attributes are all removed.
2. *Designer*: both Designer and Editor central. The user interface provides ready access to all operations up to the point of compilation. The Designer role is intended for those who are designers or who are both analysts and designers.
3. *Developer*: both Designer and Editor central. The user interface provides ready access to all operations. The Developer role is intended for people who are involved in all phases of application development, those who are analysts, designers, and programmers as well as those who are designers and programmers.
4. *Programmer*: Editor central. Provides ready access to all programming operations, including compile, debug, assemble, deploy, and run. In the Programmer role, Together opens with a closed Designer pane. It is still possible to open the Designer if required. The Programmer role is intended for those who are engaged only in programming.

NOTE: You can create different workspaces to arrange the panes and main window layout within any role.

Changing the configured role

You select a role during installation. However, it is possible to change roles at any time. The changes take effect the next time you restart Together.

To change your role:

1. Select **Tools | Options | Default Level** from the main menu.
2. Select **General** in the left frame, but do not expand it.
3. Select the role from the dropdown list in **Role after restart** on the right frame.

NOTE: The *Descriptions* field for *Role after restart* has complete descriptions of the default settings and detail levels.

4. Click **OK** to close the Default Options dialog box and save the change.

Advanced users can also modify the default role-based configurations by editing the `workspace.config` file.

Using view management

Together's view management feature allows you to control the type of data displayed in different views of a model. Thus, different members of a project team can view aspects of the model that are relevant to their roles. For example, if you are the domain expert of

a project, you can set-up a model so that you do not see implementation details, or other information irrelevant to your tasks.

Controlling the level of detail shown in diagrams

To control the desired level of detail within a diagram, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. Select the **View Management** node from the tree
3. For *Diagram detail level*, select **Default**, **Analysis**, **Design**, or **Implementation**. The default setting is assigned according to the current role.

Controlling how members display in diagrams

To control whether members of a class display in UML format or a corresponding language, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. Select the **View Management** node from the tree, to see the names and values of its options.
3. For *Member format*, select **UML** or **Language**.

Showing and hiding subpackage contents in diagrams

Check this option to display subpackages' contents as package icons with the lists of classes, interfaces, and underlying subpackages (Options - View Management - Show subpackage contents).

To show or hide subpackage contents, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. Expand the **View Management** node from the tree, and select **Show subpackage contents**
3. Check the box for *Show subpackage contents* to show contents. Uncheck the box to hide contents.

Controlling how Java Beans/C++ properties display in class diagrams

To control how Java Beans/C++ properties display in class diagrams, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (or **Project Level**)
2. From the tree, select **View Management | JavaBeans / C++ Properties**

3. Check the boxes for *Recognize JavaBeans* and/or *Recognize C++ Properties*

NOTE: This option appears for the default and project levels only. It does not appear under the Diagram Level options.

If the option *Recognize JavaBeans* is checked, the Bean tab appears on the Object Inspector of the classes, where you can add bean properties, getters and setters, and event sets. The Show attributes and accessors option controls whether bean properties and events show up on the class icon.

Showing and hiding referenced classes in diagrams

To show or hide referenced classes in a diagram, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. Select the **View Management** node from the tree
3. For *Referenced classes*, select **Show name** or **Hide**.

Showing dependencies between classes and interfaces

To show dependencies between classes and interfaces within a diagram, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. From the tree, select **View Management | Dependencies**
3. For *Check*, select **Declarations** only or **All usages**

NOTE: Recognizing dependencies can result in slow performance and large diagrams.

Controlling the display of sequence diagrams

To access the sequence diagram options for view management, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. From the tree, select the **Sequence diagram** node; You can set options for this node, or you can expand the Sequence diagram node to view more options

Banning destinations in class diagrams

You can use the Banned Destinations option to keep association links under control. If the links to standard classes are not filtered from view, your diagrams can become crowded with links that impede comprehension. Some standard Java classes are banned by default.

To ban destinations in class diagrams, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (or **Project Level**)
2. From the tree, select **Banned Destinations** and enter the class that you want to ban.

In addition to the fields provided in the options dialog, you can add more fields, by editing the `viewManagement.config` file. For more information, refer to the online Description field that appears for banned destinations in the Options dialog.

Showing or hiding aggregations of diagram and EJB elements

The view management Show and Hide options allow you to control the presentation of information in diagrams. Together comes with a number of predefined view management options and provides additional ones that you can custom-define.

To access show and hide options for view management, follow these steps:

1. From the **Tools** menu, choose **Options | Default Level** (you can also choose **Project Level** or **Diagram Level**)
2. From the tree, expand **View Management**, and then expand **Show** (or **Hide**)

CONFIGURING OPTIONS

This chapter explains how to configure options for *your* installation of Together. The Together administrator can configure options on a more global level, impacting all users. To learn how to configure Together for multiple users, refer to the *Installation and Administrator's Guide*.

This chapter includes the following topics:

- “Activating and deactivating modules” on page 78
- “Overview of configuration levels” on page 79
- “Setting options in default mode” on page 79
- “Setting options in advanced mode” on page 80
- “Viewing and editing options” on page 82
- “Reference guide to options” on page 83
- “Common Configuration Tasks” on page 89

Activating and deactivating modules

Some Together features such as the GUI builder and testing framework need to be activated before you can use them. This also applies to most integrations for third-party products such as Versant and Persistence Powertier.

NOTE: By activating a module, you can also view options specific to the module. Similarly, deactivating a module removes the options specific to the module from the user interface.

To activate (or deactivate) a module, follow these steps:

1. From the Tools menu, choose Activate/Deactivate Features
2. Select the **Together Features** tab (or the **Integrations** tab)

3. Check the modules that you want to activate; uncheck the modules that you do not need to use. For your convenience, each module's path name and size is provided.
4. Click **OK**

Depending on the number of modules you have activated, it may take several seconds for Together to refresh the user interface.

Overview of configuration levels

Set options for each of the three levels of Together's multilevel configuration:

- **Default level:** Settings apply to the entire **Together** installation, unless overridden at the project or diagram levels
- **Project level:** Settings apply only when a project is open, unless overridden at the diagram level
- **Diagram level:** Settings apply to a specific diagram (specifically, the diagram that is open at the time the option is set)

For any of the three levels, set options using a dialog in either *default* or *advanced* mode. Default mode allows you to set options for a single configuration level.

Advanced mode allows you to set options at multiple levels without re-invoking the dialog. For more information see, "Setting options in advanced mode" on page 80.

Together allows you to apply settings for most of the options at any level.

Consequently, options cannot be overridden at a lower (that is, more local) level.

When Together is installed as a server-based application, options are centralized and shared by all users. The Together administrator can customize the configuration so that specific settings apply across the enterprise (Default level marked final), or globally for a team (Project level marked final). To learn how to configure Together for a shared installation used by multiple users, refer to the *Installation and Administrator's Guide*.

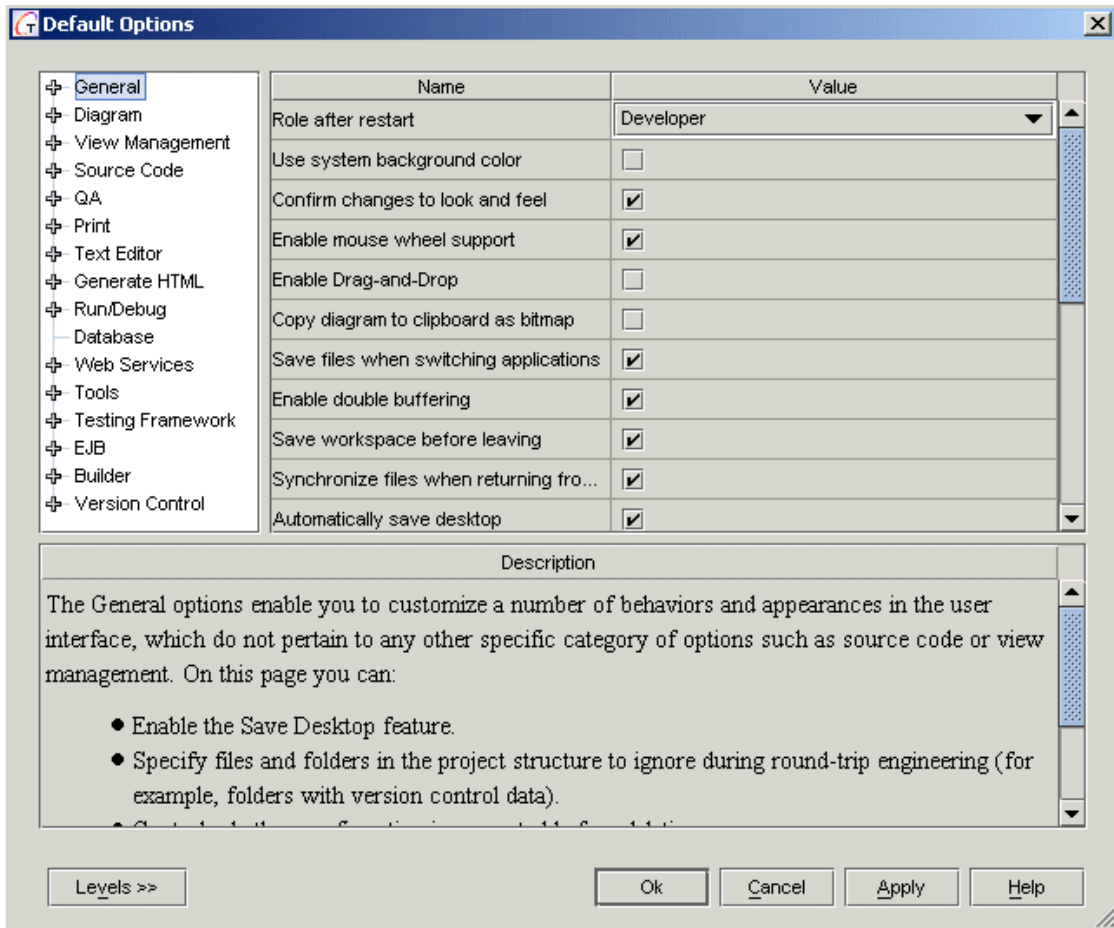
Setting options in default mode

Use the options dialog in default mode to set options at a specific default, project, or diagram level. (In contrast, advanced mode allows you to set options at more than one level.)

To set options in default mode, follow these steps:

1. From the Tools menu, select **Options**
2. Select **Default Level**, **Project Level**, or **Diagram Level**. The dialog for the respective level appears. For example, Figure 14 shows the Default Options dialog.
3. Set the options as needed. For more information, see "Viewing and editing options" on page 82. For a reference to the options, see "The Options dialog box can be resized. To resize the width of the Name and Value columns drag the separator between the column headings. In addition, the size of the Description area can be changed." on page 83.
4. Click **OK** to save settings.

Figure 14 Default Options dialog in default mode



Setting options in advanced mode

Use the options dialog in advanced mode to set options at multiple levels without re-invoking the dialog. The options dialog for each level (default, project, diagram) includes a *Levels* button that you use to toggle between default and advanced modes.

In contrast to default mode, advanced mode includes the columns *Level* and *Final*, as shown in Figure 15. The *Level* column allows you to select the configuration level. In this case, *Level* indicates the level at which each configuration option is currently set, assuming the default level definitions:

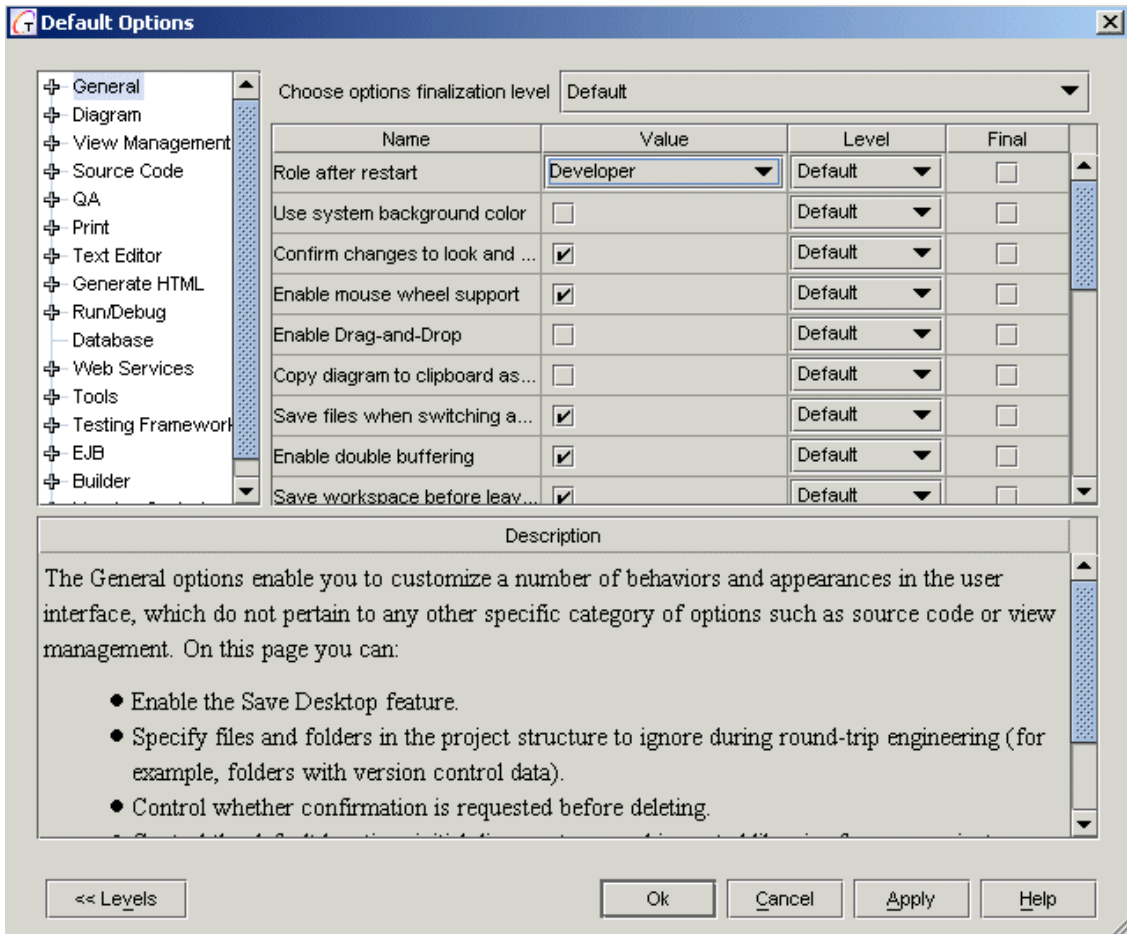
Default = Option is set at the Default level

Project = Option is set at the Project level

Diagram = Option is set at the Diagram level

The Final column indicates that an option cannot be overridden at a level more local than the one shown in the Level column. Change any settings that are not marked final at a “higher” (that is, more global) level. In a local installation, you typically have complete control over your configuration and the settings at all levels. In a shared installation, change only those settings not marked as final by the system administrator in the shared configuration.

Figure 15 The Default Options dialog in advanced mode



To set options in advanced mode, follow these steps:

1. Open a project. To set options for a specific diagram, select the diagram.
2. Open the Options dialog for the lowest available level by choosing **Tools | Options | *available_level***.
3. Click the **Levels>>** button to enter advanced mode.

4. Set options for each level. For more information, see “Viewing and editing options” on page 82. For a reference to the options, see “The Options dialog box can be resized. To resize the width of the Name and Value columns drag the separator between the column headings. In addition, the size of the Description area can be changed.” on page 83.
5. Click **Apply** to save changes as you work.
6. Click **OK** to save changes and close the dialog.

Viewing and editing options

The information in this section assumes that you have accessed the options dialog. For more information, see “Setting options in default mode” on page 79 or “Setting options in advanced mode” on page 80.

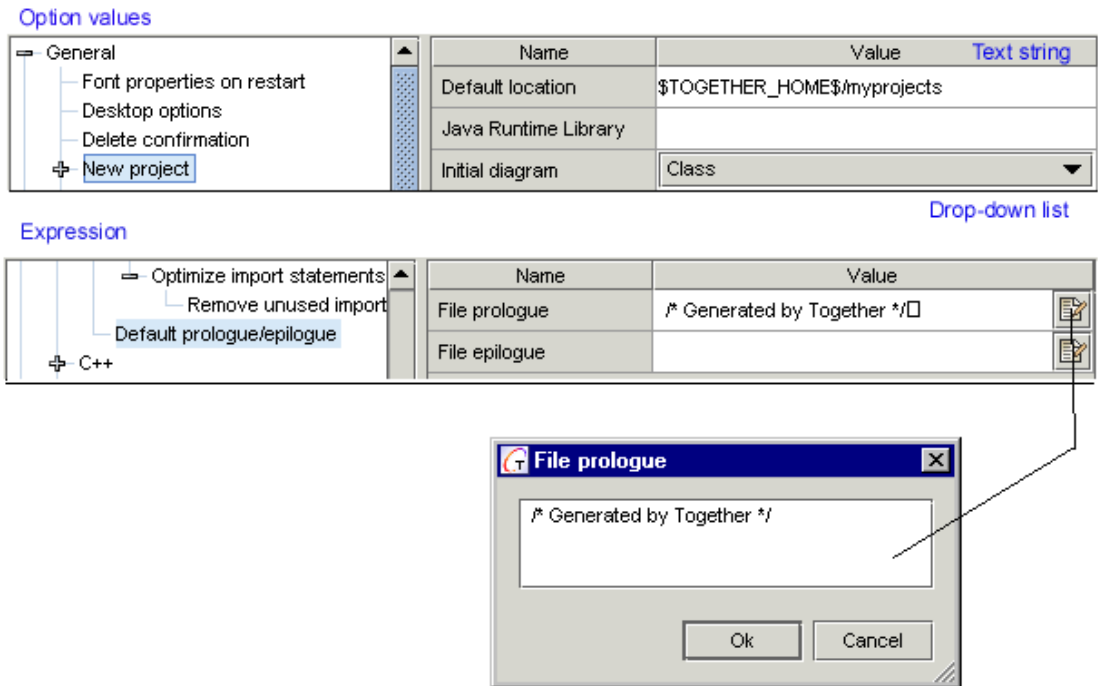
The Options dialog logically groups the options as nodes in a tree. When an option consists of groups of sub-options, you can expand the node to show all available sub-options.

Often options can be set for a feature at *each* hierarchical level of the tree view. Select a node, regardless of its hierarchical level, to view the values that can be set. For example, while the following nodes appear in different hierarchal levels of the tree, each node provides values that can be set for a sequence diagram:

- View Management
- Sequence diagram
- Generate Sequence Diagram
- Generate Source Code Options

Options have editors depending on the type of value. If an option has multiple values, the values are comma-delimited. Figure 16 illustrates these concepts.

Figure 16 Examples of values for options



Values indicated by checkboxes

Together uses checkboxes to specify a Boolean value for an option. A checked box means “true” or “yes”, and an unchecked box means “false” or “no”.

Accessing context-sensitive help for nodes and options

Help for each node and option displays directly in the dialog under Description. To see a general description of a group of options, click on its node. To see the description of an individual option, click its name.

Resizing the Options dialog

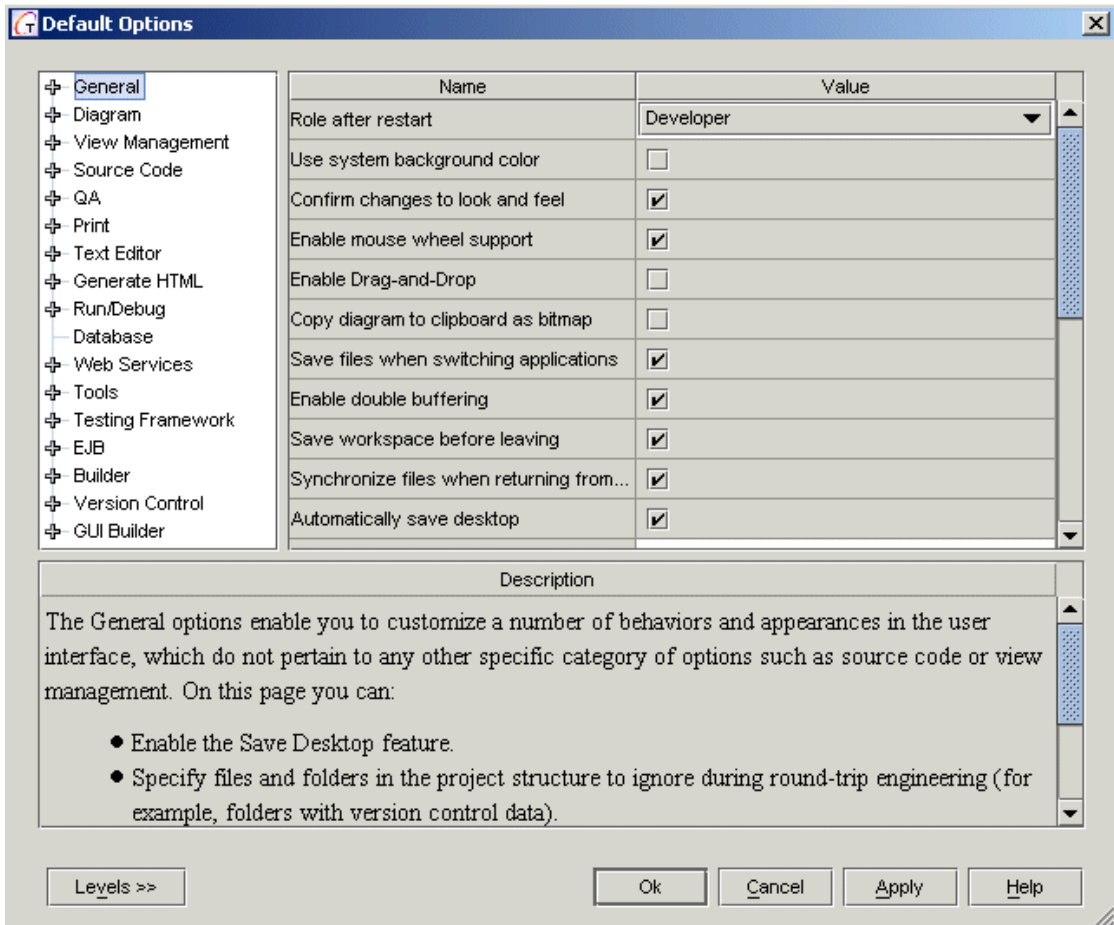
The Options dialog box can be resized. To resize the width of the Name and Value columns drag the separator between the column headings. In addition, the size of the Description area can be changed.

Reference guide to options

This section summarizes the options that can be configured in Together. Options are listed and grouped within the tree of the Default Options dialog (as well as the Project Options

and Diagram Options dialogs), as shown in Figure 17. Help text for the nodes and the individual options appears in the Description field of the options dialog.

Figure 17 Tree listing options and option groups



Keep in mind that options can be set at any one of three [pre-defined configuration levels](#). Know what level you are working on before changing any option settings.

Default and project level options

Table 10 lists the options that you can set at the default and project levels. The table lists the options in the order they appear in the Default Options dialog.

NOTE: If the user interface does not show the options listed for a specific feature, make sure that the feature is activated. Some features such as the testing

framework and GUI builder must be activated before using them. See “Activating and deactivating modules” on page 78 for more information.

Table 10 Default and project level options

Option	Allows you to...	On this node you can...
General	Customize a number of behaviors and appearances in the user interface.	• Control background color and font properties. • Define your workspace role. • Set to auto-synchronize files when returning from other applications (such as an external editor or IDE). • Enable or disable the Saved Desktop feature that remembers your desktop settings between sessions. • Specify files and folders in the project structure to ignore during round-trip engineering (for example, folders with version control data). • Control the display of delete confirmations. • Control the default location, initial diagram type, and referenced libraries for new projects. • Control whether the Message pane opens on errors. • Enable or disable email exception reports to the Together development team. • Control whether Together will search archives for diagram files.
Diagram	Control a number of default behaviors and display properties that apply only to diagrams.	• Control the default link routing method. • Control the alignment, layout, justification, and initial maximum width of classes in diagrams. • Change the font used in diagrams. • Specify how association links are drawn and how and whether they are represented in classes (and therefore, indirectly, in source code). • Specify whether to generate metafile images of diagrams when saving diagrams. • Control the display of the background grid in the Diagram pane.

Table 10 Default and project level options (continued)

View Management	Control what you want to see and when. Specify how different kinds of elements display in the diagrams, or even whether they show up at all. For more information, see “Using view management” on page 74.	• Control the general level of detail shown in diagrams. • Control whether members in classes display in UML or Java format. • Show or hide subpackage contents in diagrams. • Control how JavaBean classes and C++ properties display in Class diagrams. • Show or hide referenced classes in diagrams. • Control the display of dependencies. • Show object class names and message numbers in Sequence diagrams. • Control the display of messages in Sequence diagrams. • Set banned destinations. • Specify the maximum call stack depth for Sequence diagrams generated from operations. • Show or hide aggregations of diagram elements and EJB elements.
Source Code	Control a number of default behaviors and appearances that apply to the formatting of source code during forward and reverse engineering operations.	• Specify the relative position of Attribute declarations and Operation declarations within Class declarations (Attributes first or Operations first). • Control how link attributes are handled when the destination is deleted. • Specify when code should be reformatted. • Specify the type of line separator for your OS. • Specify language-specific code formatting, optimizing, name referencing, and importing statement options. • Specify version-specific IDL formatting options such as indenting and format of comments. • Specify exactly how source code and comments are formatted (in-line breaks, space preservation, separators, and so on). Includes Javadoc comment formatting options for comments. • Customize source file prologue and epilogue text (the “Generated by” text at the head of source code files) for Java, C++, and IDL.
QA	Customize QA options. Note: To view this option, you need to activate the QA module.	• Specify the path to saved sets of audits and metrics. • Define the applicable scope of Quality Assurance.

Table 10 Default and project level options (continued)

Print	Set a number of defaults that apply to printing diagrams, files, and generated documentation.	• Set default paper size or define a custom one (for example, for printing on a plotter). • Set page orientation and margin sizes. • Set a number of other print options such as print zoom level, page border/footer, and so on.
Text Editor	In the Text Editor options, you can control a number of default behaviors and appearances that apply to the display of text in the Editor pane. Note: Source code formatting is not customized on this node. Use the Source Code options.	• Define the number of spaces inserted in text when you use the Tab key. • Define the font size. • Define the text color and style for code comments. • Define the text style for programming language reserved words. • Define the orientation of the cursor. • Customize the keyboard shortcuts for the editor. • Customize the way the editor works with specific kinds of files in a number of programming languages.
Generate HTML	Control the inclusion/exclusion of various content in the output of the standard HTML documentation generation facility (Tools Generate HTML).	• Include or exclude author and version tags in generated output. • Specify all Javadoc settings. • Specify which visibility levels of classes to include in generated HTML output. Tip: Set these options as you begin the documentation generation process. From the Generate HTML dialog, click Options to display the Options dialog with only the Generate HTML node visible. If you change any settings, they are used for the current documentation generation operation only and then discarded. Option settings for your configuration are not changed.
Run/Debug	Customize your runner/debugger.	• Define the root directory of the JDK for running and debugging applications from Together. NOTE: The default compiler is the Sun SDK, which is specified in the field JDK Home. By default it points to \$TGH\$/jdk. If the JDK is not a part of the Together installation, you should specify the correct folder. Otherwise, compile/make/run commands will not work. • Define the location of the sources and current working directory. • Specify the project's run configurations. • Specify settings for the debugger. Specify settings for JSP.

Table 10 Default and project level options (continued)

Database	Customize a number of common database properties that are used in operations with databases.	• Define the amount of time that Together will wait for a connection with DBMS. • Control whether Together replaces tables after generating a DDL. • Specify whether to use quoting symbols for identifiers in the resulting DDL.
Web Services	Set the application server that is used by default to register web services. Together can work in conjunction with other file-based development tools, such as compilers, debuggers, IDE's, and editors. The Tools node contains the external tool definitions for your configuration. The External Editor tool definitions are already pre-defined for you. Shell definitions for other tools are included, which you can use to set up interaction with an IDE or other tools. You can edit the fields to point these definitions to appropriate tools on your system. Menu commands for launching or interacting with the various tools are displayed on appropriate menus in the menu system. You can specify which menus should display the tool command in the Show in menu subnode. If you find you need more tool definitions beyond the user-defined, you can add more options to the Tools node. This is an advanced customization that involves editing the <code>tool.config</code> file.	
Tools	Edit various fields to point to appropriate file-based development tools (such as compilers, debuggers, IDEs, and editors). The Tools options contain the tool definitions for your configuration. The External editor tool is pre-defined for you.	
Testing Framework	Customize testing framework options. Note: To view this option, you need to activate the Testing Framework module.	
EJB	Set J2EE defaults	• Select the application server for J2EE deployment, which also determines which EJB specification new EJBs will be designed to meet. • Edit EJB suffixes that are used for EJB recognition.

Table 10 Default and project level options (continued)

Builder	Set options for the Builder.	• Choose to performing compilation prior to run/debug. • Reflect the compiling process in the Status bar. • Control the format of compiler output. • Choose the target folder for the generated makefile. • Set compiler options. • Specify the maximum permissible number of compilation errors.
Version Control	Enable and set up version control integration. By default, everything is set up to integrate with CVS (which is installed with Together). If you use an SCC-compliant version control system, choose SCC in the Use option. Note: To use a SCC version control system, Together must be running under Windows (NT/98/95) and Coroutine classes must be installed (default auto-installed and configured by the Together installer for Windows).	• Enable or disable version control integration. • Specify default interactions such as automatically getting files from version control when opening a project. • Specify what type of version control integration to use (CVS or SCC). • Set properties for CVS LAN and CVS Client-Server.

Diagram level options

Set the following options at the diagram level:

- Diagram
- View Management
- Print

These options can also be set at the default and project levels. See Table 10 for a description of each.

In addition, when activated set diagram level options for the following:

- IDL
- JProbe
- Powertier

For more information, see “Activating and deactivating modules” on page 78.

Common Configuration Tasks

This section groups typical configuration tasks.

How to make Association links display a directional arrow

Configure the default properties of Association links in the Options dialog, and the properties of individual links in the properties Inspector of the link.

To configure default Association properties, follow these steps:

1. Choose **Tools | Options | Default Level** to launch the Options dialog.
2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.
3. From the tree-view, expand **Diagram** and select **Associations**.
4. Set the value of *Draw directed* as required:
 - Automatic - Links represented by the attributes whose names start with “link,” will be shown directed. All other links will be shown undirected.
 - All - All links will be shown directed.
 - None - All links will be shown undirected.
5. Click **Apply** and then **Ok**.

This option applies only to Association links whose “directed” property in the link’s Inspector is set to “Automatic”. If it is set to “Directed” or “Undirected,” the link will always be displayed according to that setting, regardless of the value set for this option.

TIP: You can also set *Show as attributes* so that attributes which are displayed as links show in the attributes section of classes.

How to change the default source file header for the generated code

1. Choose **Tools | Options | Default Level** to launch the Options dialog.
2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.
3. From the tree-view, expand **Source Code** and then expand the node of the appropriate language (these instructions apply to Java, C++, or CORBA IDL).
4. Choose *Default prologue/epilogue*.
5. In the Value column, edit the default text as required. For example: “Generated for XYZ. Corp. Copyright (c)2001. Company confidential.”
6. Click **Apply** and then **Ok**.

How to customize the default settings for C++

For information on how to customize the default definitions for C++ source and header files, and configure default library support, see “Using Together with C++” on page 698.

How to create and use custom snippets for source code and text

This features can help you minimize misspelled words and other syntax errors in your source code. See “Snippets” on page 241 for more information.

How to configure Stereotypes

Customizing stereotypes is a low-level configuration task and requires Java programming. This enables you to populate stereotype lists, modify default stereotype values, and specify RGB color values for stereotypes, and so on. See “How to customize Inspector properties” on page 91 for more information.

How to customize Inspector properties

For more information on how to modify the property names and/or default values in properties Inspectors, add your own properties, or delete properties, see “How to customize Inspector properties” on page 91. Customizing inspector properties is a low-level configuration task and requires Java programming.

How to hide and show elements

1. Choose **Tools | Options | *Applicable_Level*** to launch the Options dialog.
2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.
3. From the tree-view, expand **View Management**.
4. Expand **Show**. To hide the elements defined by one of the options (Inheritance links, for example), clear the option checkbox. To re-show elided elements, check the option box.
5. Click **Apply** and then **Ok**.

See “Using view management” on page 74 for more information.

Note that individual elements can be hidden in diagrams using the Diagram right-click menu. If you do not see an element in a diagram, and the element is not hidden by View Management options, choose Show Hidden from the Diagram right-click menu and check the hidden elements list.

Define filtering expressions for the User-Defined options under the Show options of the View Management page. Study the expressions in the pre-defined Show options to learn how to show or hide different elements.

The other options on the View Management page of the Options dialog may hide some kinds of information. For example, *Diagram Detail Level* can hide visibility symbols.

How to set options to control the formatting of your source code

1. Choose **Tools | Options | *Applicable_Level*** to launch the Options dialog.
2. If necessary, click the Levels button to enter **Advanced** mode to set options at multiple configuration levels.
3. From the tree-view, expand **Source Code** and then expand the node of the appropriate language.
4. Expand **Formatting options** and set sub-options as desired.
5. Click **Apply** and then **Ok**.

How to enable mouse-wheel support (Windows only)

Java VM 1.2, 1.3 does not support mouse wheel events. To scroll a diagram or a frame when using a mouse equipped with a wheel, Together usually transforms mouse wheel events into Ctrl-Up, Ctrl-Down, Up or Down keystrokes.

To enable mouse-wheel support:

1. On the main menu, choose **Tools | Options | Applicable_Level**.
2. From the tree-view, choose **General**.
3. Check or uncheck *Enable mouse wheel support*.

If the option is checked, this feature is enabled. If unchecked, Together does not transform mouse wheel events.

4. Click **Apply** and then **Ok**.

NOTE: This option applies only when Together is running with the Sun Java Virtual Machine (JVM) under Windows. Unlike most options, this one requires restart of Together before taking effect.

How to set up Together and projects to interact with version control

Together comes pre-configured for the CVS version control system (automatically installed) and version control support is enabled in the system by default. If you already use a SCC-compliant version control, you can change your configuration to use that system.

Version control is not automatically enabled for projects. You need to enable it and specify the version control system project to use when you create the Together project (or later in Project Properties).

To enable or disable version control integration support, follow these steps:

1. Choose **Tools | Options | Applicable_Level** (Default or Project).
2. If necessary, click the Levels button to enter **Advanced** mode to set options at multiple configuration levels.
3. Select **Version Control** from the tree-view.
4. Check or uncheck *Version Control enabled*.

To enable version control for a project, see “Setting-up version control for projects” on page 104.

PART

2

CREATING AND MANAGING PROJECTS

- Chapter 4, “Working with Projects”
- Chapter 5, “Reverse Engineering”
- Chapter 6, “Importing and Exporting Information”
- Chapter 7, “Importing Database Information”

WORKING WITH PROJECTS

Almost all the work in Together is performed in the context of projects. This chapter explains how to create and edit own projects in Together. It includes the following topics:

- “Understanding Project Basics” on page 94
- “Creating new projects” on page 95
- “Editing project properties” on page 99
- “Running multiple project instances” on page 104
- “Setting-up version control for projects” on page 104
- “Setting up large projects” on page 105

TIP: Together ships with a collection of sample projects. If you are new to Together and want an introduction to projects, see “Getting started with a sample project” on page 42.

Understanding Project Basics

To begin modeling with Together, you need to create a project.

At minimum, a project consists of the following:

- Primary root directory
- Project file
- Default package diagram

Primary root directory

The primary root directory stores the project file and any initial diagrams created with the project. It also serves as a repository for project-level properties files. (Project level

is one of the pre-defined configuration levels in Together as described in “Overview of configuration levels” on page 79).

When creating a project, select a new or existing directory for the primary root directory. By default, Together round-trip engineers any source code it finds in the primary root directory and any subdirectories below it.

IMPORTANT: Specifying the top-level directory of a particularly large code base as the primary root directory can slow the performance of reverse engineering. To avoid this, redefine the parsing of project resources. For more information, see “Performance tuning” on page 105.

The scope of a project is not limited to a single root directory. Instead, specify multiple directories as project root directories, include or exclude subdirectories of any root, and exert some initial control over how these directories are treated during round-trip engineering.

Project file

The Explorer pane, uses the Together icon  to indicate a project file. The file name of a project has the .tpr extension.

The <default> diagram

When creating a new project, Together generates a diagram that illustrates the physical project content contained in the primary root package. The generated diagram is named `default` and displays the default diagram icon. The name of the underlying diagram file is `default.dfPackage`. The `default` diagram shows Package icons representing subdirectories of the primary root directory as well as the classes (and other elements) of any source code files found in the primary root.

When creating a new project, opt to specify an initial diagram that Together creates along with the project. Class diagram is the default type. If your license supports creation of other UML diagrams, you can specify another type of diagram as the initial diagram, or *none* for no initial diagram. Together always generates the `default` diagram.

Note that there is no default diagram that opens every time you open a project. Any diagrams that open along with a project are controlled by the configuration settings under Desktop Options (see Chapter 3, “Configuring Options”). If properly set, Together remembers which diagrams were open when the project closed and reopens the diagrams the next time you open the project.

Creating new projects

Together provides two ways for creating projects. Use the New Project Expert for a step-by-step guide to create a project, or use the New Project dialog for more control during the process. The New Project dialog allows you to integrate version control as you create a project.

Regardless of whether you use the New Project Expert or the New Project dialog, you can edit the properties of your project *after* creating it. This includes adding and

removing resources as well as integrating version control. See “Editing project properties” on page 99 for more information.

Preparing to create a project

Create a new Together project with or without an existing code base. While the process is similar for either situation, there are some issues to consider when creating a project with existing code.

New projects without existing code

To create a project without using an existing code base, specify a primary root directory for the project. Complete this task by using either the New Project Expert or the New Project dialog. See either “Using the New Project Expert” on page 96 or “Using the New Project dialog” on page 98.

New projects with existing code

Before creating a Together project using an existing code base, determine the following:

- Which content can be modified vs. read-only (that is, shown in diagrams, but not modified within Together)
- Which content should be round-trip engineered

For example, a Java project that includes classes that extend Java classes or some component classes can show dependencies in a visual model, but include parent classes that cannot be modified. The project may also include some or all of the classes residing on the Java classpath of its diagrams; or, display classes or diagrams from another Together project that cannot be modified in the new project. If you compile classes using Together, you need all the resources required by your compiler available to the project, but not necessarily parsed during round-trip engineering.

The resources available to the project are specified as paths. Content can include classes and/or Together diagrams residing in different physical directories on one device, or on different devices. For each resource you define, specify whether or not to allow modification. Use classes from a physical directory, excluding specific subdirectories.

Add or remove project resources as needed (without deleting any physical files), and control what resources can be modified within the context of the project. This is particularly useful when dealing with large code bases comprised of dozens of directories and hundreds of classes. See “Setting up large projects” on page 105 for more information.

Using the New Project Expert

The New Project Expert provides step-by-step instructions for creating a new project.

To use the New Project Expert, follow these steps:

1. From the File menu, choose **New Project Expert** to invoke the dialog, as shown in Figure 18.

2. Provide the information required by the expert, as listed in Table 11, clicking **Next** to proceed to the next step.

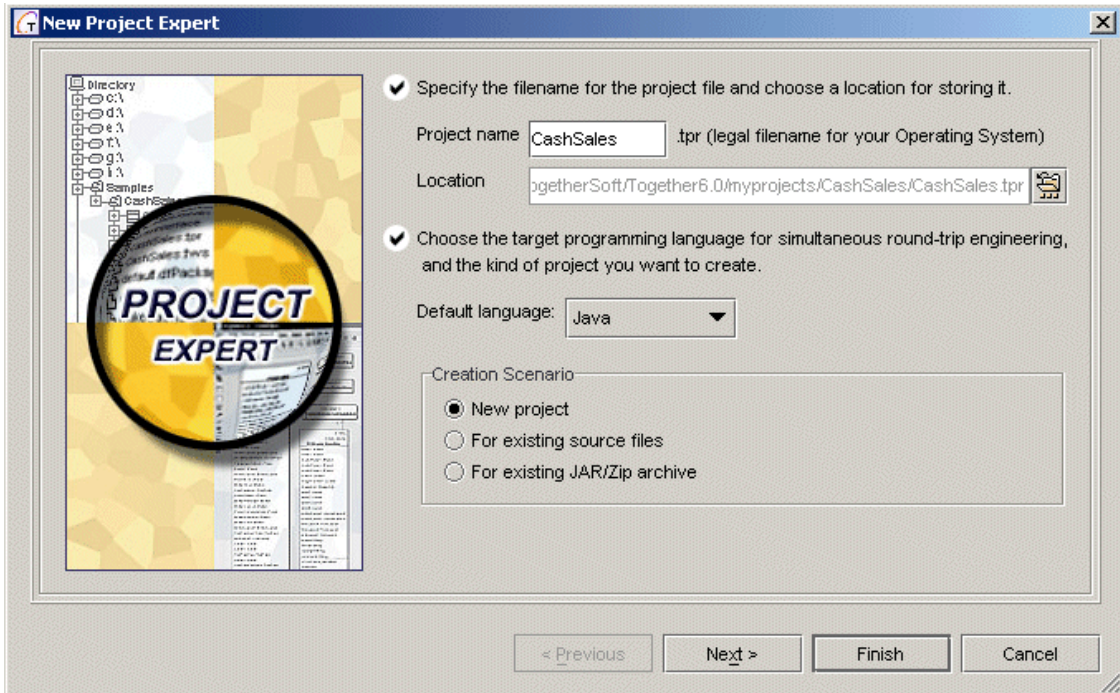
Table 11 Information requested by the New Project Expert

Field name	Description/Notes
Project name	Filename for the project file
Location	Location for the project file; Select the default or browse and select a new location
Default language	Target programming language for simultaneous round-trip engineering and the project you want to create
Creation Scenario	New project: Creates a new project without existing code For existing source files: Creates a new project using existing code For existing JAR/Zip archive: Creates a new project with existing JAR/Zip files
Select the path where the source files should be placed	<i>This appears only after selecting New project.</i> Location where the source files should be placed; Select the default or browse and select a new location
Choose the source files directory	<i>This appears only after selecting For existing source files.</i> Top level directory for the project
Specify a package prefix that Together will use...	<i>This appears only after selecting For existing source files.</i> Exact name that you want Together to use for package statements referencing the selected resource
Choose existing JAR/Zip file	<i>This appears only after selecting For existing JAR/Zip archive</i>
Do you want to place diagram files in the same folder as source files?	Select either Yes... or No...
Choose the initial diagram type?	Select the diagram type or None
Show package dependencies	Check the box for “yes,” or uncheck the box for “no”
Add import paths or JAR/Zip files if your project uses files in additional directories or additional JAR/Zip files	• Check or uncheck Standard libraries and/or Class path • Add or remove paths or archives

3. Click **Finish**.

NOTE: Click **Finish** at any time during the New Project Expert, and Together creates a project with the criteria provided. To view the entire expert, click **Next** until the Next button is disabled.

Figure 18 New Project Expert



Using the New Project dialog

When using the New Project dialog to create a project, you have more control over the process. Unlike the New Project Expert, using the New Project dialog allows you to integrate version control for projects as you create them.

To create a project using the New Project dialog, follow these steps:

1. From the File menu, choose **New**.
2. From the Object Gallery, double-click the **New Project** icon.
3. Complete the information required by the New Project dialog, as listed in Table 12.

Table 12 Preliminary information required by the New Project dialog

Field name	Description
Project name	Name of the project
Location	Location for Together to create the project file and initial and default diagrams; The Location field displays a default path. By default, all items in and under the project directory will be reverse engineered, and can be modified (unless specified read-only at the OS level).
Initial diagram	Type of diagram that Together will generate along with the new project; Select the type of diagram or None.

Table 12 Preliminary information required by the New Project dialog

Show package dependency checkbox	Allows you to automatically draw dependency links between packages. Note that this slows performance for large projects. Rescan the project at another time using the Update Package Dependencies command on the diagram right-click menu.
Default language	Target programming language for the project; When you choose a language, any language-specific options available display next to the language selection.
Components	Checking the Include Components checkbox includes Coad Components in the project. If you check the box, the fully qualified path to the Coad Components displays on the Search/Classpath tab under Resources (this tab appears after clicking Next).

4. Click **Next to proceed.**

The project now contains one root directory — the primary root as specified in the previous step. At this point, you can click **Finish** to create the project if you do not need to include any other directories as part of the project. Or, proceed to the next step.

5. Use the next page of the New Project dialog to further control how Together sets up the project, including:

- Adding resources as detailed on page 101
- Removing resources, as detailed on page 102
- Setting resource options, as detailed on page 103
- Setting up version control for the project, as detailed on page 104

6. Click **Finish.**

After clicking Finish, Together creates the project file with the filename specified and the initial diagram (with the same name) in the specified directory (called the project directory). The project file has the .tpr extension and appears in the Explorer pane. At a later time, modify the project from the Project Properties dialog box as described in “Editing project properties” on page 99.

Editing project properties

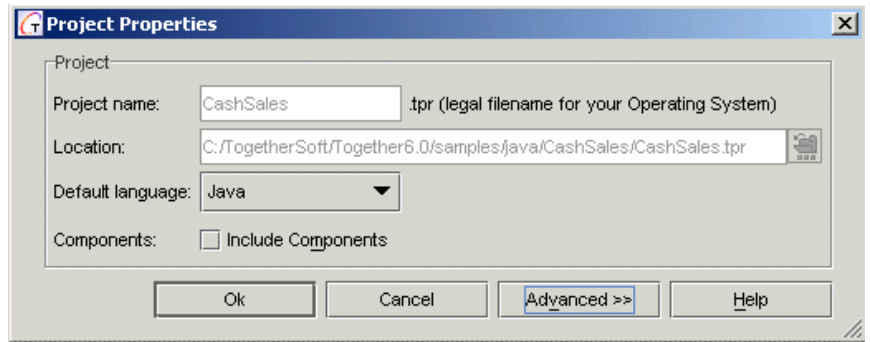
A project’s properties include its name, location, default language, components used, and resources. After initially creating a project, view and edit its properties using the Project Properties dialog.

To access the Project Properties dialog, follow these steps:

- 1. Open a project**
- 2. In the Explorer pane, choose the Model  tab.**
- 3. Right-click on the project name and select **Project Properties**.** The Project Properties dialog appears in default mode, as shown in Figure 19.

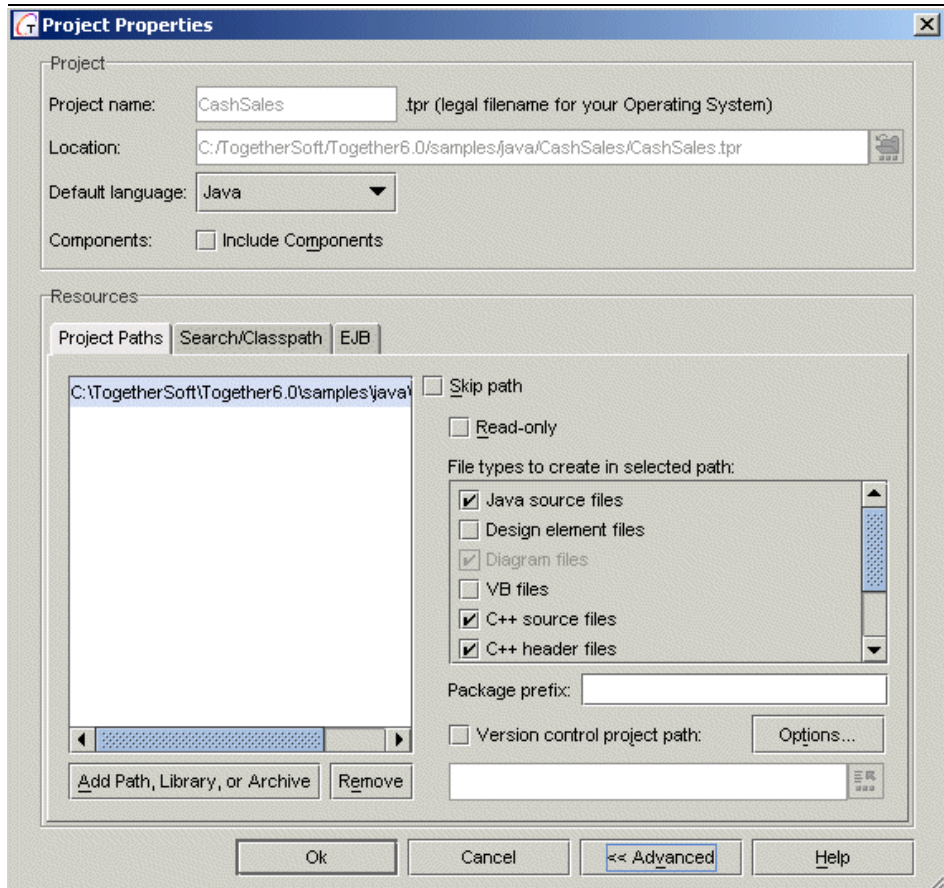
NOTE: Project Properties shows the same fields as the New Project dialog. For a description of the fields, see Table 12, “Preliminary information required by the New Project dialog” on page 98.

Figure 19 Project Properties dialog in default mode



4. If necessary, access advanced mode by clicking **Advanced >>**. The Project Properties dialog appears in advanced mode as shown in Figure 20. For information about the properties that you can set in advanced mode, see the following sections:
 - “Adding Resources” on page 101
 - “Removing Resources” on page 102
 - “Setting Resource Options” on page 103
5. Edit properties as needed and then click **OK**.

Figure 20 Project Properties dialog in advanced mode.



Adding Resources

The information in this section assumes that you have *either*:

- Accessed the Project Properties dialog in advanced mode, as described in “Editing project properties” on page 99; *or*
- Accessed the New Project Dialog, as described in “Using the New Project dialog” on page 98

The Project Paths and Search/Classpath tabs shown in Figure 20 display the lists of the directories and/or archive files currently included as resources available to the project. By default, the path specified in the Location field is present on the Project Paths tab, and the paths to the standard libraries are present in the Search/Classpath list. On the Search/Classpath tab, checking Include Classpath adds your classpath directories to the list.

Items in the Project Paths list are considered as project roots. Roots contain compiled or source classes and/or Together diagrams. Project roots are parsed during reverse

engineering and treated as modifiable unless you specify otherwise in the other controls. See “Setting Resource Options” on page 103 for more information.

The Search/Classpath tab displays resources that reside on the classpath (as defined in your environment) and on any other paths you want Together to search for resources. These resources are available to show in diagrams but are not part of the project. They are not parsed, and do not show as project content in the Explorer unless they are added to a diagram as a shortcut. Also, their content is not modifiable within the project context.

NOTE: If you use an integrated compiler/debugger, make sure all resources required for the tool are included in the Search/Classpath list.

To add a resource directory to the project, follow these steps:

1. Click **Add Path, Library, or Archive** to display the *Select path, library, or archive* dialog box.
2. Navigate to the directory you want to include in the project.
3. Click **Open** or **Select**, depending on the OS you are running on.

Note that all subdirectories of the specified resource directory are included by default. You can exclude some specific subdirectories in one of the following ways.

- If you want the parsing engine to skip them but still want the project tracked by Together, add the individual subdirectories with the Add Path or Archive button, as described above and then check Skip path on the Project Paths tab (see Skip path).
- To exclude some subdirectories completely, use the Ignore files and folders option in the General options group of the Project Options dialog to specify the ignored directories.

In Java projects, some project resources may reside in compressed Zip or JAR archive files. Any new Java project automatically gets `rt.jar` as a “hidden” project root. The path to this archive is determined from the system classpath. This archive is necessary for proper functioning of the integrated Java debugger.

NOTE: If you attempt to add `rt.jar` manually, a message saying “root added implicitly” displays. If you open an older Java project without `rt.jar` specified as a project root, you get a message and the classpath is searched for `rt.jar`, which is then implicitly added.

To add an archive file as a project resource:

1. Select either the Project Paths or Search/Classpath tab under Resources in the New Project (or Project Properties) dialog box.
2. Click the Add Path or Archive button to display the Select Path dialog.
3. Navigate to the directory containing the archive file and select a `.zip` or `.jar` file.
4. Click Open or Select, depending on the OS you are running on.

Removing Resources

The information in this section assumes that you have *either*:

- Accessed the Project Properties dialog in advanced mode, as described in “Editing project properties” on page 99; *or*
- Accessed the New Project Dialog, as described in “Using the New Project dialog” on page 98

Remove any resource from the Project Paths list or the Search/Classpath list by selecting the resource and clicking the Remove button. This is useful, for example, if you included your full Java classpath but it contains some directories that you really don't need in the new project. Removed resources are not deleted from disk, they are just not available to the project. If you need them again, add them later using the instructions in “Editing project properties” on page 99.

Setting Resource Options

The information in this section assumes that you have *either*:

- Accessed the Project Properties dialog in advanced mode, as described in “Editing project properties” on page 99; *or*
- Accessed the New Project Dialog, as described in “Using the New Project dialog” on page 98

After adding resources to the project, set options that control how they are parsed and how you access them when modeling. By default, parsing of all the paths in the Project Paths list includes all recognized file types, and all resources are parsed as part of the project and display in the Explorer's Model tab. To change the default setting, modify the treatment of each of the resources in the Project Paths list.

To set options for a resource, select it in the Project Paths list and modify any or all of the following:

- **Skip path:** Selecting this option causes the parsing engine to skip the selected resource, but the resource is still tracked with the project by Together and may be included in documentation generation or accessed as part of model information by modules.
- **Read only:** Checking this box makes the contents of the resource a read-only part of the project. It displays as project content in the Explorer's Model tab but is not modifiable from within the project. When checked, File types and Package prefix options are disabled.
- **File types to create in selected path:** Check only those file types you want the parsing engine to work on. This can speed parsing of resources having different types of content. Use the default (all file types) unless you want to skip source files not in the project's target language, or you want to specifically exclude compiled classes.
- **Package prefix:** Specify the exact name you want Together to use for package statements referencing the selected resource.
- **Version Control project:** Checking this box enables the version control integration defined in your configuration for the project. The Select button enables you to select the project or repository of the currently configured version control system that you want the project to use. For more information, see “Setting-up version control for projects” on page 104.

Running multiple project instances

By running multiple instances of Together, you can work with several projects (or files associated with Together). Note that you can run multiple instances Together, but you can run only one project in each instance.

Configure options to control how Together reuses instances. For example, select one of the following options:

- **Reuse current instance:** File always opens in the current instance of Together (without requesting confirmation). If this requires closing the project, the project is closed without any warning.
- **Run new instance:** File always opens in a new instance of Together (without requesting confirmation).
- **Always confirm reuse:** Before opening a file, a dialog opens that asks whether you want to reuse the current instance or start a new instance of Together.
- **Confirm reuse only if project must be closed:** Together asks for confirmation only when reusing the current instance requires closing the current project. If the file can be opened without closing the current project, it is automatically opened in the current instance of Together.

To set options for multiple project instances, follow these steps:

1. From the Tools menu, choose **Options | Default Level**.
2. From the tree of the Default Options dialog, select **General**.
3. Select a value for *Reusing Together instances*.
4. Click **OK** and then **Apply**.

NOTE: This option setting overrides the option *Confirm when closing project*.

Setting-up version control for projects

To set-up version control for a project, you need to complete two tasks:

- Enabling version control for your installation of Together
- Specifying a version control system for each Together project

To enable version control, follow these steps:

1. From the Tools menu, choose **Options | Default Level**
2. From the Project Options tree, choose **Version Control**
3. Check the box for *Version Control enabled*.
4. Click **Apply**.
5. Click **OK**.

To specify a version control system project for a Together project, follow these steps:

1. From the Project menu, choose **Version Control | Set up for this Project**.
2. Check the **Version Control** option and specify the project path for version control.

3. Click **Options** to view the options for Version Control and edit as needed in the **Project Options: Version Control** dialog.
4. Click **Cancel** or **Apply** to return to the Project Properties dialog.
5. Click **OK**.

NOTE: Specify a version control system as you create a new project using the New Project dialog. Check the Version Control option and specify the version control system repository.

Setting up large projects

A project can contain large code bases comprised of dozens of directories and hundreds of classes. Such large projects often consist of subsystems, with modules or components within the subsystems. Together translates this to projects and subprojects.

Instead of creating a single project that encompasses an entire code base, identify the subsystems and the modularity within them, creating a number of Together projects in key directories of particular interest or significance. As an example, refer to the following project located under your Together installation: `$TGH$/modules/components/CoadModelingComponents/all.tpr`. Although this is not a particularly large code base, it illustrates the project-subprojects organization you can use for larger projects.

Also, consider automating documentation generation for very large projects. Write a script or module to regenerate all documentation or a set of scripts to update different parts, which run automatically at different times. See Chapter 17, “Generating Documentation for Together Projects” for more information.

Creating views with referenced content

Together’s project definition feature allows you to create projects with content that is purely logical, containing only the items you need to see for a particular purpose. For example, you could create a directory that primarily provides a view of your code base. Under such a directory, you can create a series of project folders that contain Together projects that bring in different parts of your code (for example, only abstract classes in your problem domain).

When creating such projects, specify the subdirectories to use as resource roots. Other directories with classes (and other elements) that you might want to show (but not modify) can be specified in the Source/Classpath list. See “Creating new projects” on page 95 for more information.

Performance tuning

When working on a project with a large code base, consider the following issues that may impact performance:

- Check the resource paths used by the project. Deep path names such as `C:` can hinder performance. For instructions on how to view resource paths, see “Editing project properties” on page 99.

- Check the memory used while parsing. If memory increases close to the maximum, try to increase the swap file size. (Set the minimum swap file size in the range of 100-150M).
- Try running Together with `jre.exe`, the Sun virtual machine (VM) that provides options for specifying the Java heap size. Change the maximum heap size (`-mx32m`) to a larger amount (such as `-mx60m` or `-mx100m`, depending on the availability of virtual memory). Make sure your computer hardware has sufficient power and system resources to handle the values you specify in order to achieve the effect you want.
- Monitor and manage the memory usage via the *Memory Status* dialog. To open this dialog, use the keyboard shortcut `Ctrl+Shift+T`. Use the *Garbage Collect* button to release the unused memory.

REVERSE ENGINEERING

One of Together's main features is *simultaneous round-trip engineering*, the ability to immediately synchronize class diagrams with their implementation code.

Simultaneous round-trip engineering means that your UML class diagrams are always synchronized to the source code that implements them. When you change a class diagram, Together immediately updates the corresponding source code. When you change the code, Together updates visual model. There is no intermediary repository, no batch code generation, and no risk of losing code.

Simultaneous round-trip engineering applies to existing code as well as code that is being developed. Together can *reverse engineer*, building a model around existing code or restore a model from archived files.

This chapter includes the following topics:

- “Reverse engineering techniques” on page 107
- “Reverse engineering archive files” on page 109

Reverse engineering techniques

You can create a new project using existing code or incorporate the code into an existing project. The existing code can be source code, *.jar files, or *.zip files.

Creating a new project with existing code

The easiest way to create a new project from existing code is to use the *New Project Expert*.

To reverse engineer code with the *New Project Expert*:

1. Select **File | New Project Expert** from the main menu.
2. In the resulting dialog box, enter the new project name, location, and language. Select the *Creation Scenario* according to the existing code base.

- *For existing source files*
- *For existing JAR/ZIP archive*

Click **Next** to continue.

3. Enter the location of the existing code in the first field. Enter a package prefix in the second. Click **Next** to continue.
4. Click the radio button to determine if the diagrams and code to be in separate folders.
 - *Yes, I want to have them placed in the same folder.* The default choice.
 - *No, I want to separate them.* Separates diagrams from code.
 Click **Next**, then enter the name of the diagram folder in the text field.

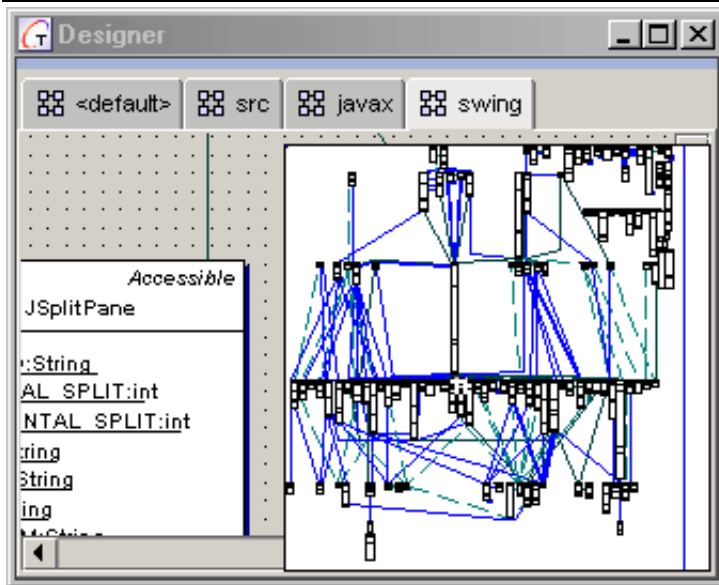
Click **Next** to continue or **Finish** to complete the process with default values for initial diagram type, package dependencies, and project paths.

5. Click **Finish** to close the dialog and start the reverse engineering.

Together generates a set of class diagrams according to structure of the source code (physical class diagrams). The <default> diagram corresponds to the top level package of the project.

Figure 21 shows results from reverse engineering JDK 1.3. The Designer pane displays the swing library. The overview (which takes half of the window in our snapshot) indicates the extent of the library and the connections among its classes.

Figure 21 Overview of class diagram for Swing



NOTE: You can create a new project with existing source code without using the *New Project Expert*. In that case, however, the physical class diagram files are stored in the same folders as the source files.

Incorporating code into an existing project

Together can reverse engineer code that you incorporate into an existing project.

To reverse engineer code by incorporating it into an existing project:

1. Open the project.
2. Select **Project | Properties** from the main menu.
3. In the *Resources* section, make sure that the **Project Paths** tab is at the front.
4. Click the **Add Path, Library, or Archive** button at the bottom of the dialog box.
5. Select the directory containing the source code files or the *.jar or *.zip file from the resulting file/path chooser dialog.
6. Determine if Together should treat the code as read-only (the default).
 - Together treats *.jar and *.zip files as read-only.
 - If you are incorporating source code into the project, un-check **Read Only** if you want to be able to modify the source code or add files to the source code folders. You can check the types of files to create in the project path.

NOTE: If you un-check **Diagram Files**, Together creates new folders inside its project directory to hold the diagram files that it generates.

7. Click **OK** to incorporate the code and quit the dialog.

Use the Model tab of the Explorer to open the diagrams for the reverse-engineered code.

TIP: When you reverse engineer Java code, run the audit *Detect Collection-Based Associations* (under *Documentation* audits) so that Together can display associations for collections without your manually inserting @association tags into the code. This auto-fix audit inserts the tags for you.

Reverse engineering archive files

Together extends reverse engineering to J2EE archive files. It is possible to restore diagram structure from an existing *.jar, *.ear or *.war file. If you have implemented and deployed EJB, Enterprise Application, or Web Application modules, you can include them to your Together project. In process of reverse engineering, Together generates the corresponding diagrams with the relevant visual design elements.

To reverse engineer a diagram from an archive file:

1. On the main menu choose **File | Import | J2EE Module Import** command.
2. In the field *J2EE module location* field, specify the source archive.
3. In the *Extracted module location* field, enter the target folder.

Result: Together restores appropriate diagram.

The extracted module corresponds to the type of archive file. So doing, Together restores the structure of Web Application and EJB Assembler diagrams included in an Enterprise Application.

If the imported module contains JSP files, Together will restore them as JSP if only they have been marked as JSP in the deployment descriptor. Otherwise, Together will regard them as ordinary web files.

NOTE: At the time of publication, importing J2EE modules is subject to limitations. Specifically, Security Constraints from *.war and *.jar archives cannot be imported to the diagram.

IMPORTING AND EXPORTING INFORMATION

Together reads and generates information in multiple formats. This chapter explains many of the import and export operations that Together supports, focusing on DDL, IDL, and XMI.

This chapter contains the following topics:

- “Overview of import and export operations” on page 111
- “Importing and exporting JDBC database information” on page 112.
- “Generating IDL” on page 114.
- “Importing and exporting XMI models” on page 115.

Overview of import and export operations

Together supports several different kinds of import and export operations:

- Import/export model information to XMI.
- Import/export to JDBC databases via DDL files.
- Generate Interface Definition Language (IDL) for a project.
- Export/Import from XML diagrams to DTD or XSD. (This is covered in Chapter 13, “XML Modeling.”)
- Export/Import from class diagrams and ER diagrams to DTD. (This is covered in Chapter 13, “XML Modeling.”)

Importing and exporting JDBC database information

Information can be imported from JDBC-enabled databases into Together. Additionally, users can export model information from Together to create model-based database schemas. Together supports the following RDBMS types:

- mySQL 3.23
- Sybase AS Anywhere 6.x / 7.x
- IBM DB2 6.1 / 7.x
- SequeLink/Oracle
- Sybase AS Enterprise 12
- Oracle 8.x/.9
- Access 97/2000 (ODBC)
- MS SQL Server 7.0 (ODBC)
- Cloudscape 3.5

Activating database import and export

DB Modeling is an activatable feature for importing from a database or exporting to DDL.

To activate DB Modeling:

1. Open a project.
2. Choose **Tools | Activate/Deactivate Features** on the main menu.
3. In the resulting dialog box, go to the **Together Features** tab.
4. Check the box in the Activate column to the left of the module named **DB Modeling**. Click **OK** when finished.

TIP: Deactivate any features you do not need and activate any others you do need simultaneously. Deactivate unused features in order to free up system resources.

With DB Modeling activated, the main menu contains two database import/export commands: **Tools | Generate | DDL** and **File | Import | Database schema**.

Generating DDL (exporting to DDL)

With Together's DB Modeling feature, generate a data definition language (DDL) file from a class diagram, an entity relationship (ER) diagram, or entity EJBs. A compliant DBMS system can read the DDL to create a table schema. Together gives a choice of generating DDL files only or generating the DDL files and then running them in the same operation.

To generate DDL:

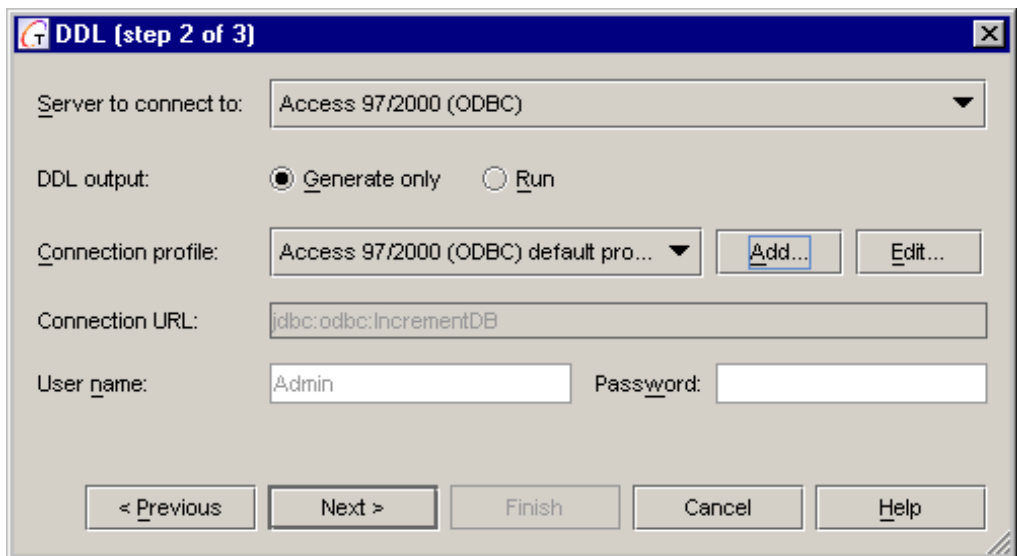
1. Open a project containing the diagram with the source for information to export.

2. Open the class diagram, ER diagram, or the class diagram with an entity EJB that contains the information for the database schema.
3. On the main menu, choose **Tools | Generate | DDL**. (The DB Modeling feature must be activated for this command to be on the menu.)
4. The resulting DDL dialog box has 3 panes for specifying the details of the DDL generation.
 1. **Source model /code information:** Indicate class diagram, ER diagram, or entity EJBs and the corresponding diagram for the source of information.
 2. **Target database:** Enter the specific database information. The DDL file generated corresponds to that database schema. Figure 22 shows the second pane after creating a proper connection profile.

If you select **Run** from **DDL output**, you must select a **Connection profile** to an actual database that Together can access. You can click **Add** to create a new profile or edit an existing one.

- 3. **DDL file:** Enter the name and location of the generated DDL file. If desired, edit the file. Click **Finish** to generate the DDL file.

Figure 22 Exporting to DDL



TIP: If you are exporting DDL from a class diagram, all classes in the diagram must be “persistent.” Set the persistence for each class through its Properties (accessed via the right-click menu.)

Importing database meta-information

The DB Modeling feature enables you to import schematic information from a supported database to create a class diagram, an ER diagram, or entity EJBs.

If the database does not already exist, you should use your DBMS administration utility to create a JDBC/ODBC data source of the source database prior to importing.

To import a database:

1. Open a Together project or create a new project in which to store the imported information.
2. Choose **File | Import | Database Schema** from the main menu. The DB Modeling feature must be activated for this command to be on the menu.
3. The resulting Database Schema dialog box has 3 panes for specifying the import details.
 1. **Source database:** Choose the supported DBMS, set a connection profile, and connect to the database.
 2. **Tables:** Select each table that you want to import from the left frame (Available Tables) and add it to the right frame (Tables for Import).
 3. **Import to:** Select the kind of import (EJBs, class or ER diagrams), and the diagram to store the information. Use an existing diagram or create a new one. Click **Finish** to complete the import.

Generating IDL

If a Together project contains class diagrams appropriately structured to support IDL, generate and then round-trip engineer IDL for specific diagrams, classes, packages, or for the entire Together project.

IDL Export Support is an activatable Together feature that is deactivated by default. You can activate it in the usual way, as described in “Activating database import and export” on page 112.

When the IDL Export Support is activated, the main menu contains the command **Tools | Generate | IDL**.

Exporting IDL from class diagrams

After activating IDL Export Support, generate IDL files from your projects.

To generate IDL:

1. Open the project containing the source to be exported to IDL.
2. If you want to generate IDL for a specific package, open its diagram in the Designer pane. (This is not necessary for generating IDL for the entire project.)
3. Choose **Tools | Generate | IDL** from the main menu.
4. In the resulting dialog box, select:
 - The package structure: **Current** (package), **Current with subpackages**, or **All** (the entire project).

- IDL type: **COM IDL** or **CORBA IDL**.
 - Target directory: for the generated files.
5. Optionally review and change the IDL Options offered in the dialog. Click **Options** and use the on-screen help to change any option settings. Click **OK** to make any changes and close the IDL Options dialog box.
 6. Click **OK** in the Generate IDL dialog box to generate the IDL.

TIP: Open the message pane before generating IDL to view the messages generated by the process.

Round-trip IDL support

Together supports round-trip IDL engineering. You do not need to import IDL into a project. Simply create a new project specifying the location of your existing IDL code as one of the project's resource roots and continue working with it in Together. The process is the same as creating any new project, except that you choose IDL as the default programming language during the creation process.

Importing and exporting XMI models

Import a model described by an XMI file into a Together project. This generates source code in the primary programming language of your Together product. You can export model information from a Together project to an XML file as well.

NOTE: XMI import-Export requires Sun JRE version 1.2x or above

XMI import and export require activating the **Import-Export** feature. (Activating a feature is described in “Activating database import and export” on page 112.) When Import-Export is activated, the main File menu has Import and Export commands.

Importing XMI models

If you are importing a large model, close other applications to free up memory.

To import a model described in an XMI file:

1. Create a Together project for the model or open an existing project to which you want to add the XMI model. Create or navigate to the desired package.
2. On the main menu, choose **File | Import | Model from XMI File**.
3. In the resulting dialog box, select XMI file and click **OK**.

The amount of time you must wait while Together processes the XMI code depends on the size of the model.

When Together finishes processing, it creates a new package diagram and opens it in the Designer pane. The Model tab of the Explorer displays the packages and the default class diagram for each package. When opening diagrams and selecting classes, the Editor displays the generated source code.

Together creates a log file with information about import operations and places it in the source folder of the imported xml file.

NOTE: If you plan to export a model from Rational Rose® to XMI and later import it to Together, you should choose the ASCII/MBCS option in the Character Set options in Rose's Unisys XML Export dialog. Models exported to other character sets will not be properly imported into Together.

Exporting XMI models

Export a Together model to an XMI file. Together supports UML 1.1 and UML 1.3 Unisys XMI interchange for all standard UML diagrams, IBM XMI Toolkit, and OMG XMI.

To export a Together model to XMI:

1. Open the Together project you want to export.
2. On the main menu, choose **File | Export | Model to XMI File**.
3. In the resulting dialog box:
 - Select the location and specify the filename for the generated XMI file.
 - Select the XMI type from those listed, then choose the option specifying which diagrams from the current project you want exported to XMI.
4. Click **OK** to start the export process.

The time required to generate the XMI code depends on the size of the model. Together displays the progress of the operation on its status bar.

Figure 23 shows the Export to XMI dialog box, which gives eight options for the XMI type.

User configurable translations

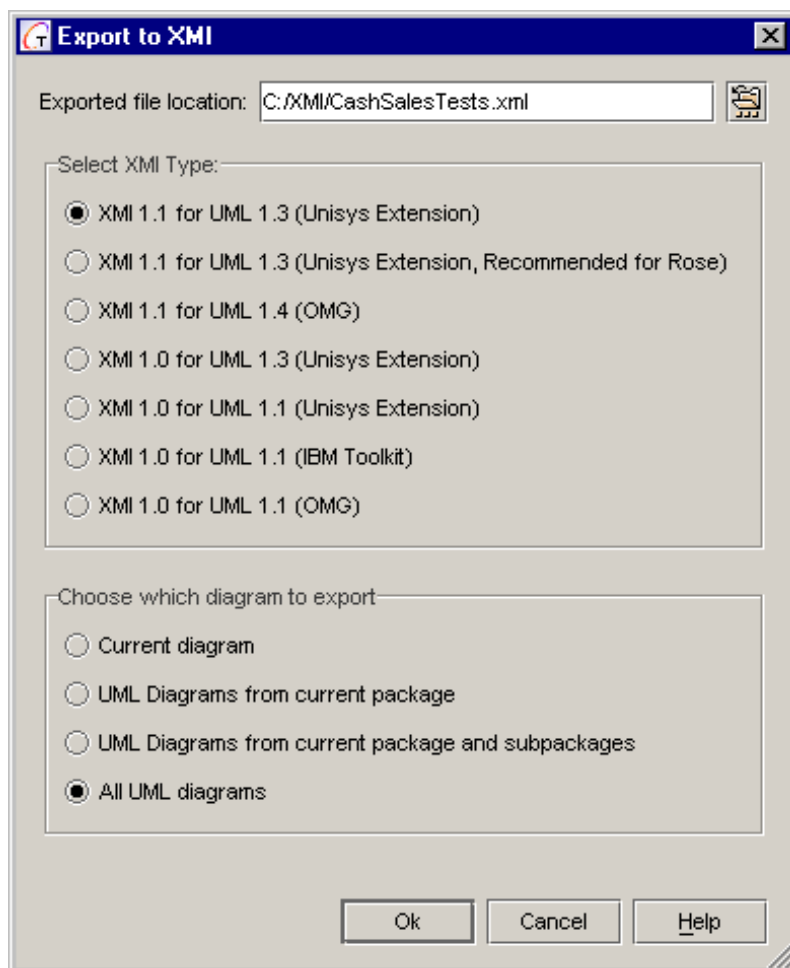
To support compatibility between Together model tags and XMI tags, the interchange module provides a list of translations that contain entries in the format:

`Tag_Name = RationalRose$MDC/:Tag Name`

To import or export some specific properties of your project from or to XMI, edit this config file:

`%TGH%/modules/com/togethersoft/modules/interchange/InterchangeTags.config`

Figure 23 Export to XMI dialog box



IMPORTING DATABASE INFORMATION

Together is used to import both modeling and database information. For information about importing model information, see the previous chapter.

This chapter includes the following topics:

- “Importing-Exporting JDBC Database Information” on page 118
- “Importing database meta-information” on page 119

Importing-Exporting JDBC Database Information

Together allows importing information from JDBC-enabled databases and exporting model information to create model-based database schemas. The following RDBMS types are currently supported:

- ODBC/Access 97/2000
- Cloudscape 3.5
- ODBC/MS SQL Server 7.0
- SequelLink/Oracle
- Oracle 7.3x, 8.x, 9i
- IBM DB2 6.1
- Sybase AS Anywhere 6.x / 7.x
- Sybase Enterprise Server 12.0
- MySQL 3.23
- DB2 v7.1

Importing database meta-information

With this feature you can import schematic information from a supported database to create a Class diagram, an ER diagram, or an EJB.

To run Import Database:

- 1.** Use your DBMS administration utility to create a JDBC/ODBC data source for the source database, if such a data source does not already exist. (You will need to specify the data source during the import procedure.)
- 2.** Open a Together project in which to create diagram(s) from imported schematic information (or create a new project).
- 3.** In the Explore (Model tab) navigate to the desired package.
- 4.** Choose File | Import | Import Database Expert on the main menu. The Import Database Expert appears.

Follow the Expert steps. Choose the supported DBMS, set a connection profile, connect to the database, and choose the tables to import.

MODELING WITH TOGETHER

- Chapter 8, “Introduction to Modeling”
- Chapter 9, “Working with Diagrams”
- Chapter 10, “UML Modeling”
- Chapter 11, “UML Extension Diagrams”
- Chapter 12, “Real-time Modeling”
- Chapter 13, “XML Modeling”

INTRODUCTION TO MODELING

The primary objective of modeling is to organize and visualize the structure and components of software intensive systems. Models visually represent requirements, subsystems, logical and physical elements, and structural and behavioral patterns. While contemporary software practices stress the importance of developing models, Together extends the inherent benefits of modeling by fully synchronizing diagrams to source code.

This chapter explains how the topics of modeling and diagrams are organized in this *User Guide*. It includes the following topics:

- “Overview of modeling” on page 121
- “Diagrams supported in Together” on page 122

Overview of modeling

In the *UML User Guide*, Booch *et al* cite a number of activities involved in modeling a system's architecture. These activities include:

- Identifying different architectural viewpoints such as use case, design, process, implementation, and deployment views
- Identifying the system context and the actors involved
- Decomposing a large, complex system into more granular subsystems.

In addition, the authors outline various views that apply to both the overall system and each of the subsystems:

Use Case view: Represents use cases describing system behavior as seen by analysts, end users, and testers. Models consist of Use Case diagrams for the static aspects of a system and appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Design view: Represents a design view specifying classes, interfaces, and collaborations. These provide a working vocabulary for the system in terms of problem

and solution. Models consist of Class diagrams (including Objects as necessary) to represent static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Process view: Represents a process view to describe the threads and processes of synchronization and concurrency mechanisms. Models consist of diagrams used for the Design view with active classes and objects representing threads and processes.

Implementation view: Represents the components used to build and release the system. Models consist of Component diagrams for the static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Deployment view: Represents the nodes, components, and interfaces forming the hardware topology for the runtime system. Models consist of Deployment diagrams for static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams to for dynamic aspects.

Patterns: Represent the architectural and design patterns of each of the previous models with appropriate diagrams to show collaborations.

Diagrams supported in Together

Together supports all of the Unified Modeling Language (UML) diagrams in addition to specialized diagrams for modeling business data, business processes, and robustness analysis; in addition to real-time modeling, and XML modeling. Table 13 lists the diagrams supported in Together and indicates where to find respective information in this guide.

TIP: For information on working with specific diagrams, see the references in Table 13. For instructions that apply to all diagrams such as creating, editing, and hyperlinking diagrams see Chapter 9, “Working with Diagrams”.

Table 13 Diagrams supported in Together

This section...	Includes these diagrams...
Chapter 10, “UML Modeling”	“Use case diagrams” on page 151
	“Class Diagrams” on page 154
	“Sequence and Collaboration Diagrams” on page 162
	“Statechart Diagrams” on page 173
	“Activity Diagrams” on page 177
	“Component Diagrams” on page 179
	“Deployment Diagrams” on page 182
Chapter 11, “UML Extension Diagrams”	“Entity relationship diagrams” on page 185
	“Business Process Diagrams” on page 190
	“Robustness analysis diagrams” on page 192

Table 13 Diagrams supported in Together (continued)

Chapter 12, “Real-time Modeling”	“Use case diagrams” on page 198
	“System context diagrams” on page 199
	“System architecture diagrams” on page 203
	“Statechart diagrams” on page 206
	“Event sheet diagrams” on page 206
	“Interaction diagrams” on page 208
Chapter 26, “J2EE Support”	“EJB Assembler diagram” on page 462
	“Web Application diagram” on page 462
	“Enterprise Application diagram” on page 463
	“Taglib diagram” on page 463
	“Taglib diagram” on page 463
	“Resource Adapter diagram” on page 464

WORKING WITH DIAGRAMS

This chapter outlines the basic techniques that users apply when working with diagrams. The following topics are explained in this chapter:

- “Creating diagrams in projects” on page 124
- “Drawing diagram elements” on page 127
- “Working with node elements and links” on page 132
- “Viewing diagrams and managing diagram elements” on page 136
- “Editing diagrams” on page 137
- “Hyperlinking diagrams” on page 141
- “Annotating diagrams” on page 144
- “Standalone Design Elements” on page 145
- “Saving and copying diagram images” on page 146
- “Printing diagrams and source code” on page 147

Creating diagrams in projects

Diagrams exist within the context of a project. Create or open a project before creating any new diagrams. If you create a Together project around an existing code base, Together automatically generates Class diagrams showing the contents of each package when it parses your code. Before creating a project, you may customize forward and reverse engineering and/or source code formatting (see [Chapter 2, “Setting Your Personal Preferences”](#)).

You can create any of the diagram types your license supports. Diagrams fall into two basic categories:

- UML diagrams (Class, Use Case, Sequence, and so on)
- Special Together diagrams (Entity Relationship, EJB Assembler, XML Structure, and others)

When you begin a new project or add new diagrams to an existing project, create diagrams using:

- The main menu or toolbar
- The Hyperlinking feature
- The Clone command

The created diagrams are stored in text files with the extension `*.dfDiagramType`. For example, the file name `.dfClass` corresponds to a class diagram, and name `.dfUseCase` corresponds to a Use Case diagram.

Using the main menu or toolbar

To create a new diagram in a project:

1. Select the destination directory or package on the Project Explorer's Directory or Model tab.
2. Choose **File | New** on the main menu or click the New button on the main toolbar to display the Object Gallery.

TIP: Alternatively, click **New** button on the main toolbar.

3. In the *Templates* pane of the Object Gallery, select the Diagram icon and click **Next** to open the New Diagram dialog.
4. Select the UML or Together tab as required, and click the icon for the type of diagram you want to create (only those diagrams enabled by your license are enabled).
5. In the *Diagram name* field specify the diagram name.
6. In the *Package name* field, specify the destination package for the new diagram file. The default destination is the package currently selected in the Explorer.
7. Check *Include in current diagram* if you want to add an element to the current diagram with an icon representing a logical link to the new diagram.
8. Click **Finish**. The new diagram opens in the Diagram pane.

Using the Hyperlinking feature

The Hyperlink feature enables you to create a new diagram that automatically links to an existing diagram or diagram element.

To create a hyperlinked diagram:

1. Choose **Properties** on the right-click menu for a diagram or diagram element.
 2. In the Properties Inspector, select the Hyperlink tab.
 3. Right-click on the **Element** node.
 4. Choose **New Diagram** on the right-click menu.
 5. In the New Diagram dialog, select the diagram type and follow the usual procedure.
- For more information, see “Hyperlinking diagrams” on page 141.

Cloning diagrams

The Clone command lets you quickly create a new diagram with the same content as the existing one.

To clone a diagram:

1. In the Model tab of the Explorer, select the diagram to be cloned.
2. Choose **Selection | Clone** on the main menu.

TIP: Alternatively, choose the Clone command from the right-click menu of an existing diagram node on the Explorer Model tab.

The new diagram is created with a unique default name in the same package as the original diagram.

Renaming diagrams

To rename a diagram using the Inspector:

1. Open the diagram for editing.
2. Click on the background to display diagram properties in the diagram Inspector.
3. Modify the *Name* property in the diagram Inspector.

To rename a diagram in-place:

1. In the Model tab of the explorer, select the desired diagram node.
2. Double-click on the diagram name or press the **F2** button to activate the in-place editor.
3. Change the name as required and press **Enter**.

Configuring diagram options

The Diagram node of the Options dialog provides a number of customization settings. You can set these options either globally for all diagrams, project-wide for just the current project, or locally for the current diagram. See Chapter 3, “Configuring Options” for information on setting up different levels.

To configure the diagram settings:

1. Choose **Diagram Options** on the diagram’s right-click menu. The Options dialog opens on the Diagram level.
2. Change the configuration level if necessary.

TIP: Alternatively, choose **Tools | Options | <level> - Diagram**.

3. Specify diagram options, following the instructions in the Description pane of the Options dialog.

Options for improving graphics

To make your diagrams look better, pay attention to the options that help improving graphics.

3D look

To make your diagram look three-dimensional, open Diagram Options and check *3D look*. This causes a shadow to appear under each diagram element to create a three-dimensional effect. This option applies only to those elements that have the 3D look option set to default on the View tab in the Properties Inspector.

Anti-aliasing

When zooming in a diagram, you might observe rough lines, particularly on links. To smooth the lines being drawn, open Diagram Options and check *Antialias graphics*.

See also:

- [Opening diagrams for editing](#)
- [Creating and opening a project](#)
- [Configuring the New Diagram dialog](#)

Drawing diagram elements

This section covers the basic techniques for placing diagram elements and annotations into diagrams and drawing relationship elements between them. Drawing diagrams with Together is simple and intuitive. If you have used earlier versions of Together, you can probably skip or just skim this information.

When you create a new diagram, the Diagram pane presents an empty background. You place the various Node elements (such as Class, Use Case, and so on) on the background and draw relationship links between them (Association, Communicates, and so on). What elements and relationships you draw is up to you - whatever meets the requirements of your model.

The main tools you use for constructing diagrams are:

- **Diagram Toolbar buttons.** Use toolbar buttons to place icons for Node elements and draw links on the diagram background.
- **Diagram right-click menu.** Right-click on the diagram background to show the diagram right-click menu. Use the right-click menu to show hidden objects, manage the layout, control zoom, configure diagram-level options, and update diagrams and hyperlinks.
- **Element right-click menus.** The right-click menus of the various nodes and links provide functions specific to each. For example, you can add or delete members (or delete the element itself), cut-copy-paste, hide and show elements, route links, and more. Explore the right-click menus of the different elements as you encounter them to see what is available for each one.
- **Inspectors.** (Right-click menus | Properties) Use Inspectors to edit diagram or element properties, create hyperlinks between diagrams, elements of diagrams and other diagrams, and between diagrams or elements and files or URLs. Edit annotations and source code comments.

Undo/Redo

All operations are reversible. Undo and Redo commands work with changes that affect the visual diagram, layout of elements, adding of Attributes and Operations, and so on. These features are not available when the change to the diagram involves creating or renaming a file or directory, such as creating and renaming classes, interfaces, or packages.

- Choose **Edit | Undo** on the main menu, or press **Ctrl + Z** to undo the last change you made to the diagram.
- Choose **Edit | Redo** on the main menu, or press **Ctrl + Y** to restore the last change you made using Undo.

TIP: The size of the Undo/Redo buffer is configurable in the global or project-local `workspace.config` file.

Using the grid

You can optionally display or hide a design grid on the diagram background and optionally have elements “snap” to the nearest grid coordinate when you place or move them. Grid display and snap are enabled by default.

To control grid parameters:

1. Choose **Tools | Options | <level>** on the main menu (where <level> is Default Level, Project Level, or Diagram Level, depending on how broadly you want the settings to apply).
2. Expand the *Diagram* node of the Options dialog.
3. Select the *Grid* node.
4. Set the *Show grid* and *Snap to grid* options as desired.
5. Specify the granularity of the grid in the fields *Grid width* and *Grid height* in pixels.

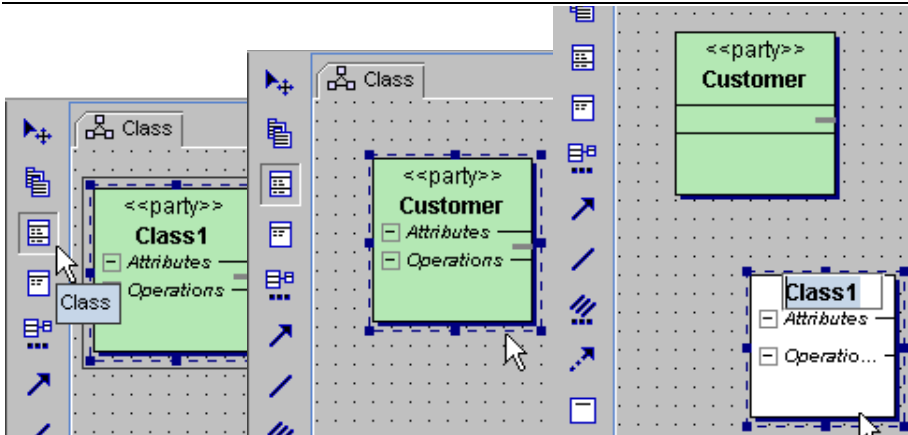
Drawing node elements

To place a node element on a diagram:

1. Create or open a diagram.
2. On the Diagram toolbar, click the button for the element you want to place in the diagram. (Icons are identified with tool tips.) The button stays down.
3. Move the pointer over the Diagram pane to the place where you want to create the new element, and click. This creates the new element and activates the in-place editor for its name. (Note: You can place multiple elements of the same type in a single operation. See “Tips and tricks” on page 137.)

TIP: Alternatively, right-click the diagram and choose **New** from the right-click menu. The submenu displays all basic elements that can be added to the diagram, and the Shortcuts command.

Figure 24 Steps of placing a node element



Adding multiple elements

Place a number of nodes of the same type on the diagram without returning to the toolbar. These will all have default names which can be edited in-place or in the properties Inspector for each one.

To create multiple similar elements:

1. Holding down the **CTRL** key, click on the toolbar button for the node you want to create (the button stays down).
2. Release the **CTRL** key.
3. Click the desired location on the Diagram pane. The new element is placed on the diagram at the point you click (the button stays down).
4. Click the next location on the Diagram pane. The next new element icon is placed on the diagram.
5. Repeat the previous step until you have one less than the desired number of elements of that type.
6. To place the last element in the series, click once more on the background and close the in-place editor for the element name.
7. Click the **Select** toolbar button or just press **ESC**.

TIP: After making a selection on the toolbar or doing the first of a multi-draw or multi-placement operation, cancel the operation by clicking the Select Arrow on the toolbar.

Drawing relationship links

To draw a relationship link between nodes:

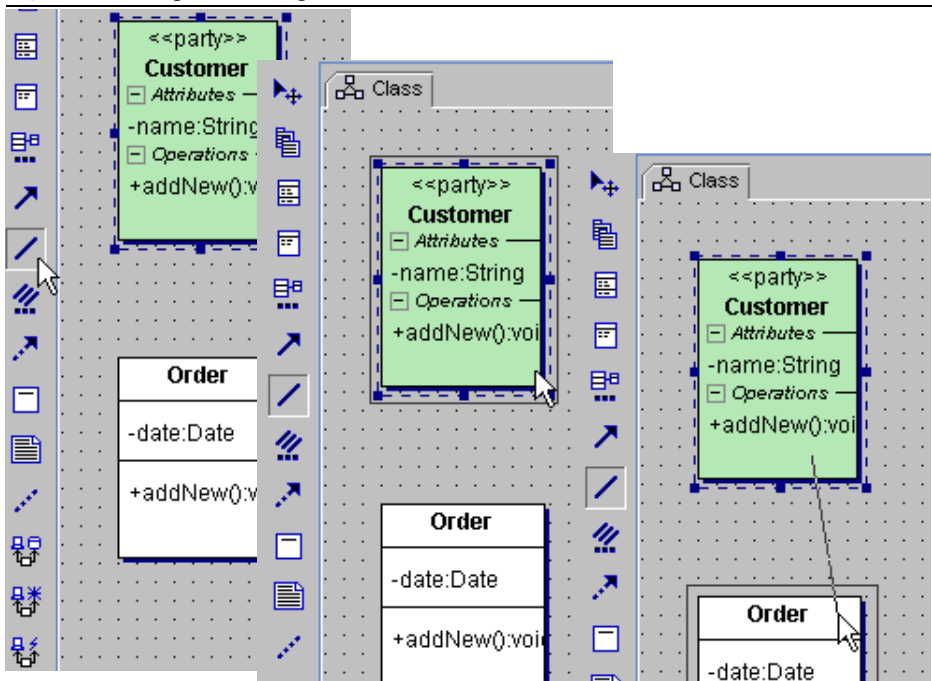
1. On the Diagram toolbar, click the button for the type of link you want to draw in the diagram. The button stays down.
2. Click on the source element.

3. Drag to the destination element.
4. Drop when the second element is highlighted.

TIP: If the destination element is out of reach, there are two ways you can select it:

- Drag the pointer to the edge of the diagram window so that it scrolls to the desired location.
- Drop the link on the background to display the Choose Destination dialog, and choose the destination element from the list. (Note: You can place multiple links of the same type in a single operation. See [“Tips and tricks” on page 137.](#))

Figure 25 Steps of linking node elements



Drawing links with bending points

If your diagram is densely populated, draw bent links between the source and target elements to avoid other elements that are in the way.

To create a link with bending points:

1. Click the link button on the toolbar.
2. Click on the source element.
3. Drag the link line, clicking on the diagram background each time you create a section of the link.

4. Finally, click on the destination element.

TIP: You can configure the links to drawn bent at right angles. To do that, choose *Rectilinear* option in the Diagram options (*Diagram options* | *Diagram: Links*)

Drawing a link to self

On the Diagram toolbar, choose the Association link button and double-click the element. This draws a link to self.

Drawing dependency links

Together automatically draws links between classes and interfaces within the same diagram (for example, if a class extends another class or implements an interface). However, if the diagram contains packages, the dependencies between those packages are not drawn automatically.

The Update Package Dependencies command on the Diagram right-click menu builds links between the packages. A link between two packages means that there is some sort of dependency such as implementation, inheritance, or usage. Only real dependencies are taken into account. For example, if a certain class is imported but never actually used, this is not considered a dependency.

To draw dependency links:

1. Select one or more packages in the Diagram pane.
2. Choose **Update Package Dependencies** on the diagram right-click menu.

IMPORTANT: Together builds all the links between the selected packages and the rest of the diagram. If no particular package is selected, all packages are considered selected. In this case, the command provides two options: *Current Diagram* (self-explanatory) and *All* (arm yourself with patience - this builds links for the entire project).

Once built, the links on a diagram are not kept up-to-date. Obsolete or even non-existent dependencies continue displaying until they are deleted or created anew. To update a link individually, select it and choose **Rescan** from the link's right-click menu.

The process of building dependency links is quite time-consuming. However, its speed is configurable. The flag `option.dependencyLinks.fastSearch` in `$TGH$\\config\\diagram.config` controls recursion into subpackages. Setting this flag to `true` skips subpackages and provides faster operation, while `false` generates the links between subpackages.

Filtering out autodependency links

If a diagram contains autodependency links between the classes, in addition to other links, it can become too entangled. Together provides a special filter that allows users to hide the autodependency links, if necessary

This is `filter.extradependency` in the `%TOGETHER_HOME%/config/filter.config` file. By default the filter is on and hides all autodependency links if there are any other links between the classes.

This features does not have any controls in UI. To change the behavior, edit `filter.config` file.

Working with node elements and links

Once you place elements into a diagram you may have to manipulate them. If you experiment a little, this will become intuitive and easy to do. Most manipulations involve dragging with your pointing device or executing right-click menu commands of the selected elements.

Selecting elements

Click on any element in the diagram to select it.

- Select multiple elements by holding down the **Ctrl** key and clicking on them individually, or click on the background and drag a “rubber-band box” around an area to select all the elements it contains.
- For elements containing members, click on a member to select it.
- Select a node for a diagram element in the Model tab of the Explorer and double-click to select it in the diagram. Scroll the view to make it visible.

Right-click menus

Many operations that you perform with elements or the diagram as a whole can be executed from the various right-click menus. Some of the operations include:

- Add or delete members
- Set association cardinality
- Cut, copy, and paste attributes, operations, or text
- Hide individual elements or show hidden elements

Right-click on diagram elements, including class members for access to element-specific operations on the respective right-click menu. Right-click the background to access the diagram's right-click menu.

NOTE: The right-click menu of a selected element is duplicated in the main Selection menu.

TIP: When right-clicking on an element in a diagram, you may accidentally lose the focus and get the right-click menu of some internal element instead. To avoid this situation, use Shift+Right-click. This invokes the right-click menu of the desired element and preserves the current selection.

Moving elements and drag-drop copying

Visually move Node elements into different packages, or members into different classes, by dragging them on top of the destination icon and dropping them. For example, you can drag Classes into a Package (Class diagram), or Components into a Node (Deployment diagram).

Use the same technique to copy members to different classes and interfaces by pressing CTRL before initiating the drag. When you move an element, it is *deleted* from the

source package or element and moved into the destination package or element. *Copied* elements are *not deleted* from their source.

When you drag Nodes or members around on a class diagram, the appearance of other element icons changes when the dragged element crosses their boundaries. This indicates that the icon being “crossed” has received focus and is a potential drop target. For Class, Interface, and Package elements, the drop focus is represented by a blue rectangle around the icon border. Members are highlighted in blue when focused.

If you drag an element outside the borders of the diagram, the diagram automatically scrolls to follow the dragging.

Full drag-and-drop support

Drag-and-drop support can be extended beyond the borders of the single diagram. This feature is set in the Options dialog - *General: Enable drag and drop*. When this option is checked, drag classes and members between the Explorer's Model tab and the diagram, or between two diagram tabs. It is even possible to move elements within the Model tab from one node to another.

Copying and cloning elements

Copy and paste Node elements within the same diagram or between different diagrams. Also copy and paste members within the same class or between different classes, and paste a class into another to create an inner class.

- To copy an element, select it and choose **Copy** on the right-click menu or **Edit | Copy** on the main menu.
- To paste an element, select its destination (element or diagram background) and choose **Paste** on the destination's right-click menu or **Edit | Paste** on the main menu.

NOTE: Cut command works the same way as Copy except the source item is deleted once the Paste operation is complete.

An element that can be cut/copied and pasted can also be *cloned* by using the Clone command on its right-click menu. Cloning is basically a one-step copy-and-paste.

You can perform cut, copy, paste, and clone operations from the Explorer using the appropriate right-click menu command for an element selected there.

Resizing node elements

To resize a node element:

1. Click on the element.
2. Drag the corner of the selected diagram element to the desired size. The element grows or shrinks in the direction of the cursor.

TIP: If you hold down the Shift key while dragging the corner of a selected element, the element grows or shrinks uniformly in all directions.

Manually resized Node icons shift to an automatically optimized size when their contents change such as when members are added or deleted.

To restore the default size:

1. Choose **Layout** on the diagram right-click menu.
2. Choose **All** on the submenu.

Converting bidirectional links

An association link between two nodes can be converted to a bidirectional link. This is done using patterns.

To make a link bidirectional:

1. Select the association link to be converted.
2. Choose the **Choose Pattern** command on the right-click menu of the link.
3. In the Choose Pattern dialog box, select the **Convert to Bidirectional** pattern, adjust members if necessary, and click **Finish** to complete the operation. This adds the *Split up* checkbox to the list of pattern parameters.

If there are two nodes on a diagram associated that are with each other, both associations can be coupled into a single bidirectional link.

To combine two associations in a single bidirectional link:

1. Hold down the Ctrl key and click on both associations to select them.
2. Choose the **Choose Pattern** command on the right-click menu of the selection.
3. In the Choose Pattern dialog box, select the **Convert to Bidirectional** pattern, adjust members if necessary, and click **Finish** to complete the operation. This adds the *Split up* checkbox to the list of pattern parameters.

All common operations are applicable to bidirectional links as well: you can move, reroute, or delete them as required. If a bidirectional link is not needed, revert to two single associations.

To split one bidirectional link into two separate associations:

1. Select the bidirectional link.
2. Choose the **Choose Pattern** command on the right-click menu of the selection.
3. Check the *Split up* box and click **Finish**. The link breaks apart into two associations.

NOTE: If one (or both) of the associated classes is read-only, the Bidirectional Link pattern can not be applied.

Labeling links

For a more understandable diagram, add textual labels to the links. Three labels can be associated with a single link: the link label itself, a client label, and a supplier label.

To add a link label:

1. Choose **Rename** on the right-click menu of the link.
2. Choose the **Label** command from the submenu to open the in-place editor for the link label.
3. Enter the link description and press **Enter**.

To add a client/supplier role label:

1. Choose **Rename** on the right-click menu of the link.
2. Choose the **Client Role (Supplier Role)** command from the submenu to open the in-place editor for the role.
3. Enter the role description and press **Enter**.

If the description is too long, the text wraps to the next line. This facility is controlled by the *Wrap link labels text* option in the Options dialog (*Diagram - Wrap link label text*). When this option is checked, and the label exceeds a pre-defined length in pixels, the label text wraps and displays in multiple lines (the maximum length is also specified in the same place in the Options dialog).

To enable label text wrapping:

1. Choose **Diagram Options** on the right-click menu of the diagram.
2. Select the *Diagram* node.
3. Check the *Wrap link label text* box and close the dialog.

Labels can be oriented along the links. This behavior is controlled by the option *Show labels oriented along links* on the Diagram node of the Options dialog (*Diagram - Show labels oriented along links*). However, oriented labels and multi-line labels are mutually exclusive. If oriented labels are selected, word wrapping is disabled.

Rerouting links

Once the links are created, change the client or supplier nodes of a link, add or remove bending points, and reverse link direction.

To change the link routing:

1. Select the link.
2. Pick the source or destination point of the link.

TIP: If the required end of the link is out of reach, choose **Scroll to Source** or **Scroll to Destination** on the right-click menu of the link.

3. Drag either the source or destination end point of the link to a new Node element.

TIP: If the drop destination is out of range, drop the link end point on the diagram background and select the target from the dialog that appears.

To add bending points to a link:

1. Select the link.
2. Pick a point on the link and drag.
3. Drop in the required place on the diagram background.

To remove bending points:

1. Select the client or supplier node element.
2. Choose **Layout** on the right-click menu.
3. Choose the **Route Links** command on the submenu.

To reverse a link:

1. Select the link.
2. Choose **Reverse Link Direction** on the right-click menu.

Viewing diagrams and managing diagram elements

Together makes it easy to manage simple or complex diagrams with automated layout features that optimize the diagram layout for viewing or printing. You can also create your own diagram layouts.

Viewing

Scroll the Diagram pane horizontally and vertically using its scrollbars or the Overview icon in the lower right corner of the Diagram pane.

- Resize the pane vertically by dragging its lower edge up or down.
- Resize the pane horizontally by dragging the separator located between the Toolbar and the Explorer pane or by resizing the window.
- Enlarge your work area in the Diagram pane by hiding any or all of the other panes using the View menu or main toolbar buttons.
- Click the Overview button (or choose **View | Overview**) to display the overview of the current diagram. Resize and move the shadow area to obtain the desired view.

Zooming

There are several alternative ways to obtain the required magnification on the Diagram pane.

- Use the Zoom lens button on the Diagram toolbar:
 - Click the **Zoom** lens button and click on the Diagram pane to zoom in.
 - Hold down the **Alt** key and click on the Diagram pane to zoom out.
- Choose **View | Zoom** on the main menu and choose a command from the submenu.
- Use keyboard shortcuts. Default settings use numeric keypad keys:
 - NumPad+ for Zoom in
 - NumPad- for Zoom out
 - NumPad* for Fit in Window
 - NumPad/ for Fit 1:1

Using the automated layout features

The Diagram right-click menu provides access to the automated layout optimization features with the following commands.

IMPORTANT: Note that the contents of the Layout node depends on the current selection.

If you click on the background:

- **Layout | All** positions all diagram elements automatically according to the Layout options settings.
- **Layout | All for Printing** positions elements within page borders. You can set Print options to display the print grid on your output.

If you select one or more diagram elements:

- **Layout | Selected** and **Layout | Selected for Printing** repositions only selected elements.
- **Layout | Route Links** streamlines the links.
- **Layout | Actual Size** optimizes the node size.

Creating your own manual layout

Create your own layout by selecting and moving single or multiple diagram elements. You can:

- Select a single element and drag it to a new position.
- Select multiple nodes and drag them as a group to a new position.
- Select multiple nodes, cut them as a group, and paste them to a new position.
- Manually reroute links.

To learn more about these tasks, see [Drawing Diagram Elements: Tips and Tricks](#).

Tips and tricks

- Manual layouts are saved when you close a diagram or project and restored when you next open it.
- Manual layouts are *not* preserved when you run one of the auto-layout commands.
- You can revert to your manual layout after an auto-layout operation using Undo. For example, you might invoke Layout All for Printing, print the diagram, then call Undo to restore your manual layout.

Editing diagrams

After opening a project that has diagrams, open one or more diagrams for editing. Each diagram opens in its own tabbed page in the Diagram pane. Tabs display the diagram name and diagram type icon. (The icons correspond to those shown in the Model tab of the Explorer.) Switch between open diagrams by clicking the desired tab.

Opening and closing diagrams

Diagrams are opened from the Model tab of the Explorer using the right-click menu commands.

To open a diagram:

1. Navigate to the diagram you want to open.
2. On the right-click menu of the diagram, choose **Open** (this replaces the contents of the current diagram) or **Open in New Tab** (if you want to work with several opened diagrams).

TIP: There are alternative ways to open diagrams.

- Double-click on the diagram node. This opens the diagram in the currently focused diagram page, replacing any open diagram.
- On the right-click menu of a package, choose **Open Diagram** or **Open in New Tab**.

Diagrams are closed using the commands on the main menu or diagram right-click menu.

To close the current diagram:

1. Right click on the diagram tab.
2. Choose **Close**.

TIP: There are alternative ways to close the current diagrams:

- Use one of the hot keys Ctrl+F4 or Ctrl+W.
- Choose **File | Close** on the main menu.

To close all opened diagrams:

1. Right click on the diagram tab.
2. Choose **Close All**.

Editing properties

When selecting an element in a diagram or the diagram background, the properties of the element (or diagram) are available in the properties Inspector. The contents of Inspectors vary depending on what is selected. If an element is selected in the Diagram pane, its properties are loaded in the appropriate Inspector. (If nothing is selected, the background is selected by default and diagram properties are loaded.)

Generally an Inspector presents several tabbed pages representing different categories of properties. For example, the Class Inspector has the following tabs:

Properties: General properties of the class (name, stereotype, abstract, and so on) This tab displays two columns:

- **Name** shows the name of each property
- **Value** displays the value or setting of the property named in the same row of the Name column.

Hyperlinks: Intra-project and URL hyperlinks to related information (see [Hyperlinking diagrams](#)).

View: Visual properties of the diagram icon (such as color).

Description: Source code comments.

Javadoc: Properties used for Javadoc generation (such as author and version).

HTMLdoc: Source code comments in HTML format.

Req: Information tracking requirements (such as number, priority, and difficulty).

Beans: Optional JavaBean properties (if the class is a JavaBean).

Stereotype: The *stereotype* property typically has a pick-list but these are not always populated by default. If you want to modify an Inspector to add or modify stereotypes, it is possible to do so with a little Java programming. See [Advanced Customization: Customizing Properties Inspectors](#).

In general, you can enter a value for string type properties by typing a value in the edit field of the *Value* column. Some string type properties display a “browse” button next to the field to indicate that an editor dialog is available.

- Multi-line string properties display a multi-line text editor dialog.
- String properties whose value must be the name of some object or element in the project display the file chooser button.

TIP: Use the keyboard shortcut Alt+Insert to edit the fields with file chooser buttons.

- Some properties can have multiple values such as *parameter* of an operation. Here, the editor accepts a comma-delimited string.
- Some properties have a pick-list of available values. You can select from the list or enter your own value in such fields.

Opening the Inspector

There are several ways to open the Inspector:

- Choose **Properties** on the right-click menu.
- Choose **Selection | Properties** on the main menu.
- Use the keyboard shortcut Alt + Enter.

Modifying properties

To modify a property value, click it in the Inspector. Some properties are not modifiable with the Inspector and are therefore disabled for editing. Note also that some diagram elements, such as compiled classes, are read-only and therefore all their properties are disabled for editing.

Press **Enter** to complete the operation, or exit the edited field. Pressing **Esc** closes the Inspector.

Applying changes

Changes to property fields are applied when you:

- Press **Enter** (Inspector dialog remains open).
- Close the Inspector (Ctrl+Enter, Alt+F4, or click **x** button).
- Exit the edited field.
- Lose the focus on the open Inspector dialog.

IMPORTANT: If you click the dialog's close button while changes are pending to a field, *the changes are not saved*.

In-place editing

Working with diagrams encourages in-place editing of the properties, in addition to editing in the Inspector. Each diagram element has an initial string with a certain set of properties. However, some of these properties are not displayed and become available only during the course of in-place editing. You can modify all properties or just some of them so that the underlying engine completes the changes.

There are several ways to enable in-place editing:

- Double-click the selected element.
- Choose **Rename** on the element's right-click menu.
- Press **F2** for the selected element.

This opens a highlighted text string with a cursor that allows modification unless the diagram is read-only. The changes are applied by pressing Enter or clicking on another element, which generates the relevant code.

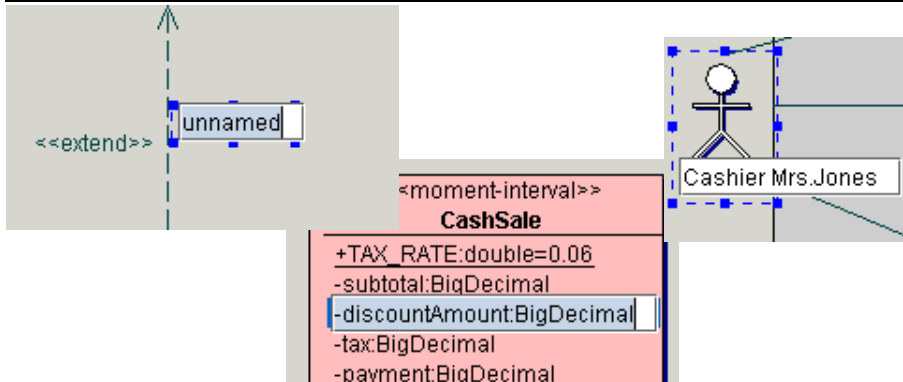
When editing properties manually, adhere to correct syntax. This specifically applies to code-generating diagrams. Together responds to the wrong syntax of modifiers with error messages. But beware - a misspelled attribute type is reproduced exactly in forward engineering.

In case of an incomplete entry, the omitted visibility modifiers, attribute types, or return types of the operations are replaced with their existing values. For example, if an attribute was private and the visibility is not entered, then it will be kept private. If no value was specified before, then default values are substituted as they are specified in the appropriate template properties. For example, an attribute with undefined visibility modifier and type will be a *private integer* if an operation's return type is not specified, it defaults to *void*.

Almost all strings can be modified at any time.

NOTE: Shapetypes and link types are not editable.

Figure 26 In-place editing of a link comment, member, and actor



Hyperlinking diagrams

This section explains the types of hyperlinks that can be created with Together and their benefits. There are also step-by-step instructions for creating, viewing, removing, and browsing hyperlinks.

Create hyperlinks from your diagrams to other system artifacts and browse directly to them. Create hyperlinks from the current diagram as a whole or from a selected diagram element (or group of elements) to:

- A new diagram.
- An existing diagram or diagram element anywhere in the project.
- A document file on a local or remote storage device.
- A URL on your company's intranet or on the Internet.

Create, view, remove, and browse hyperlinks with the Hyperlink Tab right-click menu.

Add or remove hyperlinks using the right-click menus of the link-type nodes of the Hyperlinks tab.

Why use hyperlinking?

- Link diagrams that are generalities or overviews to specifics and details.
- Create browse sequences leading through different but related views in a specific order; create hierarchical browse sequences.
- Link descendant classes to ancestors; browse hierarchies.
- Link diagrams or elements to standards or reference documents or generated documentation.
- Facilitate collaboration among team members.

How to create hyperlinks

This section explains how to create the various kinds of hyperlinks. Hyperlinks exist within the context of projects, so open a project to create or browse them. Hyperlinks are essentially *properties* which is why they appear in the properties Inspector.

Hyperlinking to a new diagram

Create a hyperlink from an existing diagram or one of its elements to a new diagram that you create as part of the hyperlinking task. The main procedural difference in creating a new diagram this way is launching the New Diagram dialog from the Hyperlink tab instead of from the main menu. Of course, the new diagram is then hyperlinked to your originating element by default.

To create a new hyperlinked diagram:

1. Open an existing diagram from which to create the hyperlink (or create a new diagram).
2. Select the element or group of elements to which you want to link the new diagram. If you want to link to the diagram as a whole, click on the diagram background to deselect all elements.
3. Choose **Properties** on the right-click menu and select the Hyperlinks tab in the Inspector.
4. Right-click on the Element node to display the right-click menu.
5. Choose **New Diagram** to display the New Diagram dialog.
6. Specify the new diagram's type and location as described in [Creating Diagrams in Projects](#) and click **OK**.

The new diagram opens in the Diagram pane and the Hyperlinks tab displays the link to the originating diagram or element.

Hyperlinking to an existing diagram or diagram element

Create a hyperlink from an existing diagram or one of its elements to any other diagram or diagram element in the project.

To create a hyperlink to an existing diagram or element:

1. Open an existing diagram from which to create the hyperlink (or create a new diagram).
2. Select the element or group of elements that you want to link to another diagram or element. To link to the diagram as a whole, click on the diagram background to deselect all elements.
3. Choose **Properties** on the right-click menu and select the Hyperlinks tab in the Inspector.
4. Right-click on the Element node to display the right-click menu.
5. To link to a diagram or a diagram element, choose **Existing Element**.
6. Select the desired diagram or element in the dialog that is now displayed. For element selection, expand diagram nodes in the selection dialog's tree-view.
7. Click **OK** to close the dialog and create the link.

Hyperlinking to a URL or file

Create hyperlinks from your UML diagrams to any online resource anywhere on the planet. For most users, such hyperlinking will probably take the form of documents on a LAN or document server, or URLs on the company intranet. But you can just as easily link to online information from the OMG, newsgroups, or discussion forums - if it is available on line, you can link to it.

To create a hyperlink to a file or URL:

1. Open an existing diagram or create a new diagram.
2. Select the element or group of elements to link to another diagram or element. To link to the diagram as a whole, click on the diagram background to deselect all elements.
3. Choose **Properties** on the right-click menu and select the Hyperlinks tab in the Inspector.
4. Right-click on the node named URL to display the URL dialog.
5. If linking to a file, enter the complete path or browse for it. If linking to a URL, enter the URL.
6. Click **OK** to create the link.

Viewing hyperlinks

All the hyperlinks defined for any diagram or element display in the Hyperlinks tab of the properties Inspector. Select the Hyperlinks tab before checking diagrams for defined hyperlinks.

To view hyperlinks to a diagram, click on the diagram background and then view the Hyperlinks tab for defined links.

On a diagram, all names of diagram elements that are hyperlinked display in **blue font**. To view what is hyperlinked to the element, click on the element (node or relationship link) and then view the Hyperlinks tab.

Browsing hyperlinked resources

Once you find the defined hyperlinks for a selected diagram or element, use the Hyperlink tab's right-click menus to browse to the linked resources.

- Browsing to a linked **diagram** opens it in the Diagram pane or makes it the current diagram if already open.
- Browsing to a linked **element** causes its parent diagram to open or become current, and the diagram scrolls to the linked element and selects it.
- Browsing to a linked **file** launches the application registered in the system for the file type and loads the file.
- Browsing to a linked **URL** launches the system's default Web browser and loads the URL.

To browse a hyperlink:

1. Right-click on the desired hyperlink in the Hyperlink tab.
2. Choose **Open** on the right-click menu.

Removing hyperlinks

Remove hyperlinks as easily as you create them. Removal can be done from either “end” of a diagram or element hyperlink. That is, if you created a hyperlink from the diagram *Foo* to the diagram *Bar*, you can remove the hyperlink from either *Foo* or *Bar*.

To remove a hyperlink:

1. Open a diagram that displays the link you want to remove.
2. Choose **Properties** on the diagram right-click menu.
3. Right-click on the link in the Hyperlinks tab.
4. Choose **Remove** from the right-click menu.

Annotating diagrams

The Diagram Elements toolbar always displays Note and Note Link buttons that place Notes and Note Links on the diagram. Notes can be free floating, or you can draw a Note Link to some other element to show that a Note pertains specifically to it.

Using Notes

- Paste text from the clipboard into a Note when its in-place editor is active.
- Note text wraps when you resize the Note smaller.
- Edit a Note's properties using its Inspector (Alt+Enter).

In the Note Inspector you can:

- Edit the text as text, inserting HTML tags on the Text tab if desired.
- Change the text-only property on the Properties tab.
- View the Note text as HTML on the HTMLdoc tab.
- Change the foreground and background colors, and control 3D appearance on the View tab.
- Track various requirements properties including stereotype, priority, and difficulty on the Requirements tab.
- Add custom properties in the Custom Properties tab.

The Note Link

Notes are automatically included when you generate HTML documentation. The text of Notes linked to Class diagram elements does not appear in the source code. Use the

Description tab of the Class Inspector to enter the class description, and enter source code comments directly in your code using the Editor pane.

The Note Link Inspector evoked by selecting Properties from the Note Link right-click menu, is similar to the Note Inspector but for the Text tab, which is replaced by the Link tab. In the *Link* tab you can view both sides of the link - client and supplier.

Inspector documentation tabs

The *Description* tab in element Inspectors displays the description of the selected element (for example, class description) for source-generating diagram elements. When editing the text in this tab, source code updates when you press **Alt+Enter** or return to the diagram.

Use this tab to create and edit comments for elements that do not generate code (including non-class diagrams). These comments are stored with other diagram-specific information by Together.

Use the Javadoc tab to enter the values of Javadoc tags for the source code comments.

In addition to the tabs for viewing and editing comments as plain text, the *HTMLdoc* tab displays comments in browser mode.

Standalone Design Elements

Together enables you to work with diagrams and diagram elements in a really flexible way. To keep projects clean and organized, the design elements are stored inside the diagram packages. However, this makes it difficult to share elements between team members or store the elements in a source code control system.

To handle these problems, Together enables storing design elements in separate files. Such design elements are called Standalone Design Elements (SDE). Physically, they are stored in files with the *.ef<shapetype>* extension. For example, a standalone actor can be stored in a file named *Actor1.efActor*.

Using standalone design elements simplifies visual analysis and facilitates exchange of design elements among team members through the version control system, email, and so on.

IMPORTANT: It is worth mentioning that a design element is any element that has no underlying source code such as an Actor, an Object, a SwimLane, or a Note, but not a Class or a Package.

Creating SDEs

To create SDEs:

1. On the main menu, choose **Tools | Options | <level>**.
2. In the *General* options group, check the box *Create design elements as standalone*.
3. When selecting this option, any new design element created on a diagram is added as a shortcut, and the appropriate *.ef<shapetype>* file is written to the physical package of this diagram.

NOTE: On the Directory tab of the Explorer, the icons for the standalone design elements are different than the icons for the ordinary design elements.

There is an alternative way to create SDEs when this option is unchecked. Create a design element on a diagram, copy it, and add it to the model as an SDE.

1. In the Diagram pane, copy an element.
2. On the Model tab of the Explorer, select the destination package and right-click.
3. Choose **Paste as Standalone** on the package right-click menu.

Using SDEs

As mentioned above, with the *Create new design element as standalone* option selected, each subsequent design element added to a diagram is created as a separate file in the diagram package. Observe all different files on the Directory tab of the Explorer, such as Actor1.efActor, Actor2.efActor, Actor3.efActor.

If this option is unchecked, and you still want to make use of an existing or imported standalone design element perform the following steps:

1. Select the standalone element in the Model tree-view.
2. Choose **Copy** from the node right-click menu.
3. Right-click on the target diagram and choose **Paste** or **Paste Shortcut**.

Saving and copying diagram images

Save or copy entire diagrams or their selected parts for further reuse. There are three possible behaviors in Together:

Copy - Paste within Together

1. Select the required part of a diagram.
2. Copy it by choosing **Edit | Copy** on the main menu or the selection's right-click menu.
3. **Paste** the selection to the target location.

NOTE: If link labels are not displayed after pasting, press F5 with the focus on the Diagram pane to refresh the diagram.

Copy Image

In the Windows environment, copy a diagram to the clipboard and then paste the clipboard content to an external application.

Opt to copy images in bitmap or WMF format. To choose the desired format, check or uncheck the option *Copy diagram to clipboard as bitmap* in the *General* options. Bitmap format cures several problems with WMF format such as distorted fonts, incorrect conversion of localized fonts, and lack of Java 2D functions support. Besides

that, some rasterized graphics applications do not handle clipboard metafiles, but recognize bitmaps.

NOTE: When a diagram is copied in WMF format, labels of non-horizontal links are not reproduced in the target application.

TIP: Pasting bitmap images to MS PowerPoint produces “invisible slides.” To avoid this, use the Paste Special command, rather than Paste. However, if you copy a diagram in WMF format, it can be reproduced in PowerPoint using the Paste command.

Save Image

Once created, a diagram can be saved on the hard disk for further use. Together offers a choice between WMF, GIF, and SVG formats. To save the current diagram, choose the **Save Image** command on the File menu, and select the required target format from the list. This opens the File Chooser for the selected format.

The diagram image is stored on the disk and can be imported to the applications that recognize the selected format.

Printing diagrams and source code

Print both diagrams and text. To print individual diagrams, open them in the Diagram pane. To print the source code, open it in the Editor pane.

Setting Print options

Set printing options at different levels in the Options dialog.

To check Together's Print options for the specified level:

1. Choose **Tools | Options | <level>** on the main menu to display the Options dialog for the default, project, or diagram level.
2. Select the Print tab and check the settings.

For further information, see Chapter 2, “Setting Your Personal Preferences” on page 71.

How to print diagrams

Print any diagram separately, as a group, or all diagrams in the project.

To print a single diagram:

1. Open the diagram in the Diagram pane (the project must be open).
2. If there are several open diagrams, click the tab of the one you want to print.
3. Choose **File | Print Diagram** on the main menu to open the Print Diagram dialog.
4. Click **OK**.

To print several diagrams at once:

1. Open the diagrams in the Diagram pane (the project must be open).
2. Click on the workspace of any open diagram.
3. Choose **File | Print Diagram** on the main menu to open the Print Diagram dialog.
4. Select the *All Open* option and click **OK**.

How to print text

To print source code:

1. Make sure the Editor pane is visible (check the Editor Pane checkbox in the View menu, or press **F9**).
2. Click on the desired element of a diagram to display its source code in the Editor pane, or just type your text in the active tab of the Editor pane.
3. With the focus on the Editor pane, choose **File | Print File** on the main menu to open the Print File dialog.
4. Make the necessary settings and click **OK**.

Using auto-layout for printing

Together has automated layout optimization for printing diagrams. Auto-layout for printing ensures that all diagram elements fall within page borders defined by page size in Print Options.

Invoke print auto-layout immediately before printing a diagram. Right-click on the diagram background, then choose **Layout | All for Printing**.

TIP: To preserve a diagram layout created manually, invoke Undo after running auto-layout and print the diagram.

Printing generated documentation

- After generating HTML documentation, open it in your Web browser and print from there.
- After generating documentation in another format, open it in an application that reads the file format and print from there.

Troubleshooting

Printing problems (such as distorted diagram images) can stem from a number of things unrelated to a particular application program such as Together.

If you are working on a UNIX system, check the Known Problems section in the `readme.html` file in the root of your Together installation. There are specific printing issues for Together on some UNIX systems.

In Windows, Together provides a workaround for printing via the property: `print.dpi=72` in the `$TOGETHER_HOME$/config/misc.config` file. This property prevents problems in most cases.

If you experience printing problems, try one of the following:

- Adjust the value of the `print.dpi` property.
- Comment out the `print.dpi` property.

Then try printing again.

For each test:

1. Modify and save the `misc.config` file.
2. Open the Options dialog (**Tools | Options | <level>**).
3. Click the **Synchronize with External Changes** button (assuming Together is running).
4. Invoke **File | Print Diagram**.

Tips and tricks

- If you are running Together under Windows, set your printer's True Type Fonts property to *Print As Graphics* in the Font tab of the Windows Printers dialog before printing diagrams.
- To make sure that all diagram elements fall within page boundaries, run the auto-layout command **Layout All for Printing** from the Diagram right-click menu before printing a diagram. If you have created a manual layout, restore it using the **Undo** command after printing. Be sure to run **Undo** *before closing the diagram* or your manual layout will be lost.

CHAPTER 10

UML MODELING

Together provides support for the most frequently needed diagrams and notations defined by the Unified Modeling Language (UML). This chapter includes the following topics:

- “Notation and stereotypes of UML diagrams” on page 150
- “Use case diagrams” on page 151
- “Class Diagrams” on page 154
- “Sequence and Collaboration Diagrams” on page 162
- “Statechart Diagrams” on page 173
- “Activity Diagrams” on page 177
- “Component Diagrams” on page 179
- “Deployment Diagrams” on page 182

Notation and stereotypes of UML diagrams

Together supports both notation and stereotypes of UML diagrams:

- **Notation**

Together enables you to render UML diagrams using UML-compliant notation. In class diagrams, you control the code generation of various notational elements through either global or diagram level configuration options.

- **Stereotypes**

Together supports the use of stereotypes. You can adhere to UML-defined stereotypes, or customize stereotypes based on your requirements. With Together, you can also add color to stereotypes although this practice is not currently specified in the UML.

Use case diagrams

Use case diagrams provide a way of describing the external view of a system and its interactions with the outside world. *Actors* are a representation of the outside world, and can represent people or computer systems. A *use case* is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged within the system, and one or more outside interactors (the actors), along with actions performed by the system. A *use case diagram* depicts the relationship between actors and use cases within a system, as shown in Figure 27.

Figure 27 Use case diagram

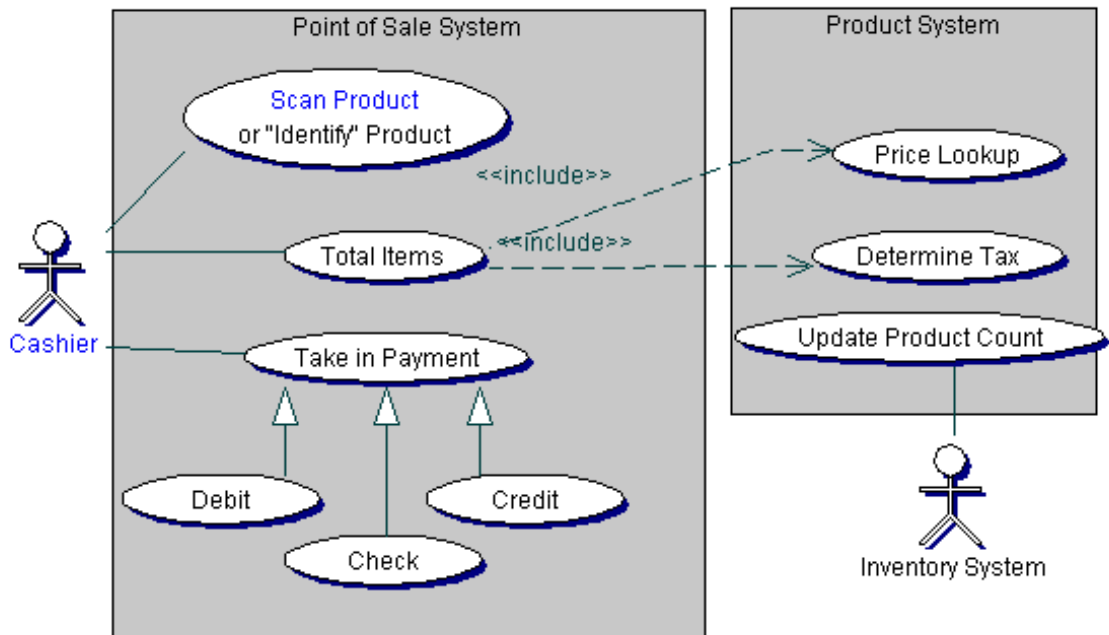


Diagram elements

Table 14 lists the contents of use case diagrams as defined by the UML. In Together, the elements reside on the use case diagram toolbar. Use tool tips to identify the different elements on the toolbar.

NOTE: Relationship links appear as *italicized* text.

Table 14 Contents of use case diagrams

Icon	Element name
	Actor
	Use case
	Note
	<i>Communicates</i>
	<i>Extends</i>
	<i>Includes</i>
	<i>Generalization</i>
	<i>System boundary</i>
	<i>Note link</i>

To learn how to create new diagrams in a project or to understand the techniques for placing elements and drawing links, see “Creating diagrams in projects” on page 124.

To create a new diagram with the same content as an existing diagram, use the Clone command on the context menu of the navigation pane node. The resulting new diagram has a unique name and is created in the same package.

To apply elements from an existing use case diagram to a new use case diagram, use the Add Shortcut command on the diagram’s context menu.

Creating browse-through sequences of use case diagrams

Use case diagrams typically represent the context of a system and system requirements. Usually, you begin at a high level and specify the main use cases of the system. Next, you determine the main system use cases at a more granular level. As an example, a “Conduct Business” use case may have another level of detail that includes use cases such as “Enter Customers” and “Enter Sales”. Once you have achieved the desired level

of granularity, it is useful to have a convenient method of expanding or contracting the use cases to grasp the scope and relationships of the system's use case views.

Together's hyperlinking feature allows you to create browse-through sequences comprised of any number of use case (or any other) diagrams. You can link entire diagrams at one level of detail to the next diagram up or down in a sequence of increasing granularity, or you can link from key use cases or actors to the next diagram. By browsing the hyperlink sequence, you can follow the relationships between the use case diagrams.

Together does not confine hyperlinking to such sequences, however. You can use hyperlinking to link diagrams and elements based on your requirements. For example, you can create a hierarchical browse-through sequence of use case diagrams, creating hyperlinks within the diagrams that follow a specific actor through all use cases that reference the actor.

To learn how to create and browse hyperlinks, see “Hyperlinking diagrams” on page 141.

Showing other diagrams in a use case diagram

A use case diagram can indicate its relationship to another diagram without using the hyperlinking feature described in the previous section. To accomplish this, place a shortcut to another diagram within the current diagram.

A diagram shortcut is represented with a *package* that displays the icon of the diagram type it represents and the diagram name. The package also lists the elements of the diagram. For more information regarding packages, see “Working with package diagrams” on page 157.

To create a shortcut to the current diagram, follow these steps:

1. In the Model tab, select the node for the diagram you want to import and choose **Copy** from the right-click menu.
2. Right-click on the diagram background and choose **Paste** from the right-click menu.

You can open the diagram represented by the shortcut using the right-click menu of its package icon.

Adding an extension point

An extension point refers to a location within a use case where you can insert action sequences from other use cases. An extension point consists of a unique name within a use case and a description of the location within the behavior of the use case.

In a use case diagram, extension points are listed in the use case with the heading **Extension points** (appears as bold text in Together).

To add an extension point to a use case diagram, right-click on an element of the diagram and choose **New Extension Point**.

Class Diagrams

Class diagrams show the static structure of a system. Specifically, this includes entities that exist (such as classes and types) and their internal structure. Class diagrams also depict collaborations and relationships between classes, and inheritance structures, as shown in Figure 28.

A class diagram can also show instances (or objects) and links between objects. Class diagrams define a vocabulary for the system and can become the basis for component and deployment diagrams. In addition, class diagrams form the basis of visualization, specification, documentation, and implementation of the structure of a system. (By implementation, for example, Together synchronizes class diagrams to source code).

To learn how to create new diagrams in a project, or to understand the techniques for placing elements and drawing links, see “Creating diagrams in projects” on page 124.

Figure 28 Class diagram

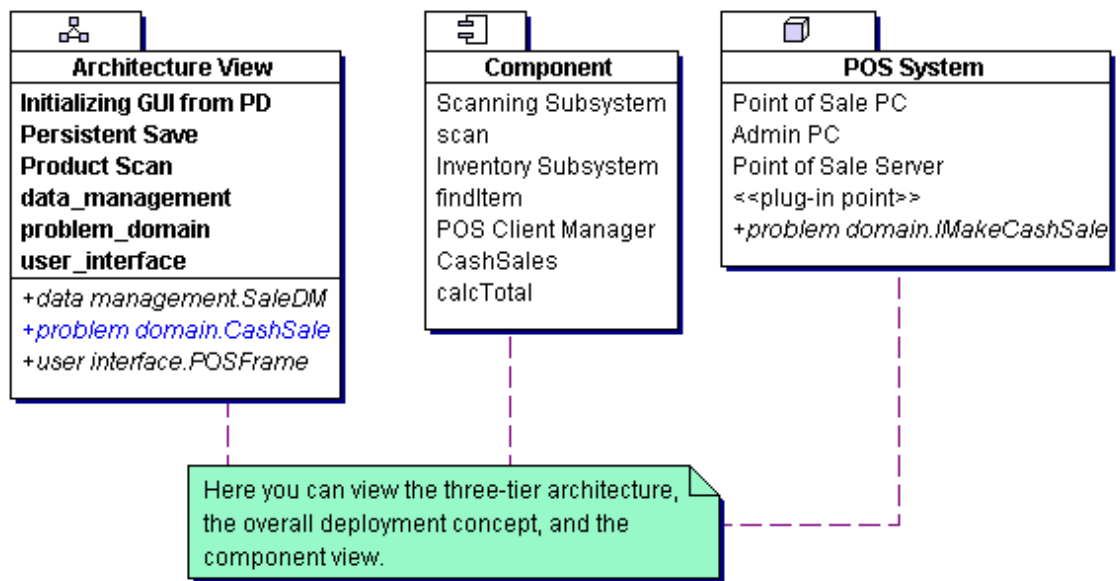


Diagram elements

The contents of a class diagram are language-dependent. Table 15 lists the contents of class diagrams as defined by the UML. Together provides these on the Class Diagram Toolbar. Use tool tips to identify the various elements on the toolbar.

NOTE: Relationship links appear as *italicized* text.

Table 15 Contents of class diagrams

Icon	Element
	Package
	Class
	Interface
	Class by pattern
	<i>Generalization/Implementation</i>
	<i>Association</i>
	<i>Link by pattern</i>
	<i>Dependency</i>
	Object
	Note
	<i>Note link</i>

In addition to these elements, class diagram can contain shortcuts to the elements from the different packages and diagrams.

Depending on the current project language, the class diagram toolbar displays icons specific for the selected language:

Table 16 Language specific elements

Language	Icon	Element
Java		Entity EJB
		Session EJB
		MessageDriven EJB
IDL		Struct
		Valuetype
		Exception
C++		Aggregation

Setting-up shortcuts

To refer to a class located outside the current diagram, or to another diagram, you can use shortcuts. To add a shortcut, right-click on the diagram background and choose **New | Shortcuts** from the right-click menu. This displays Selection Manager dialog, where you can choose the desired element from the Model, Search/Classpaths or Favorites nodes.

NOTE: You can not use the Copy and Paste operations for shortcuts on class diagrams. The Paste operation does not create a new shortcut element.

Showing Java Beans

To show classes as Java Beans on a class diagram, follow these steps:

1. Enable the bean support option.
2. From the Tools menu, choose **Options | Diagram Level** to invoke the Options dialog.
3. From the tree, choose **View Management | Java Beans / C++ Properties**

4. Check the *Recognize Java Beans* option.

Java Beans, including Enterprise JavaBeans (EJB), are displayed as classes with two additional sections, one for properties, and the other for events to which the Java Bean can respond. When you select the Beans support option, all the classes on the diagram are rescanned. Those classes with get and set methods are displayed as Java Bean nodes on the diagram denoted by a Bean symbol by the class name. Display support for Beans is controlled in Default and Diagram Options.

Working with package diagrams

Class diagrams that show packages are referred to as *package* diagrams. Specifically, package diagrams contain package icons for each physical subpackage under the project directory. Package diagrams are physically represented as text files with the extension `dfPackage`, located in relevant folders under the project root directory.

Viewing packages and their content

The Model tab of the Explorer displays the packages and subpackages of a project. You can use the Explorer to navigate within package diagrams and view their contents.

Opening packages

To open a package, choose one of the Open commands on the right-click menu.

Renaming a package

You can rename a package in place, or change the name in all source files by using the properties inspector (Right-click Menu | Properties).

Moving elements into packages

Classes and interfaces can be visually moved to other packages by dragging from the diagram to the target package. Other packages, such as imported classes, compiled classes, and interfaces cannot be moved in this manner. Pressing the CTRL key and dragging a class into a package imports the class into that package. The class name appears italicized in the package icon.

By default, the package icon on the diagram displays its updated contents. You can use the class/interface context menu and add attributes and operations directly. The diagram option Subpackages controls how the packages display in the diagram.

Moving nodes from packages

You can move a class or interface to a diagram, delete it from its current package and move it to its new package by dragging the class or interface name from the package element to the diagram background.

Showing project content from different packages

Packages, classes and interfaces that belong to your project display in the Model tab of the Explorer. Their respective paths are defined in your project properties (File | Project

Properties). If, when creating the project, you specify the class diagram as the default, then the project package contains a class diagram that maps out the packages belonging to the project. Depending on your specifications in the New Project dialog (and Project Properties), Together can reverse engineer the classes of these packages into class diagrams.

In such a case, the class diagram having the same name as its respective package shows the classes and interfaces in the package, along with relationships and dependencies. You have the option to add classes from any package in the project in these diagrams, or you can create new class diagrams in a package to show different views of the package contents and relationships to other packages. For example, you can create different class diagrams to show classes for components or subsystems contained in multiple packages.

To show packages, classes, or interfaces from another project package in a class diagram, follow these steps:

1. In the Model tab, navigate to the package containing the element(s) you want to show.
2. Select the desired element in the Explore and copy it to the Clipboard.
3. Paste the element in the desired location in the diagram.
4. After copying all elements, choose Update Package Dependencies on the diagram right-click menu.

Dependency links

When you choose the Update Package Links command on the diagram right-click menu, Together scans each package on the diagram. If it finds an element within the package that is linked to an element of another package, then it creates a Dependency link between those package icons on the diagram.

This applies only to source-code based elements. Inter-package dependencies between such elements as Objects, Notes, search-path imported classes, use cases, and so on are not recognized.

Deleting a Package

To delete a package using its context menu, you must first remove all of the contents of the package. A physical package cannot be deleted if it contains source or diagram files, files of non-Together types, or if other applications share the corresponding directory.

Showing classes on search paths

When you create a project you can define directories with any number of search paths whose content you may want to show in diagrams. For example, you can show entities that reside on the standard for full Java classpath. Such resources exist for the project but Together does not include them in the generated HTML documentation for the project.

To show classes etc. from one of the search paths in a class diagram, follow these steps:

1. Open or create a class diagram.

2. Right-click on the background and choose **New | Shortcuts** to launch the Add Shortcut dialog. This dialog displays the available content available for the diagram and all content residing outside the current package.
3. Navigate to the resource you want to add and click the **Add** button. Repeat until you have added all the resources you want.
4. Click **OK** to close the dialog.

TIP: If the resource you are looking for is not shown, it is probably not in the search paths defined in Project Properties. You can add resources to the search paths at any time by choosing File | Project Properties | Advanced and selecting the tab for search paths.

Defining inner classes

To create an inner class (when an inner class already exists), drag it over another class and drop it. If the inner class does not exist, you can create and define it within its parent at the same time.

To create and define a new inner class, follow these steps:

1. Select the parent class.
2. Choose **New | Inner Class** from the right-click menu.

TIP: When defining inner classes, consider the following:

- You can define new inner classes from the right-click menus of classes in the Explorer.
- You can use drag-drop to remove inner classes from classes in the diagram.

Importing other diagrams

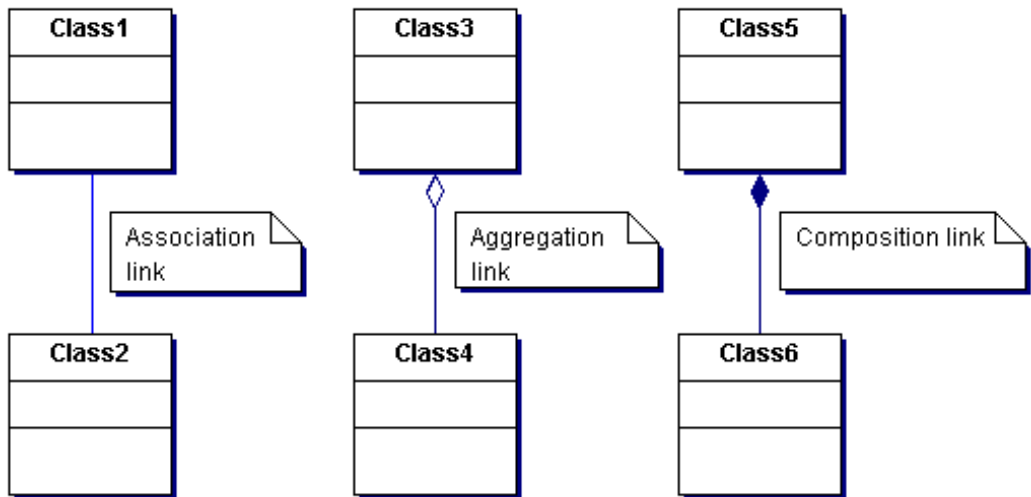
You can import another diagram of any type into a class diagram creating an automatic link to that diagram. A diagram that is imported into another diagram is shown as a package with a corresponding stereotype icon. You can open the diagram from the context menu of the package icon.

Showing Association, Aggregation, and Composition

To draw an association link, use the association icon  on the class diagram toolbar. Afterwards, you can right-click on the link and specify an association, aggregation, or composition link. You can also change the type of link by applying the appropriate pattern to the link. Figure 29 illustrates how Together depicts the various types of association links.

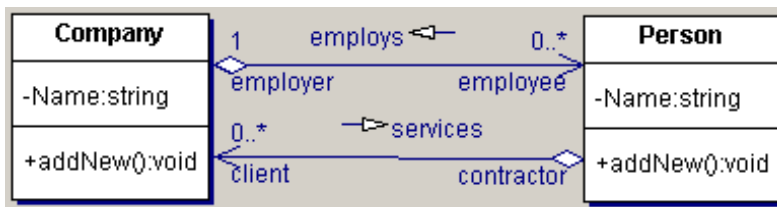
To show the direction of association, aggregation, and composition links, right-click on the link and select **Properties** to invoke the inspector. From the Link tab, set the *directed* property to Automatic, Directed, or Undirected.

Figure 29 Types of association links



You can also illustrate the semantic direction in the link label of the relationship links using the label direction property of the link. Semantic direction may be independent of the link navigation direction, as shown in Figure 30. In this example, two possible relationships exist between Company and Person. Note that the label direction and semantics depict that Company employs Person in the first instance, and Person services Company in the second case.

Figure 30 Semantic direction of the link label vs. link direction



Editing Members and Properties

This section includes instructions for adding and editing members and properties, including rearranging the order of attributes and operations, and setting compartment controls.

Adding and editing members

To add an attribute or an operation to a class or interface, follow these steps:

1. Right-click on the class or interface

2. Choose **New | Attribute or New | Operation**

3. The member is inserted after the last attribute or operation, respectively

If a class or interface already has attributes or operations, you can right-click on these directly to create the additional member using the existing member's right-click menu. The new member is inserted before the selected member.

You can also add members from the Explorer. Use the right-click menu of class or interface nodes.

TIP: To edit an attribute or an operation, use inplace editing or editing in the properties inspector.

Adding and editing properties

To add a property, right-click on the class and choose **New | Property**.

The location of the newly created property depends on whether or not the Recognize Java Beans options is on or off. If this option is selected, the property adds to the properties' compartment, and to the list of properties in the Beans tab of the Inspector. The class displays as a bean icon. If this options is off, the property adds to the attributes' section, and its accessor methods add to the operations' section.

When the option is off, you have to take special care when editing or deleting properties. When a new element is created, in-place editing is immediately available. However, if you edit property type in place, the relevant types in the accessor methods will not be synchronized. The same result occurs when a property is deleted; that is, the accessor methods stay in place and should be deleted individually. Thus, the safest method for editing properties is to use the Choose Pattern dialog.

To edit properties using the Choose Pattern dialog, follow these steps:

1. Right-click on the property to be edited and select Choose **Pattern**.
2. In the pattern tree-view select **Property pattern** and complete all necessary changes
3. Click **Finish** to apply the changes to all the components of a property.

Rearranging the order of Attributes and Operations

You can use drag and drop to reorder members within a class icon. Together simultaneously updates the source code with the new order.

Dropping one member on another member name positions the dropped member before the target member. Dropping a member on the class name moves it to the last position in the attribute or operation list respectively.

Setting compartment controls

You can also show an expansion/contraction control in the Attributes and Operations compartments of Class nodes. This is particularly useful when you have large classes with content that you do not need to view at all times. You can set this option at any configuration level (default, project, or diagram). From the Tools menu, choose **Options | Applicable_Level**, then select the Diagram node of the Options dialog and check *Show controls for compartments*.

Sequence and Collaboration Diagrams

Sequence and collaboration diagrams, sometimes called *interaction diagrams*, are two diagrams you can use to model the dynamic aspects of a system or subsystem. Both depict interactions consisting of a set of objects, their relationships, and the messages exchanged between them.

Collaboration diagrams emphasize the *structural organization* of objects, as illustrated by Figure 31. In contrast, sequence diagrams emphasize the *time ordering of messages*, as illustrated in Figure 32. Collaboration diagrams are essentially graphs; sequence diagrams are essentially tables with different objects and messages depicted across the X axis and increasing time down the Y axis. However, the two diagrams are *semantically equivalent*: one type can convert to the other type with no loss of information. Together enables you to do this conversion with a simple mouse click.

Although semantically equivalent, sequence and collaboration diagrams do not necessarily show the same information. For example, collaboration diagrams explicitly show how objects are linked, while in sequence diagrams the links are implied. Message return values show in sequence diagrams but not in collaboration diagrams.

Figure 31 Collaboration diagram

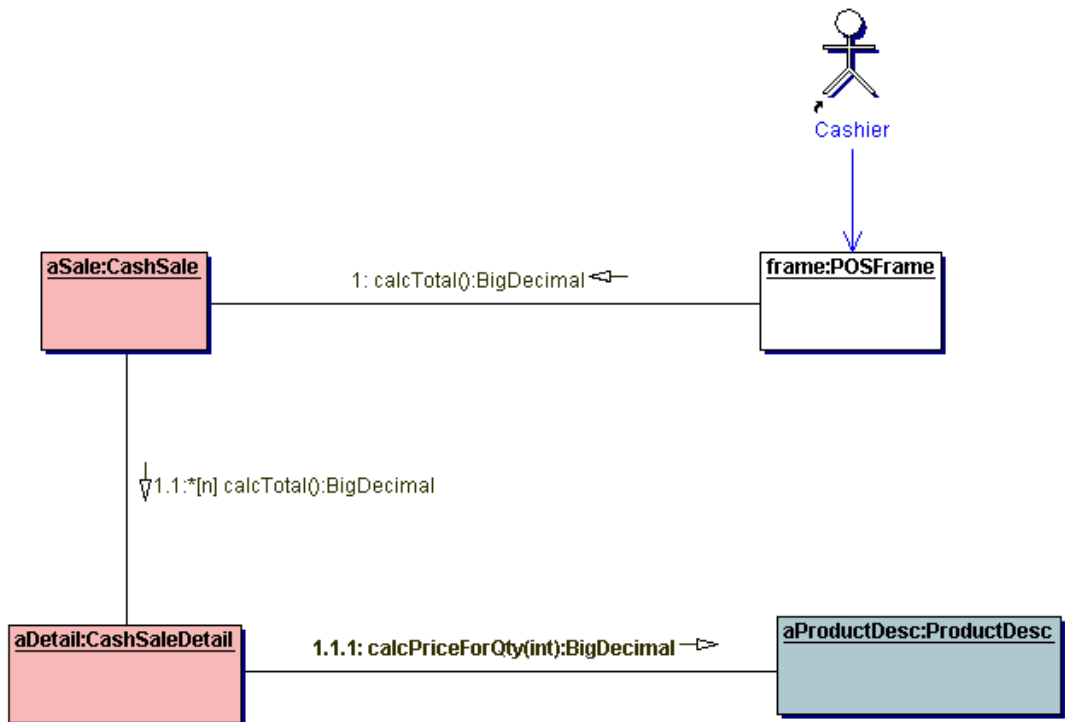


Figure 32 Sequence diagram

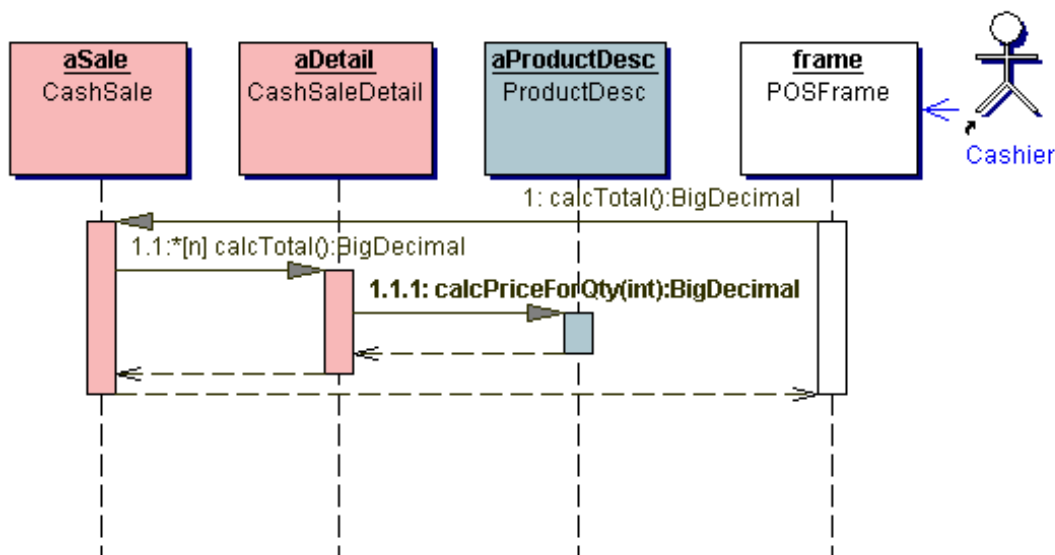


Diagram elements

Table 17 lists the contents of collaboration diagrams and Table 18 lists the contents of sequence diagrams. Both the contents of collaboration and sequence diagrams are defined by the UML.

NOTE: Relationship links appear as *italicized* text.

Table 17 Contents of collaboration diagrams

Icon	Element
	Object
	Actor
	<i>Message</i>
	<i>Self Message</i>
	<i>Associates</i>

Table 17 Contents of collaboration diagrams (continued)

Icon	Element
	<i>Aggregation</i>
	Note
	<i>Note link</i>

Table 18 Contents of sequence diagrams

Icon	Element
	Object
	Actor
	<i>Message</i>
	<i>Message with delivery time</i>
	<i>Self Message</i>
	<i>Statement Block</i>
	Process Delay Link
	Note
	<i>Note link</i>

Creating sequence and collaboration diagrams using the expert

NOTE: This section provides instructions for using the Generate Sequence Diagram Expert to create a sequence diagram. The expert is disabled for languages other than Java or C++.

To generate a sequence diagram from an operation, follow these steps:

1. Open the class diagram containing the class whose operation you want to model.
2. Locate the desired class and choose the desired operation.
3. Choose **Generate Sequence Diagram** from the Operations' right-click menu to display the Generate Sequence Diagram Expert. View a list of the packages and classes involved in the operation for which you are generating a sequence diagram.
4. In the Package/class list, select the packages and classes you want to display in the generated diagram. All packages and classes are selected by default. However, some Java things might not be relevant... `java.lang.Integer`, for example. You can increase the meaningfulness of the generated diagram by removing anything that doesn't really help explain the sequence of operations.
5. In the Package/class list, for those elements you decide to show in the diagram, check whether or not to show implementation detail in the generated diagram.
6. Click **OK** to generate the diagram and open it in the Diagram pane.

Together generates a new sequence diagram in the same package as the source class, with the same name as the source operation, and opens it in the Diagram pane.

Using the Options dialog to control a sequence diagram's generation

You can use Options dialog to check options for the generated diagrams or control their behavior. To accomplish this, select *Diagram Options* on the diagram's right-click menu. This command displays *View Management, Diagrams* and *Print* tabs of the Options dialog. Alternatively, invoke the Options dialog from the main menu (**Tools | Options**).

It is possible to generate sequence diagrams for several methods. You can display sequence diagram for each method in a separate tab, or to show sequence of calls for all selected methods in a single sequence diagram. To control this behavior, use the options *Create multiple diagrams* and *Show multiple diagrams* of the sequence diagram node in *View Management* tab of the Options dialog.

When the option *Create multiple diagrams* is checked, a separate diagram is generated for each selected method. Otherwise, sequence of calls for all methods is generated on a single diagram.

When the option *Show multiple diagrams* is checked, all generated diagrams should show up automatically after generation completed.

It is also possible to control nesting level. *Depth of call nesting* option allows to change nesting value according to the required degree of complexity.

Converting between sequence and collaboration diagrams

Together allows you to quickly convert between sequence and collaboration diagrams. However, when you create a new diagram, you must specify that it is either a sequence diagram or a collaboration diagram, and Together then tracks it as such. The diagram displays in the Explorer as the type of origin, and opens in that view. For example, if you create a sequence diagram, it will always show in the Explorer, and open in the Diagram pane as a sequence diagram. But you can view it as a collaboration diagram.

To convert between sequence and collaboration diagrams, follow these steps:

1. Right-click on the diagram background.
2. If the diagram is a sequence diagram, choose **Show as Collaboration** from the diagram right-click menu. If viewing a collaboration diagram, the menu command is **Show as Sequence**.
3. Repeat this process to switch back and forth.

NOTE: You can also do this switch from the right-click menus of interaction diagrams in the Explorer.

Refining sequence and collaboration diagrams

This section provides techniques for refining the sequence and collaboration diagrams that you have created.

Techniques for sequence and collaboration diagrams

This instructions in this section apply to both sequence and collaboration diagrams. For techniques specific to the type of diagram, see “Techniques specific to sequence diagrams” on page 167 or “Techniques specific to collaboration diagrams” on page 168.

Creating a Message-to-self

To create a message from an Object back to itself:

1. Click on the **Self Message** button on the diagram's toolbar
2. Click on the **Object's life line** at the point where you want the Message to appear

Creating a Message link that calls an operation

For such message links there is additional tab “Operation” in the Inspector. If the Editor pane is visible, it displays the source code of this operation.

To create a Message link that calls an operation:

1. Create a Message link between two objects. Both objects must have their instantiates property set to point to a class.
2. Select the message link.

3. In the Inspector, go to the operation field and launch the picker dialog using the field's browse button to display the Choose **Operation Name** dialog. The dialog displays the operations of the recipient object's class.
4. Select the operation and click **OK**. This renames the Message link to the operation's name.

If you select this Message link, the Inspector displays two tabs. The tabs are MessageLink (Message link properties) and Operation. The Operation tab displays the operation's properties.

Reordering Message links

Select and drag Message links up and down the Object lifeline to reorder them. Reordering automatically updates the Message link numbers.

Sometimes you may wish to reorder message links keeping their sequential order and freeing the space between for new links. To do this, select a Message link line, press CTRL, and drag it. This shifts all succeeding links. If you select a number of Message links (pressing Ctrl key), then those selected are moved keeping their increments.

Techniques specific to sequence diagrams

This section provides instructions for refining the sequence diagrams that you have created. The section “Techniques for sequence and collaboration diagrams” on page 166 contains further instructions.

Adjusting the default lifeline of objects

Objects display with a default lifeline when placed on the diagram. Their tops align vertically. If you draw a Message to an Object and then check the **creation** property of the Message, the created Object will move downward to show that it exists at a point forward in time from its creator.

Adjusting the size of object lifelines

You can lengthen or shorten object lifelines as needed by dragging the horizontal line of the bottommost Message link upward or downward. You can arrange the position of other intervening Messages this way also.

Changing the order of a sequence diagram

You can reorder a sequence diagram and maintain any message links already created between objects. Select any Object and drag it to the desired position. Such change is performed across the X axis of Objects- you cannot move Objects vertically along the Y axis except as described in the first point above.

Marking a destroyed object with an “X”

Together can automatically render a bold **X** indicating the destruction of a created object. To use this feature, draw a message to the object and check the message's **destruction** property (Link tab of the Inspector).

Indicating periods of time

Together can automatically render the focus controls of objects that show the period of time that an object performs. To use this feature, draw message links to create the focus controls.

Nesting messages

You can nest messages by originating message links from a focus control.

Specifying properties of message links

You can specify the following properties for message links:

- linked operation
- arguments
- return value
- simple branching condition
- iteration
- time and other constraints

Techniques specific to collaboration diagrams

This section includes instructions for refining the collaboration diagrams that you have created. For more information, see “Techniques for sequence and collaboration diagrams” on page 166.

When you draw a Message between Objects, a generic link line displays between the Objects, and a list of Messages is created above it. The link line is present as long as there is at least one Message between the Objects.

As you add Messages, they display in time-ordered sequence from top to bottom of the messages list. You can select Messages and edit their properties in the message properties Inspector just as you can do in a sequence diagram.

The collaboration diagram adds the capability of showing relationships between objects. In addition to the default link, you can add links to show **Association** and **Aggregation** relationships. These links do not display if you view the diagram as a sequence diagram.

Setting-up shortcuts for sequence diagrams

Together allows you to add a shortcut to a sequence diagram using the Add Shortcuts dialog. To invoke the Add Shortcuts dialog, right-click on the background of a sequence diagram and choose **New | Shortcuts**.

The steps that follow provide an example of adding a shortcut to a sequence diagram:

1. Create a sequence diagram.
2. Add two objects, Object1 and Object2.
3. Add a link between the two objects.
4. Create a second sequence diagram.

5. Create an object, Object1.
6. Right-click on the background of the sequence diagram and choose **New | Shortcuts** to invoke the Add Shortcuts dialog.
7. Double-click the content that you want to add to existing elements and click **Ok**.

Note that in the example above, Together does not reproduce the link between Object1 and Object2 in the second sequence diagram. Instead, you can create links in the second sequence diagram if necessary.

Using Sequence automation to analyze patterns

This section provides a short tutorial exercise that can help you learn how to use Sequence automation to analyze patterns.

For the exercise, you need a Together product that supports sequence diagrams and sequence diagram automation. (To learn where to get product feature information, see Where to Get Help.)

IMPORTANT: For a sequence diagram automation to function correctly, the Together installation directory name must not contain spaces.

Create an example project for training purposes:

1. Launch Together.
2. From the main menu, choose **File | New Project** to invoke the New Project dialog.
3. In the New Project dialog enter a project name patterns. Use browse button to select or create a directory for your project (under “myprojects” for example), return to the New Project dialog and click **OK** to create the project.

Analysis of the Singleton pattern

Now use the Patterns feature to create an instance of the GoF *Singleton* pattern in the project just created.

To create the Singleton pattern:

1. Create a new package and name it singleton.
2. On the “singleton” package icon invoke the right-click menu and select **Open in New Tab** to open the default package diagram for the new package. This diagram is empty.
3. On the diagram toolbar, click **Class by Pattern**, then click again on the diagram workspace. The “Choose Pattern” dialog appear.
4. In the patterns tree, select the pattern GoF - Singleton and click the **“Finish”** button. A new class Singleton is created in the package and displays in the diagram.

At this point you have an instance of the Singleton pattern in the project. It is displayed visually in the default diagram and its source code is loaded in the Editor. You can see the attributes and operations of the class in the diagram.

Suppose now that you want to understand more clearly, and then model the workings of the `getInstance()` method. Normally you would have to study the code and manually construct a sequence diagram for this purpose. With Together, you can simply generate this diagram.

To generate a sequence diagram for the `getInstance()` method:

1. In the Singleton class icon, select the method `getInstance:Singleton`, invoke right-click menu and choose Generate Sequence Diagram. The Generate Sequence Diagram Expert dialog appears.
2. Click **OK**. A sequence diagram named “Singleton.getInstance” is created and opened in a new tab in the Diagram pane.
3. Select the activation rectangle of the message labeled `/getInstance():Singleton...` that is, the destination of this message.
4. On the main toolbar click the Editor Pane icon to display the Editor pane, which is hidden by default for sequence diagrams, or select View | Editor Pane from the main menu. The insertion cursor will be positioned on the method declaration for the `getInstance()` method.

Analysis of the composite pattern

1. Left click on the diagram tab named “<default>”. The <default> package diagram will be opened in the Diagram Pane.
2. Create new package and rename it to “composite”.
3. On the “composite” package icon invoke right-click menu and select “Open in New Tab”.
4. Press the diagram toolbar button “Class by Pattern” and click on the diagram workspace. The “Choose Pattern” dialog displays.
5. In the patterns tree select pattern GoF|Composite and click **Finish**. Two classes will be created in the package.
6. In the “Composite” class icon select method “sampleOperation:void”, invoke right-click menu and select menu item “Generate Sequence Diagram”. The “Generate Sequence Diagram Expert” dialog displays.
7. Press **Ok**. A sequence diagram named “Composite.sampleOperation” will be created and opened in a new diagram tab.

The resulting sequence diagram shows some things which are not relevant: two `java.util` classes (`Vector`, `Enumeration`), and message to self in the “initial” object.

To refine the diagram and show only collaboration between the pattern classes the GSD options should be changed.

1. In the main toolbar press “Undo” button. The “Composite.sampleOperation” diagram tab will disappear, “composite” package diagram will become current diagram.
2. On the main toolbar press “Diagram view management...” button. The “Diagram options” dialog appears.
3. Uncheck option sequence diagram - Include messages to self and press **OK**.
4. In the “Composite” class icon select method “sampleOperation:void”, invoke right-click menu and select menu item “Generate Sequence Diagram”. The “Generate Sequence Diagram Expert” dialog displays. This dialog contains a tree-table showing the classes that take part in the given collaboration. The tree-table consists of packages on the first level and classes on the second level. This table enables you to exclude from the generated sequence diagram either the implementation of some classes, or the classes themselves.

5. In the tree-table select package `java.util` and uncheck checkbox in the column “Show on diagram”.
6. Click the **OK** button. A sequence diagram named “Composite.sampleOperation” is created and opened in a new diagram tab.

Generating implementation source code

This section demonstrates how to draw and edit a sequence diagram that generates source code.

Creating project and class

1. Create a new project. In the New Project dialog enter project name and press **OK**. <default> package diagram displays.
2. Create new class `MyApplication`.
3. On the `MyApplication` class right-click menu select Choose pattern to display the Choose Pattern dialog.
4. In the patterns tree select Main class pattern.
5. Press the **Finish** button. New method `main()` is created.

Generating a sequence diagram from a class

1. Select **Generate Sequence Diagram** from the main method right-click menu. The Generate Sequence Diagram Expert dialog appears.
2. Click **Ok**. The sequence diagram `MyApplication.main` opens in a new diagram tab.
3. On the main toolbar click the **Diagram View Management**  button, or choose **Options | Diagram Level** from the Tools menu and choose **View Management | Sequence Diagram** from the tree. The View Management page of the Options dialog displays.
4. Check the option **Sequence diagram | Show message numbers** and press **OK**.

Creating source-generating elements in the sequence diagram

1. On the diagram toolbar press the **Statement Block** button and click on the activation rectangle of the message #1. Choose statement type dialog displays.
2. Check for radio-button and click **OK**. The statement for and its shaded rectangle show up on the diagram.
3. Invoke in-place editor and enter: `int i = 0; i < 4; i++`. The label reads: `for(int i = 0; i < 4; i++)`
4. Press **Object** button on the diagram toolbar and create a new object with the name frame.
5. Select **Choose Class | More** on the object's right-click menu to display Choose Object's Class dialog.

6. In the Search/Classpath, select class `javax.swing.JFrame` and click **OK**. The name of selected class displays in the object's icon.

Creating message sends

1. Draw a message link from the statement block `for(int i = 0; i < 4; i++)` to the lifeline of the frame object. Message #1.1.1 is created. All the other messages should have same source and destination.
2. Select message #1.1.1. and choose Type | Creation on its right-click menu. The message will change visually to the creation type (the link now points to the frame object, which means that the object is being created).
3. Draw a new message. Its label is message #1.1.2. Select Choose Operation on the right-click menu. From this dialog choose `setDefaultCloseOperation(int):void`. The message label becomes: 1.1.2:setDefaultCloseOperation(int):void.
4. Invoke in-place editor and enter (`JFrame.EXIT_ON_CLOSE`) after the message number and column. The message label becomes 1.1.2: {setDefaultCloseOperation(:int):void}(JFrame.EXIT_ON_CLOSE).
5. Draw message #1.1.3. Using Choose Operation dialog, select `setSize(int,int):void` method. Invoke the in-place editor and enter: (`600, 400`) after the message number and column. The message label becomes 1.1.3: setSize(600,400):void
6. Draw message #1.1.4. Using Choose Operation dialog, select `setLocation(int,int):void` method. Invoke in-place editor and enter (`50*i, 50*i`) after the message number and column. The message label becomes: 1.1.4: {setLocation(50*i,50*i):void}.
7. Draw message #1.1.5, invoke its right-click menu and select Choose Operation | More to display the Choose Operation Name dialog. Choose `show()` method from the list. The message becomes 1.1.5:show():void

TIP: If you cannot find the desired name in the list, expand All Operations node and search through the loaded list of methods.

Generating implementation code

1. Select the activation rectangle of the message #1, invoke right-click menu and select Generate Implementation. to create implementation of the main() method. This implementation code displays in the Editor. Message labels for which implementation code has been generated, display **bold** style on the diagram. Message pane opens. If for some reasons code generation fails for certain messages, they remain normal style on the diagram, and appropriate messages show up in the Message pane.
2. Select any message on the diagram, and observe that the Editor scrolls to the point of appropriate method invocation.
3. In the Editor, add import statement for `javax.swing`.
4. The main menu | Tools and the diagram right-click menu allow to Run the generated code and Make the project. In the former case you will observe a cascade of four frames. In the latter case the following messages show up in the Builder tab:

```
*** Make started
*** Make completed
*** Output directory: D:\Together5\out\classes\MyApp
```

Statechart Diagrams

Statechart diagrams enable you to model UML state machines. State machines are one of several ways to model the dynamics of a system or subsystem. They generally model the behavior of a single object as it goes through different states in response to events that occur during its lifetime, including its response to events. The state of an object is some situation during its life where it meets some condition(s), does something, or waits for an event. Figure 33 illustrates an example of a statechart diagram.

Statechart diagrams emphasize the possible states of an object and the transitions between states. You can use them to model the lifetime of an instance of a class, an instance of a use case, or even an instance of a system. During that period, the object of interest can be exposed to different kinds of events to which it responds with some action, and the result of which is a change in the object's state.

TIP: To focus on the actions and activities of objects, use activity diagrams.

Figure 33 Statechart diagram

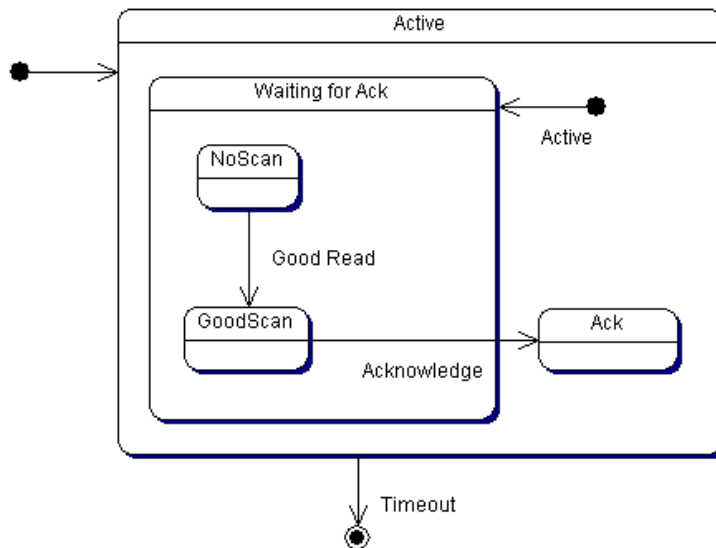


Diagram elements

Table 19 lists the contents of statechart diagrams as defined by the UML.

NOTE: Relationship links appear as *italicized* text.

Table 19 Contents of statechart diagrams

Icon	Element
	State
	Start
	End
	History
	<i>Horizontal Fork/Join</i>
	<i>Vertical Fork/Join</i>
	Transition
	<i>Object</i>
	<i>Concurrent State</i>
	<i>Note</i>
	<i>Note link</i>

- **Instantiates**

Specifies the Object's classifier. You can either enter the value yourself, or pick the class from a dialog that lists all the resources available to the project.

When you apply this property, the Object displays its fully qualified name. The properties Inspector displays the Class tab in which you can access the properties of the Object's class.

TIP: You can also specify the classifier by selecting the Object in the diagram and calling **Choose Class** from its **right-click menu**.

Drawing a self-transition

Self-transition means that the flow leaves the State, dispatching any exit Action(s), then re-enters the State, dispatching any entry Action(s).

- Select the desired State on the diagram.
- Drag a Transition link from the selected state and drop it on the diagram background.
- In the Choose Destination dialog, navigate to the State where you began the link, select it, and click **OK**.

TIP: Another way is to draw a Transition between two States, and then drag the opposite end of the link line back to the desired State.

Creating internal transitions

Internal transition is a shorthand for handling events without leaving a State and dispatching its exit/entry Actions.

- Select the desired State on the diagram
- From its right-click menu choose **New Internal Transition**.

You can immediately edit the event name in place. For additional information see Entry/Exit Actions below.

Specifying entry/exit actions for a state

Entry and exit actions are executed upon entering or leaving a State, respectively. You create Entry and Exit Actions in Together Statechart diagrams as stereotyped internal transitions.

Create an internal transition in the desired State. When you edit the name in place, use the following syntax: `stereotype/actionName[argument]`

Example: `exit/setState(idle)`

Optionally, you can create the internal transition and set the event name, event arguments and action expression properties in the properties Inspector (right-click menu | Properties).

Creating nested substates

You can create a composite State by nesting one or more levels of States (i.e. substates) within one State. You can also place Start/End states and History states inside a State, and draw Transitions among the contained substates.

The easiest way to create a nested substate is to place a State node on the diagram background, drag it on top of another State, and drop it.

Showing multiple transition sources or targets

A Transition may have multiple sources, meaning it is a join from several concurrent states, or it may have multiple targets, meaning it is a fork to several concurrent States.

You can show multiple transitions with either a vertical or horizontal orientation in your Statechart diagrams. The Statechart diagram toolbar provides separate Fork/Join buttons each orientation. The two orientations are semantically identical.

To create multiple Transitions:

1. Identify the States involved. If necessary, place all the States on the diagram first, and lay them out as desired.
2. Place either a horizontal or vertical Fork/Join on the diagram. Resize it as needed.
3. If depicting multiple sources, draw Transitions from each of the source States to the Fork/Join;
4. If depicting multiple targets, draw a Transition from the source State to the Fork/Join; then draw Transitions from the Fork/Join to each of the target States.

Tips for modeling complex states

The techniques in this section pertain to models of particularly complex composite states and substates.

- **Resizing the main state**

You can resize the main state. You can essentially draw another Statechart diagram inside it, complete with Start/End/History states and transitions of all kinds, to create, in effect, a substate diagram

- **Using hyperlinks to indicate nested substates**

You can nest multiple levels of substates inside one State. For especially complex substate modeling, however, you may find it more convenient to create different diagrams, model each of the substate levels individually, and hyperlink the diagrams sequentially.

- **Applying existing content to a new diagram**

Using the Clone command on the context menu of the diagram's Explorer node, you can quickly create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.

- **Reusing elements of existing diagrams**

Using the Add Shortcut command on the diagram's right-click menu, you can reuse existing elements in other statechart diagrams. After invoking the Add Shortcut command, navigate within the dialog's Explorer to the existing statechart diagram and select one or a number of its elements, States, Histories, and/or SyncBars (Fork/Joins).

Activity Diagrams

Activity diagrams provide one of the possible ways to model system dynamics. An activity diagram is basically a flowchart that describes the flow of control from one activity to the next. You can show sequential and/or concurrent steps of a process, model business workflows, model the flow control of an operation, or the flow of an object as it passes through different states at different points in a process. Unlike interaction diagrams (Sequence, Collaboration) that emphasize the flow of control between objects, activity diagrams emphasize the flow of control between activities.

Figure 34 Activity diagram

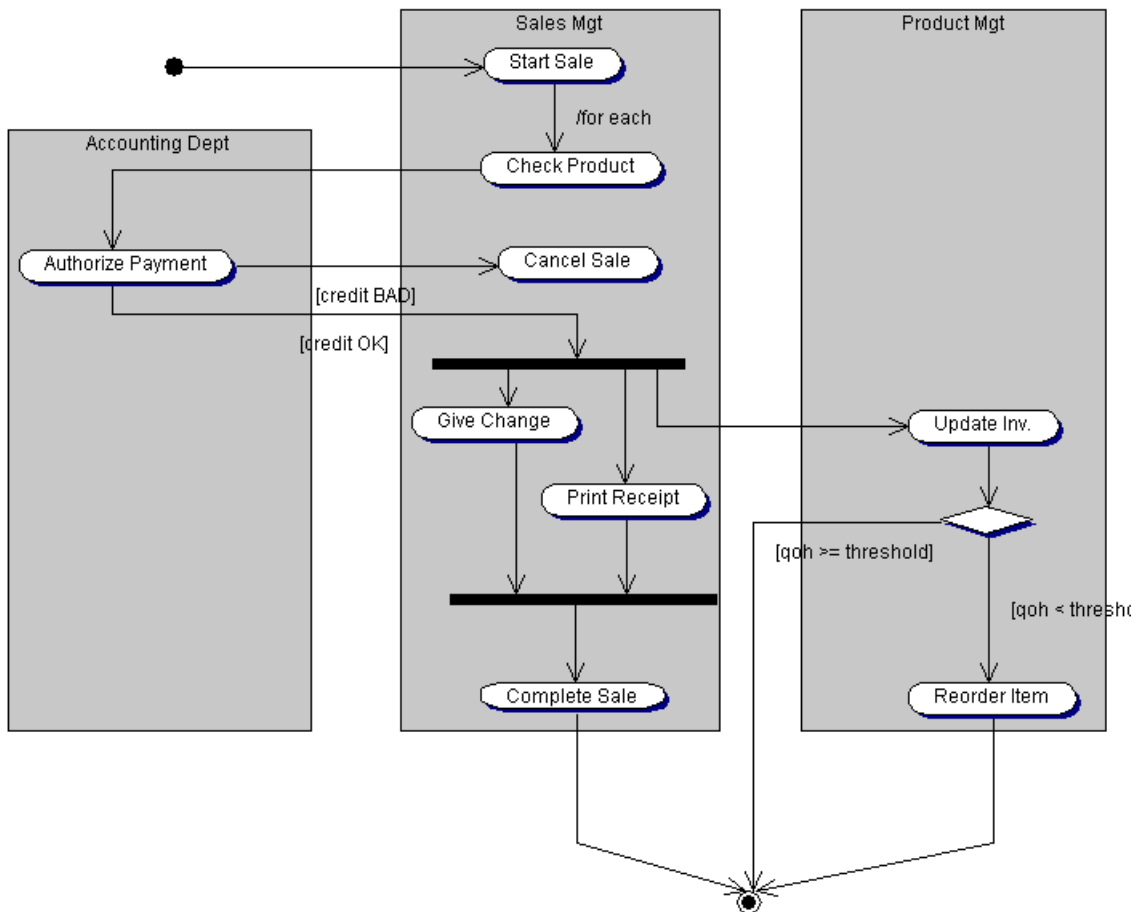


Diagram elements

The content of an activity diagram can vary depending upon what kind of control flow you are modeling and what level of detail you choose to provide. Table 20 lists the contents of activity diagrams as defined by the UML.

NOTE: Relationship links appear as *italicized text*.

Table 20 Contents of an activity diagram

Icon	Element
	Activity
	Decision
	Signal Receipt
	Signal Sending
	State
	History
	Object
	Start state
	End state
	Horizontal Fork/Join
	Vertical Fork/Join
	Swimlanes
	Transition

Table 20 Contents of an activity diagram (continued)

Icon	Element
	Object flow
	Note
	Note link

Working with activity diagrams

Following are tips and techniques that you can use when creating activity diagrams:

- **Creating diagrams with identical content**

Non-trivial systems will probably require many activity diagrams to capture the dynamics of a workflow or operation. Use the Clone feature to create new diagrams with identical content in the same package.

- **Reusing elements of existing diagrams**

Using the Add Shortcut command on the diagram's right-click menu, you can reuse existing elements in other activity diagrams.

NOTE: Elements imported this way are independent copies of the existing ones.

Start with the main flow modeling. Next, cover branching, concurrent flows, and object flows. Use separate diagrams as needed and hyperlink them for easy browsing later on.

- **Showing labels inside decisions**

To show labels inside decisions, go to the `$TGH\config\diagram.config` file and change the value of the option `option.show_decision_label` to true (it is set to false by default).

Note that if your labels are long (for example, when a project from an older version of Together is loaded), it may affect the diagram layout.

Component Diagrams

Component diagrams, similar to deployment diagrams, depict the physical architecture of a computer-based system. A component diagram usually shows the components and interfaces of a system, and the relationships between them. Figure 35 illustrates an example of a component diagram.

A component — as the main element in a such type of diagrams — is used to package other logical elements, and represents things that participate in the execution of a system. Components also use the services of another component via one of its interfaces. Usually, components are used to visualize logical packages of source code

(work product components), binary code (deployment components) or executable files (executions components).

TIP: For detailed information about component modeling in color, refer to *Java Modeling in Color with UML: Enterprise Components and Process* by Peter Coad, Jeff De Luca, and Eric Lefebvre.

Figure 35 Component diagram

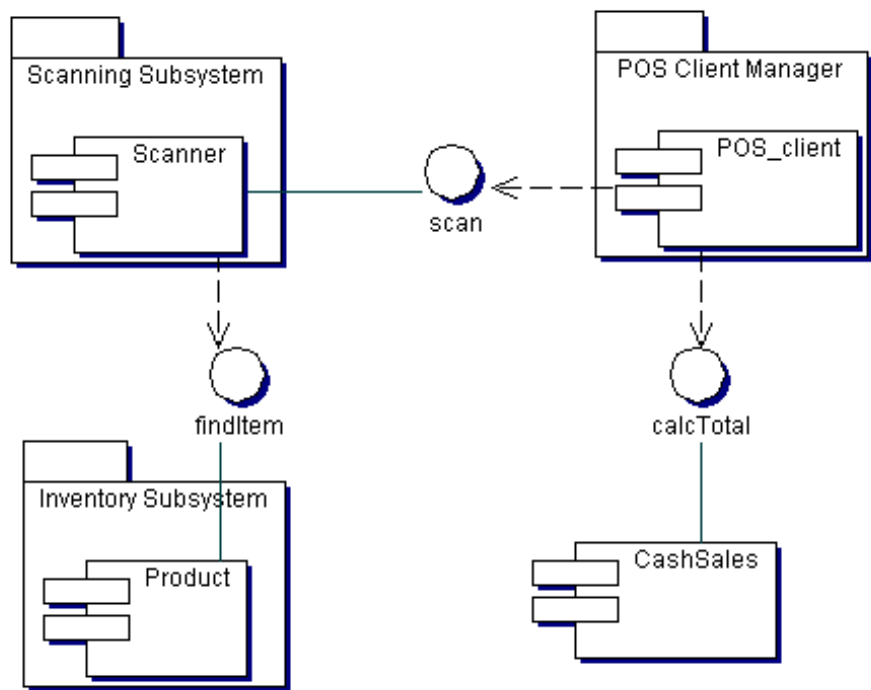


Diagram elements

Component diagrams in Together depict two kinds of relationships: dependency and realization. To group one or more logical elements of the model, component diagrams also can use packages or subsystems. Table 21 lists the contents of component diagrams as defined by the UML.

NOTE: Relationship links appear as *italicized text*.

Table 21 Contents of component diagrams

Icon	Element
	Subsystem

Table 21 Contents of component diagrams (continued)

Icon	Element
	Component
	Interface
	Note
	Note link
	<i>Dependency</i>
	<i>Supports</i>

Working with component diagrams

Following are tips and techniques that you can use when creating component diagrams:

- **Applying existing content to a new diagram**

Using the Clone command on the right-click menu of the navigation pane node, you can quickly create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.

- **Reusing elements of existing diagrams**

Using the Add Shortcut command on the diagram's right-click menu, you can reuse existing elements in other state diagrams.

NOTE: Elements imported this way are independent copies of the existing ones.

- **Representing realization**

You can represent realization in two ways: using the support stereotype and the dashed line (canonical form), and using the solid line (lollipop notation)

- **Hiding subcomponents**

You can hide subcomponents on a component diagram. Choose **Tools | Options | Default Level | View Management**, and enter the following for one of the “User Defined” fields under “Show”:

Name: "Nested components"

Expression:

```
hasPropertyValue("$shapeType", "Component" ) && !(getContainingNode()  
== null)
```

Deployment Diagrams

A deployment diagram, similar to a component diagram, provides a way to model the physical aspects of a computer system. It is a graph of nodes connected by communication associations, showing the physical architecture of the hardware and software of the system. Figure 36 illustrates an example of a deployment diagram.

Figure 36 Deployment diagram

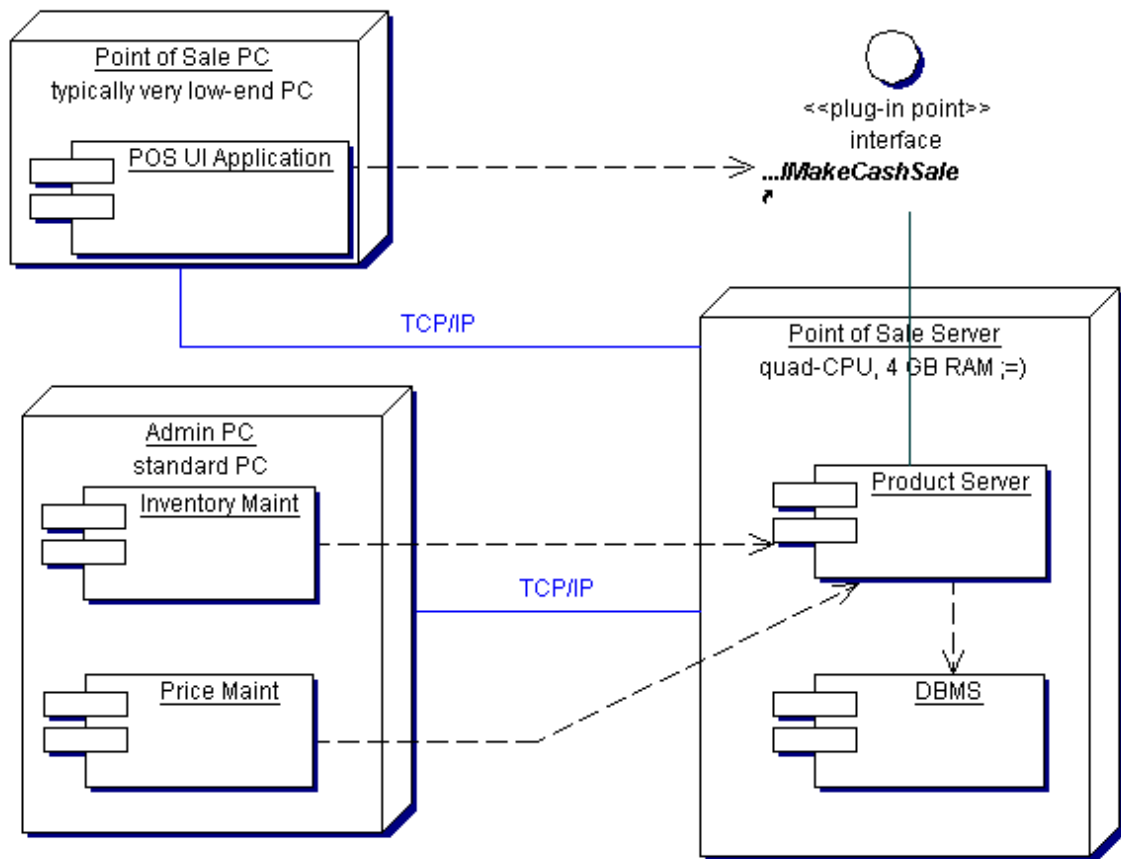


Diagram elements

Table 22 lists the contents of deployment diagrams as defined by the UML.

NOTE: Relationship links appear as *italicized* text.

Table 22 Contents of deployment diagrams

Icon	Element name
	Node
	Component (specific to Together): Live on nodes and they may provide realization of interfaces
	Interface (specific to Together)
	<i>Supports</i>
	<i>Associates</i>
	<i>Aggregates</i>
	Object (specific to <i>Together</i>): Live in processes that live in components
	Dependency
	Note
	<i>Note link</i>

Working with deployment diagrams

Following are tips and techniques that you can use when creating deployment diagrams:

- **Applying existing content to a new diagram**

Using the Clone command on the right-click menu of the navigation pane node, you can quickly create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.

- **Reusing elements of existing diagrams**

Using the Add Shortcut command on the diagram's right-click menu, you can reuse existing elements in other state diagrams.

NOTE: Elements imported this way are independent copies of the existing ones. Organize components by specifying the relationships between them.

- **Indicating a temporary relationship between a component and node**

Objects and components can migrate from one component instance to another component instance, and respectively from one node instance to another node instance. In such a case, the object (component) will be on its component (node) for a temporary period of time. To indicate this, use the dependency relationship with a *becomes* stereotype.

- **Representing how a component resides on a node**

A component may reside on nodes. You can represent this in two ways: using the *support* stereotype and the dashed arrows, and by graphically nesting the component symbol within the node symbol.

UML EXTENSION DIAGRAMS

In addition to the UML diagrams discussed in Chapter 10, Together supports extension diagrams that are not currently supported by the UML standard. To access the extension diagrams, select the **Together** tab from the New Diagram dialog. For more information, see “Creating diagrams in projects” on page 124.

This chapter includes the following topics:

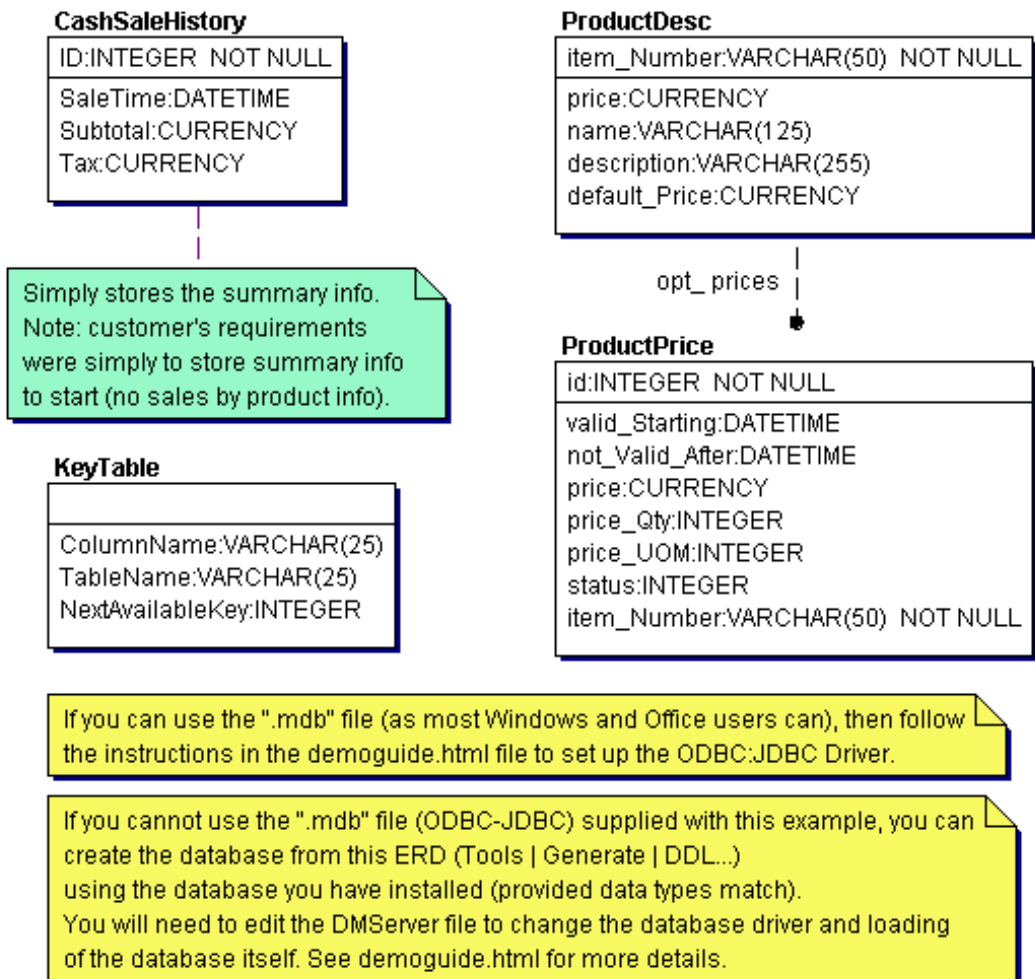
- “Entity relationship diagrams” on page 185
- “Business Process Diagrams” on page 190
- “Robustness analysis diagrams” on page 192

Entity relationship diagrams

The entity relationship diagram (ERD) is a high-level data model that views business data in terms of entities and relationships. The ERD visually represents data objects, as illustrated in Figure 37, providing a view of the business information required to develop an information system.

For more information about Together products that support ERD’s, visit www.togethersoft.com.

Figure 37 Entity relationship diagram

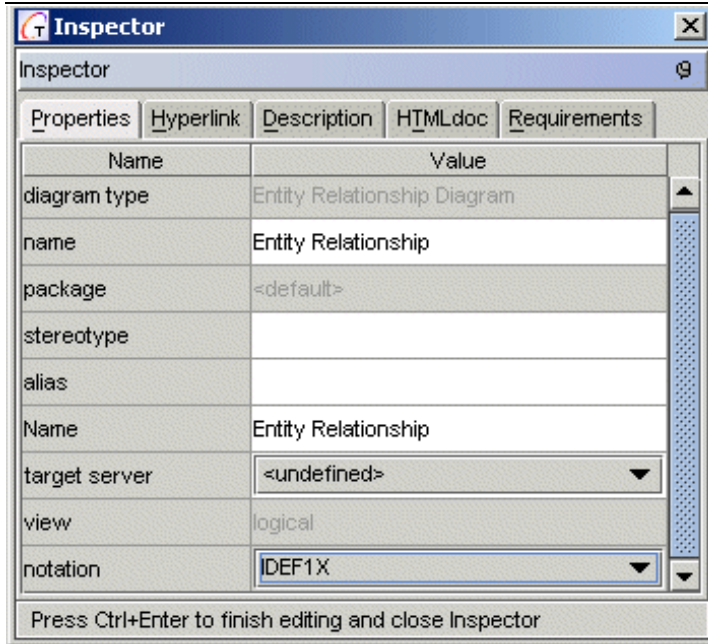


Notation

There are two ways to represent data objects in an ERD. Together offers a choice between IDEF1X, IE, or Crow Feet notation.

To switch between notations, open the diagram object inspector and select the desired notation from the drop-down list in the *Properties* tab:

Figure 38 Set notation for an ERD



When you change the notation, the diagram automatically redisplay its contents.

Logical and physical diagram view

Various database servers use different names of similar attribute types (for example, NUMBER in Oracle and INT in Cloudscape). Together supports portability of the data structures providing logical and physical views of ERD's. Normally, modeling is carried out in a *logical view*, which displays the attribute names only. However, you can choose *physical view*, and show attribute names and types in the notation specific for the selected database server.

The server is chosen from the *target server* drop-down list in the diagram object inspector. If no target server is specified, logical view is assumed by default, and the *view* field is disabled.

TIP: Changing the target server, press F5 to redisplay the diagram contents.

Object inspector provides *Logical view* and *Physical view* tabs for the entities, relationships and attributes. Modification of a property in the logical view causes automatic change in the physical view. The reverse is not true - some property values are editable in the physical view, but the corresponding values in the logical view stay intact.

Diagram elements

Table 23 lists the contents of ERD's.

Table 23 Contents of entity relationship diagrams

IDEFIX	IE	Description
		Entities are represented by labeled rectangles. The label is the name of the entity. Entity rectangles are divided into two sections.
		Attributes , when included, are listed inside the lower section of the entity rectangle.
		Primary key attributes when included, are listed inside the upper section of the entity rectangle.
		Identifying relationships: The relationship name is written above the line. Verbs are preferable.
		Non-identifying relationships: The relationship name is written above the line. Verbs are preferable.
		Many-to-many relationships
		Note
		Note link

Entities

Entities are represented by the labeled rectangles, divided into two sections. Client entities with identifying relationships are displayed as rectangles with rounded corners. Names of the entities should be singular nouns.

Attributes

Attributes are displayed in the lower section, and the Primary key attributes are displayed in the upper section of the entity icon. Names of the attributes should be singular nouns.

In the logical view of the attribute object inspector you can edit the attribute name and choose its type from the drop-down list. You can also make an attribute a *Primary key*, and impose certain restrictions on its value (*not null*, *unique*).

Is Foreign key flag is a read-only field that displays the status of an attribute depending on the relationship between two entities.

The physical view displays read-only fields *Physical type*, *Size and Digits*, which depends on the selected target server and specific driver.

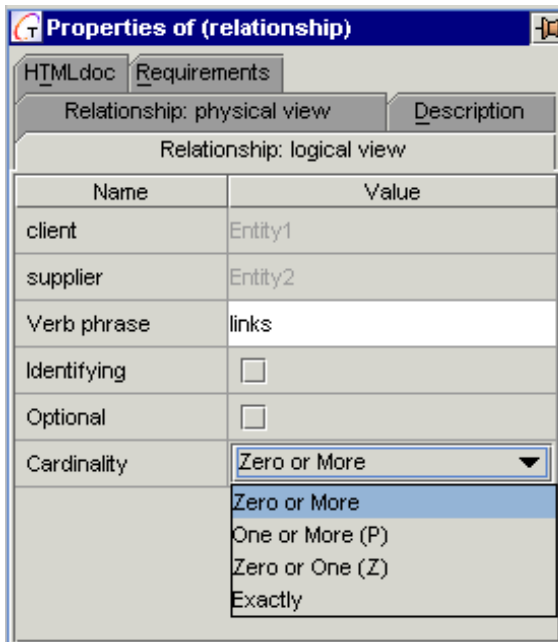
The *Size* parameter used for the string variables specifies the maximum number of characters (Digits field is disabled). If attribute type is numeric, *Size* specifies the total length of the number. The sense of *Digits* field depends on the selected database server. In general it defines the number of digits after the decimal point. However, for certain servers (e.g. Oracle 7.3.x/8.x and SequelLink/Oracle) this value stands for precision.

Relationship links

Relationship links are displayed according to the selected cardinality, with the names written above the line. It is advised to use verbs as the names of the links.

You can create identifying or non-identifying relationships using the appropriate toolbar icons. Identifying relationships are displayed in solid line, and non-identifying relationships display as dotted lines. Once created, a relationship link can be modified through its properties inspector, as shown in Figure 39.

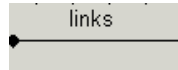
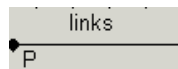
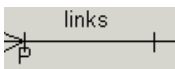
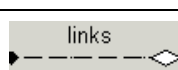
Figure 39 Properties inspector for an ERD



Logical view tab of the inspector displays client and supplier names of the selected relationship link, and provides a test area *Verb phrase*, where you edit the relationship name.

Display of cardinality also depends on the selected notation. The following types of cardinality are possible:

Table 24 Cardinality

Cardinality	IDEF1X	IE
Zero or more		
One or more		
Zero or one		
Exact cardinality (arbitrary integer value)		
Optional		

Business Process Diagrams

While not currently specified as a diagram type in the UML, Together provides business process diagrams to enable you to apply some of the UML extensions to business modeling. Business object models depict the structure, processes, use cases, and relationships of a business. The business object model describes the realization of business use cases, providing an abstraction of how business workers and business entities are related, and how they must work together to actually run the business.

A business object model describes the use cases of a business from the internal viewpoint of business workers. It defines the static and dynamic aspects of relationships between the workers and the classes and objects they use to produce the expected results. In aggregate, the objects of the model's classes are capable of performing all the use cases of the business.

Using business process diagrams, you can model the static aspects of a business object model, especially business use cases. To one or more business process diagrams, you can add Sequence and Activity diagrams to show the dynamic aspects of the business object model and thus achieve a complete business model.

Diagram elements

Table 25 lists the contents of business process diagrams.

Table 25 Contents of business process diagrams

Icon	Element
	Business Actors
	Business Use cases
	Business Workers
	Business Entities
	Relationship links
	Association/Communicates
	Extends
	Includes
	Generalization
	Aggregate
	Subscription
	System boundary
	Note
	Note link

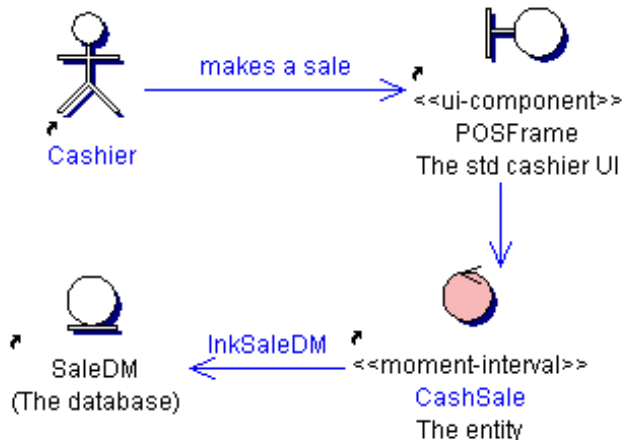
Notation

The current UML specification (v. 1.3) does not specify any graphical variation for elements such as Use Case and Actor when used in a business object modeling context. Together business process diagrams use the standard UML graphical notation for these elements, and provides compliant notation for UML extensions such as business worker and business entity.

Robustness analysis diagrams

Robustness analysis involves evaluating the text of a use case and planning how to implement a given use case using the objects you've discovered up to this point. One of the main purposes of robustness analysis is to determine whether you have all the objects you need, and then add them to your class diagram. While exploring possible designs, robustness analysis suggests classifying objects into one of three stereotypes: boundary, control, and entity. Figure 40 illustrates a robustness analysis diagram.

Figure 40 Robustness analysis diagram



The robustness model provides a bridge between the "analysis level" view provided by the text of the use case and the "detailed design" view shown on a sequence diagram. Since it's very difficult to jump from analysis directly to detailed design, it's hard to do modeling successfully without this step.

Drawing the robustness diagram for a use case provides a visual completeness check that shows that the entire use case has been accounted for. It also serves to enforce the "active voice" style that is most effective when writing use cases. It is beyond the scope of this guide to provide detailed review of the robustness analysis. The most comprehensive information on this issue can be found in *Use Case Driven Object Modeling with UML* by Doug Rosenberg with Kendall Scott.

Diagram elements

Table 26 lists the contents of robustness analysis diagrams.

Table 26 Contents of robustness analysis diagrams

Icon	Element
	Boundaries
	Entities
	Controllers
	Worker boundaries
	Worker controllers
	Actors
	Relationship links:
	Associations
	Robustness Associations
	Note
	Note link

Key elements and properties

The robustness diagram symbols such as Entity, Boundary, or Worker Boundar, represent stereotyped views of classes. For each symbol on the robustness diagram, there is a corresponding class. There is a mechanism for converting a Controller and a Worker Controller to methods. Rosenberg’s suggestion of using “invokes” in place of “includes” and “precedes” in place of “extends” is realized via specific Robustness Association.

Key elements and properties of robustness diagrams include:

- **Boundary**

Boundary objects are the objects in the new system with which the actors will be interacting. These may be windows, screens, dialogs, and menus.

- **Entity**

Entity objects refer to the database tables and files that store data, fetch data and perform computations that do not change frequently. These objects should be enough simple and generic enough to provide the possibility of future reuse.

- **Controller**

Control objects, or controllers, represent the functionality and system behavior of the use cases. Controllers may be converted into methods associated with the interface objects and entity objects.

- **Association**

Links are represented by arrows. Unlike sequence diagrams, these arrows do not correspond to messages. They rather indicate logical associations.

The associations should follow certain rules:

1. Actors can talk only to boundary objects
2. Boundary objects can talk only to controllers and actors
3. Entity objects can talk only to controllers
4. Controllers can talk to both boundary objects and controllers but not the actors.

CHAPTER 12

REAL-TIME MODELING

The Together real-time module provides support for defining, modeling, and simulating object-oriented real-time systems and applications. The module draws heavily on the object-oriented approach to developing real-time systems described in *Object-Oriented Technology for Real-Time Systems* by Maher Awad, Juha Kuusefa, and Jurgen Ziegler.

The topics discussed in this chapter include:

- “Introduction to Together’s real-time module” on page 195.
- “Special real-time diagrams” on page 197.
- “Together real-time audits” on page 211.
- “Simulation” on page 212.

Introduction to Together’s real-time module

The real-time module adds new diagrams to Together and new capabilities to existing Together diagrams to support requirements-to-implementation real-time system development.

Capabilities of the real-time module

Together’s real-time module has the following features:

- **System context diagram** for capturing real-time requirements
- **System architecture diagram** to model subsystems
- **Event sheet diagram** to define real-time system events
- **Interaction diagram** to show object groups, threads, and messages
- **Concurrent state element** for statechart diagrams
- **Expanded set of real-time asynchronous messages** for sequence and collaboration diagrams

- **Support for scenario** sequence diagram
- **Real-time specific use case stereotypes:** activity and exception
- **Analyze thread execution of real-time system design:** use the integrated simulator (**Tools | Real-Time | Simulate**) or export the model to XML and use an external simulator
- **Traceability** from system context and requirements to individual threads, objects, and messages
- **Time-based requirements**
- **Real-time audits**

Getting started

Getting started with the real-time module is a 3-step process:

1. Activate the real-time module:
 - Select **Tools | Activate/Deactivate Features** in Together's main menu.
 - In the **Together Features** tabbed page, check **Together Real-Time**.
 - Click **OK**.
2. Open a real-time project or create a new one. Together ships with a sample real-time project under \$TGH\$/Samples/real-time/RetinaScanner.tpr, which you can use to learn more about real-time modeling in Together ControlCenter. Be sure to read through the specific documentation for each diagram, looking at the project examples.
3. Read the process overview to familiarize yourself with the development approach.

Real-time development process

The real-time module can support several approaches to object-oriented software development. However, the approach described in *Object-Oriented Technology for Real-Time Systems* is the foundation for the real-time diagrams included in the module. This section gives a brief overview of the approach advocated by Awad, Kuusefa, and Ziegler in their book.

Real-time development can be described in phases: system requirements, system architecture, subsystem analysis, subsystem design, and subsystem implementation.

Gathering system requirements

During the first part of the requirements phase, you capture use cases and requirements for the system. You then create a System Context Diagram to place the system within the environment described by the use cases and requirements. This includes establishing the relationships between a system and the actors connected to the system.

Deciding system architecture

The system architecture is defined in terms of subsystems and their interfaces. In this phase, you decompose the system into subsystem modules, translating the requirements

into subsystem responsibilities. By decomposing into subsystems early, you can create loosely coupled systems that allow for parallel analysis of separate concerns.

Analyzing subsystems

All object-oriented analysis is performed at the subsystem level. The analysis of a subsystem consists of:

- One or more class diagrams
- Functional descriptions in terms of class operations and attributes
- State diagrams
- Event sheets
- Scenarios

You spend most of this phase developing the object model and describing its entry points and state transitions.

Designing subsystems

During the design phase, you focus on defining groups of objects involved in particular event/thread interactions. At the end of this phase, you should have a clear concurrency design and a set of class outlines. The concurrency design deals primarily with describing the interactions or messages between objects and groups of objects running in (potentially) separate threads.

Implementing subsystems

At this point, you add code to implement the functionality represented in the class outlines. You also implement any required state machines and messages. *Object-Oriented Technology for Real-Time Systems* describes some simple rules for moving from design to implementation. Together's patterns and templates capabilities also provide a solid foundation for introducing reusable implementation options.

Special real-time diagrams

Together supports real-time modeling through enhanced UML diagrams as well as four special additional diagrams. You can create one of the four special diagrams by following these steps:

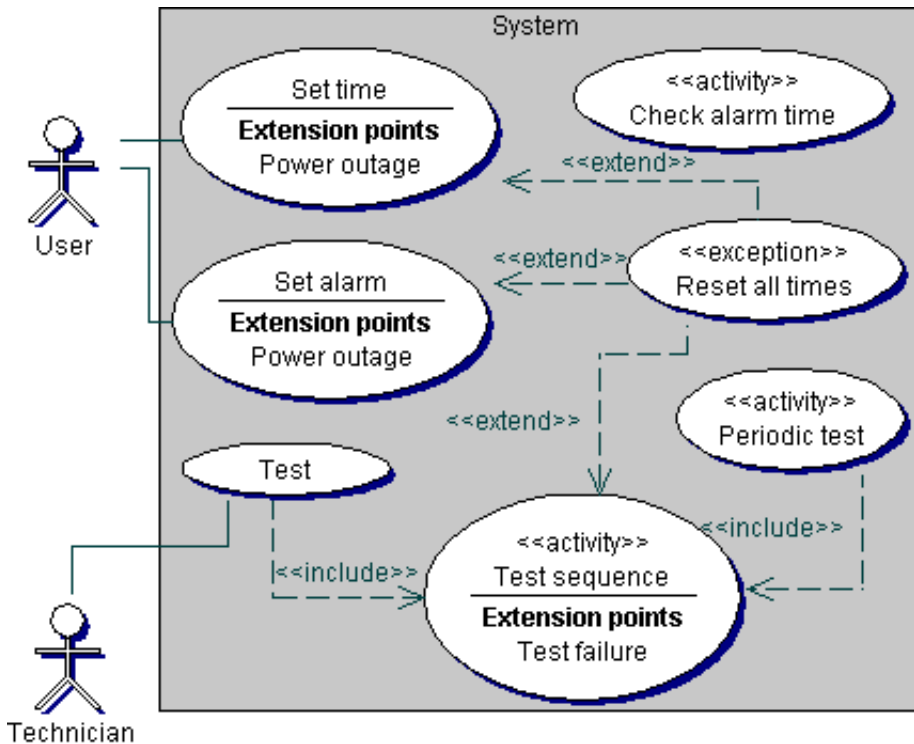
1. Click the **New Diagram** button or press **Ctrl+Alt+N**.
2. Choose the **UML plus Real-Time** tab from the resulting dialog box.
3. Select the desired diagram type: **System Context**, **System Architecture**, **Interaction**, or **Event Sheet**.
4. Fill-in the name and the rest of the diagram information.
5. Click **Ok**.

Use case diagrams

Real-time use case diagrams are enhanced UML use case diagrams. The real-time versions include two new use case stereotypes: **activity** and **exception**. Use case diagrams capture functional requirements for real-time systems. These requirements include tasks initiated by an outside actor, internally initiated tasks, and exception handling capabilities.

Figure 41 is an example of a real-time use case diagram. We will use it to discuss the different stereotypes and links.

Figure 41 Real-time use case diagram



The System includes three use cases on the left side of the diagram, three activities , and one exception.

Use case stereotypes

A use case, like Set time, without a stereotype represents a traditional use case. Some external entity such as a person, an environmental change, or another system initiates the use case. In the example diagram, a User could set the time of the system and set an alarm. A Technician could request that the system run a test.

Activity

An activity use case represents a sequence of events triggered internally. The system is responsible for starting the use case's action. Activities usually represent things like periodic checks, timed actions, or continual processes. In Figure 41, the system is responsible for checking the time against the set alarm, for periodically kicking off tests, and for conducting the test sequence.

Exception

An exception is the sequence of events the system should take in response to an exceptional condition occurring. In Figure 41, resetting all of the times in the system is a response to two different exception conditions: Power outage and Test failure. Exceptions do not have to be connected to use cases and activities in this manner. Some global exception handlers may belong directly to the system.

Use case links

An **include link** indicates that one use case is included in or part of another. For instance, the Test sequence from Figure 41 is executed as part of the sequence of the periodic Test.

Extend

An extension represents a behavior specialization at a particular point within another use case. These points of specialization or variation are called extension points. In the example diagram, the extension points are exceptions that can occur during the use cases or activities to which the exception is linked. For example, in Figure 41, if the Test sequence activity reaches the Test failure extension point, the system invokes the Reset all times exception handling sequence.

Extends links are not limited to relationships between exceptions and other use cases. Use cases and activities may also extend each other to indicate variation in behavior.

Actors

You can link an actor to any use case with which it interacts or to the system use case. You should never link actors directly to activities nor to exceptions.

System context diagrams

When you create the use case diagram, you should have a good understanding of the functional requirements for the system and the actors who will interact with the system. From this information, you can move on to an analysis of the system within its environment. The **system context diagram** is the appropriate tool to use for this structural analysis.

The system context diagram is a specialization of the class diagram described in *Object-Oriented Technology for Real-Time Systems*.

System context diagram elements

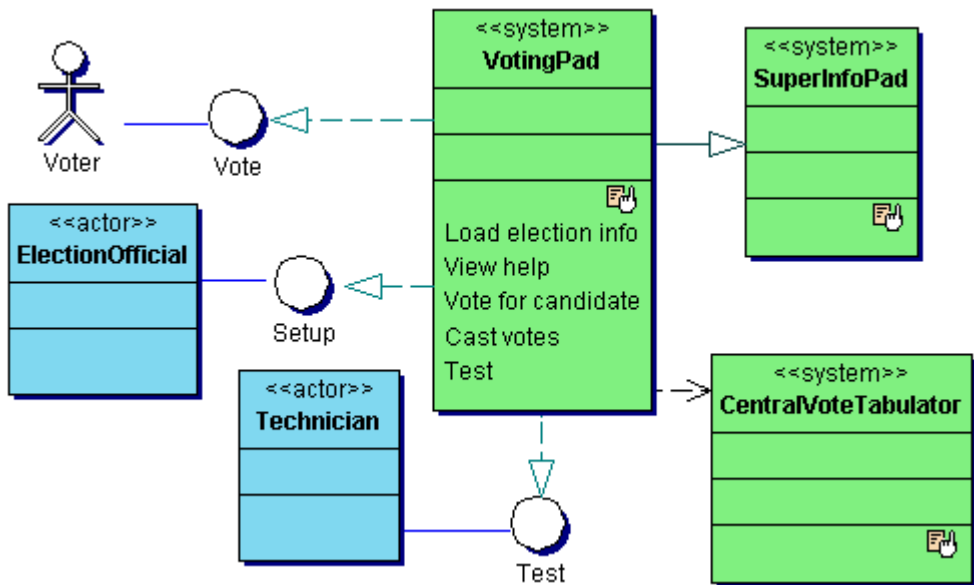
Table 27 lists the elements on the toolbar that you can place on a system context diagram.

Table 27 Elements of a system context diagram

Icon	Element Name	Description
	System	A single real-time system specified by its requirements. A system context diagram may include more than one system.
	Actor	An entity that interacts with a system for some purpose.
	Interface	A named interface through which actors or other systems may access a system.
	Association	This is a standard UML association. In a system context diagram, associations are most often drawn from actors to systems and from actors to interfaces.
	Generalization/ Implementation	This is a standard UML generalization/implementation link. Typically systems implement interfaces. Generalization is also used to describe relationship between actors and between systems to say, for instance, “This system is a more general version of this other system.”
	Dependency	This is a standard UML dependency link. In a system context diagram, dependencies are normally used only to indicate dependencies between systems.

Figure 42 shows the system context for an electronic voting slate named `VotingPad`. The `VotingPad` is a special application of the existing `SuperInfoPad`. The `VotingPad` depends upon the presence of a `CentralVoteTabulator` system. The `VotingPad` realizes five requirements over three interfaces. The diagram shows three actors, each using a different interface. Notice that an actor can be represented by a standard class rectangle or by an actor icon.

Figure 42 System context diagram



Real-time inspector properties

There are three special properties for elements on system context diagrams:

1. **trtelement** (on the Custom Properties tab): common to all Together real-time elements. This property makes it easier to write modules, patterns, audits, and metrics around the real-time module.
2. **use stereotype icon**: displays the element using its icon instead of the standard class rectangle.
3. **system requirements**: described below.

System requirements

Together Real-Time introduces the idea of specific requirements elements attached to a system. System requirements appear in the bottom compartment of a system.

There are two types of system requirements:

- **Functional Requirements**: describe responsibilities the system must fulfill.
- **Quality of Service Requirements**: serve as separate time-based constraints that can be applied to functional requirements.

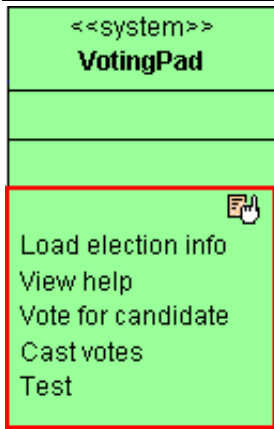
For example, the VotingPad system of Figure 42 has a Cast votes functional requirement. An analysis might also turn up several quality of service requirements to attach to Cast votes. These could be things like: Max 30 second response time or Minimum 10000 concurrent voters casting votes. These kinds of quality of service, or timing, constraints are at the heart of real-time development.

Creating a new system requirement

To create a new system requirement, right-click on the system to bring up its right-click menu. Select **New | Functional Requirement** or **New | Quality of Service Requirement**.

The new requirement appears in the **requirements compartment** of the system. Figure 43 shows the requirements compartment outlined in red.

Figure 43 Requirements compartment of a system



You can use the in-place editor (double-click on the requirement or press F2 when the requirement is selected) to change the name of the requirement. To edit any of its other properties, select the requirement and choose **Properties** from the right-click menu.

Properties of functional requirements

The properties of functional requirements are as follows:

- **name:** The name of the requirement. The name is also the text label that appears in the diagram.
- **description:** A text description of the requirement. This text appears in the generated documentation.
- **type:** Toggles the requirement between functional and quality of service. Changing the type of the requirement also changes the other properties available in the panel.
- **use cases:** Each functional requirement can be connected to one or more existing use cases. Use this property to associate the requirement with one or more use cases. This is a convenient mechanism for building requirements traceability into the real-time development process.
- **pre-conditions:** Pre-conditions in addition to those already included in the linked use cases.
- **post-conditions:** Post-conditions in addition to those already included in the linked use cases.
- **exceptional cases:** Details of any exceptional cases the for which the requirement may be responsible.

- **return value:** If specified at this level, the return type of the operation realizing the functional requirement.
- **input:** Input(s) used by the requirement.
- **output:** Output(s) returned by the requirement.
- **minimum response time:** The minimum time expected to elapse before the requirement completes. This property uses a scalar and units property field.
- **maximum response time:** The maximum time allowed for the requirement to complete. The maximum response time is a real-time deadline. This property uses a scalar and units property field.
- **rate of occurrence:** How frequently activities should happen and/or the rate of occurrence the system must support at the maximum response time. The rate of occurrence can be specified as:
 - when the system starts
 - periodically, with a set period
 - at a certain time

Properties of quality of service requirements

A quality of service requirement has a more limited set of properties than a functional requirement. Quality of service requirements are typically additional time-based constraints placed on a functional requirement. As such, a quality of service requirement has properties for specifying response times, for rate of occurrence, and for assigning the requirement to a particular functional requirement.

System architecture diagrams

System architecture diagrams model the divisions of systems into subsystems. Operations and responsibilities are attached to each subsystem. **Responsibilities** link requirements to specific operations in a subsystem, establishing traceability from use cases to specific operations.

A system architecture diagram typically starts with a shortcut to the system and any relevant context. Figure 44 shows an example of a system architecture diagram for a retina scanner, which is a networked biometric access system.

Figure 44 System architecture diagram

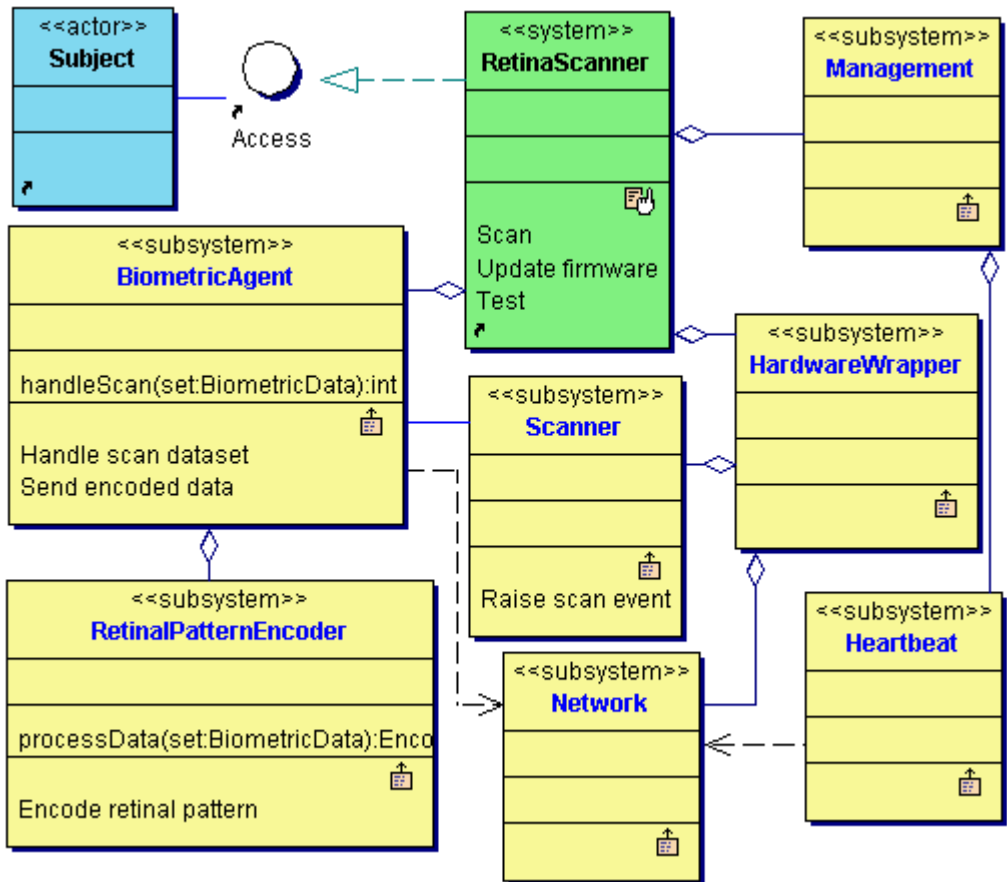


Figure 44 shows the retina scanner divided into three main subsystems:

- BiometricAgent
- Management
- Hardware Wrapper

The biometric agent uses the scanner driver, the RetinaPatternEncoder, and a networking subsystem to respond to a scan, to retrieve the raw scan data, to encode the data, and to send the data to the security system. The management subsystem is responsible for maintaining a network heartbeat. Both of these responsibilities are delegated to other subsystems. Finally, there is a hardware wrapper layer in which the scanner, network driver, and API modules reside.

Subsystems can have the attributes and operations available to classes in UML. Subsystems can also have responsibilities, which are shown in the bottom compartment along with the icon. Responsibilities define the functional bounds of the system. They tie requirements to operations. In this way a quality of service requirement specifying the frequency of the heart beat or dictating the maximum time allowed for the biometric

agent to handle a retina scan--from scan to network send--can be tied to the specific operations that implement the requirement.

Subsystems

The system architecture diagram includes a subsystem element as well as all the elements available in system context diagrams.

Table 28 System Context diagram element

Icon	Element Name	Description
	Subsystem	A module that is part of a system or another subsystem. Subsystems can have attributes, operations, and responsibilities.

A **subsystem** represents a module capable of fulfilling some number of responsibilities through specific operations. A system is made up of subsystems which, in turn, may be made up of still more subsystems. This relationship is most often shown as an aggregation link, as in Figure 44. If a subsystem uses another subsystem with which it does not have a containment relationship, you should indicate this with a plain association or with a dependency. See network subsystem in Figure 44 for an example.

NOTE: Subsystems are modules or components. Design and implementation occurs at the subsystem level. Together automatically creates a new package for each subsystem. The package includes a class diagram, but you may use other diagrams to describe the subsystem in future development phases.

Subsystem Properties

In addition to the standard properties and those common to real-time elements, discussed in “System context diagrams” on page 199, subsystems have two unique properties:

- **modes of reuse** enables you to label a system as:
 - new
 - reused
 - implemented
 - modified

These labels have no formal meaning within Together’s real-time module. They simply provide a means of documenting a subsystem’s status.

- **_referredPackage** (on the **Custom Properties** tab) holds the path to the package Together has associated with the subsystem.

Responsibilities

A subsystem is defined by its operations and responsibilities. Responsibilities provide the means for tracing an operation back to the requirements that drive it.

You can add a new responsibility to a subsystem by selecting **New | Responsibility** from its right-click menu. Use the in-place editor to change its name.

Responsibilities have two properties that allow them to act as the connection between operations and requirements:

- **requirements** points to zero or more requirements of a system in the diagram.
- **operations** points to zero or more operations of the subsystem to which the responsibility belongs.

Statechart diagrams

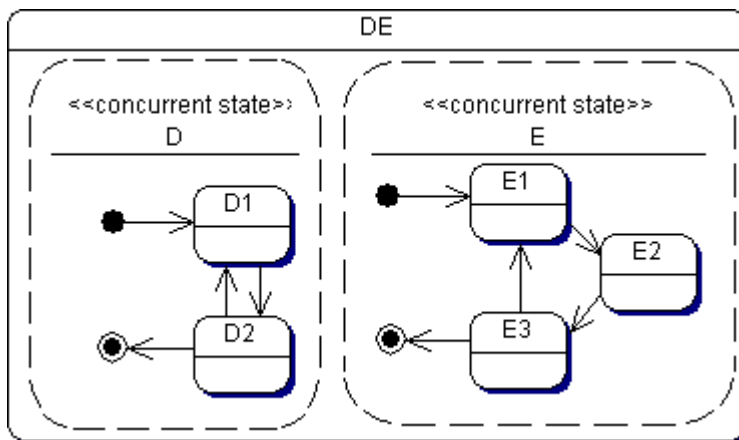
Together's real-time module adds one element to the standard statechart diagram, **concurrent state**.

Table 29 Special statechart element for real-time modeling

Icon	Element Name	Description
	Concurrent State	If the system is in a state with two or more concurrent states, then the system is in a combination of multiple substates, one from each concurrent state.

Figure 45 shows a statechart diagram with concurrent states.

Figure 45 Statechart diagram



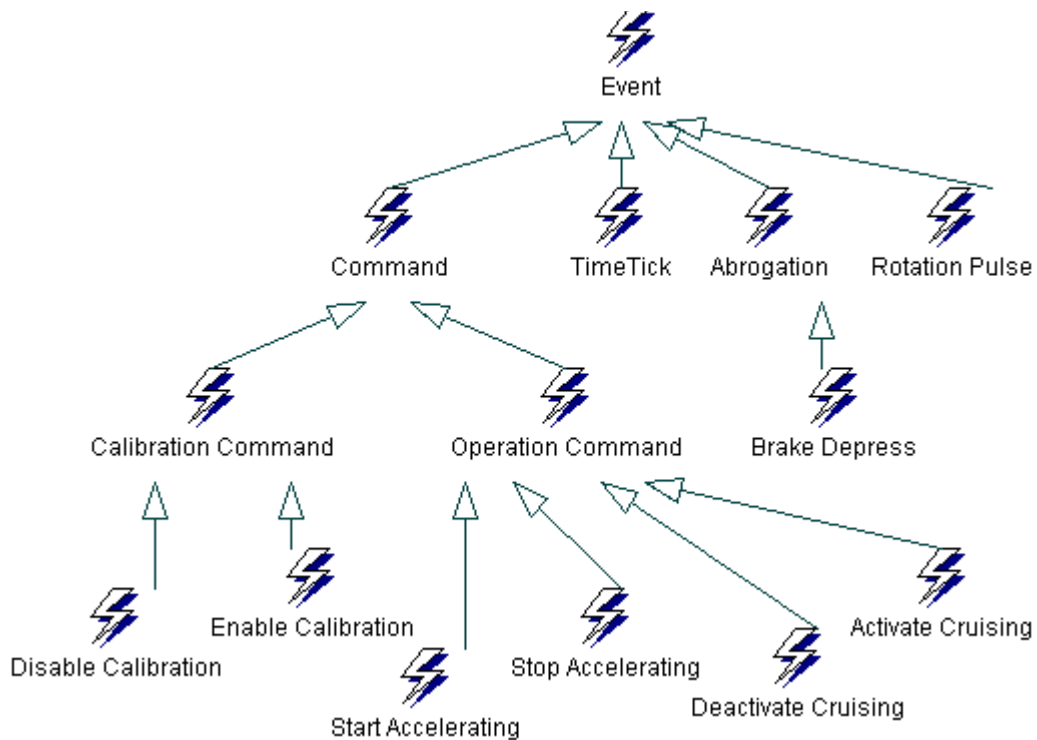
If a system such as DE in Figure 45 is in a state with two or more concurrent states, then the system is in a combination of multiple substates, one from each concurrent state. So, when the system enters state DE, it must be in some combination of a state in D (D1 or D2) and a state in E (E1, E2, or E3). For example, the state might be D2 and E3, but not E1 and E2.

Event sheet diagrams

Events are often the drivers behind actions in real-time systems. An **event sheet diagram** provides a single place to store information about events occurring in and around the system. It acts as a glossary of events, defining their properties and relationships to one another.

Figure 46 shows a sample event sheet diagram that describes events related to a cruise control system. This diagram is adapted from the cruise control example in *Object-Oriented Technology for Real-Time Systems*.

Figure 46 Event sheet diagram



Event Sheet diagrams have two event related elements: **Events** and **Generalizations/Implementations**.

Table 30 Event Sheet diagram elements

Icon	Element Name	Description
	Event	A particular event that could trigger action in the system
	Generalization/Implementation	Events may be generalizations (or specializations) of other events.

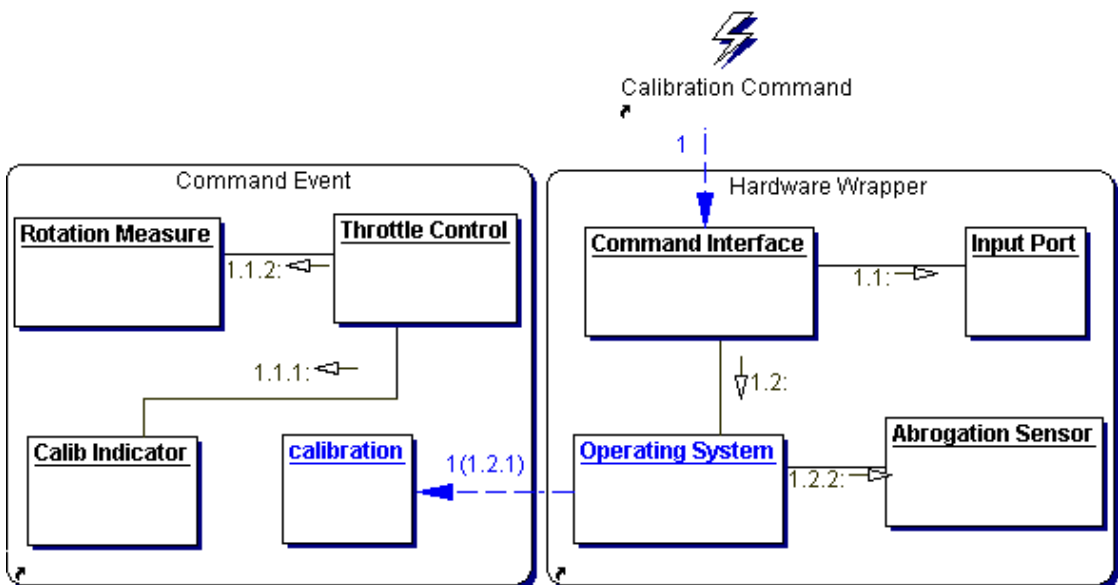
An event has one property: **rate of occurrence**. You can use rate of occurrence when simulating thread interaction in the presence of events.

Interaction diagrams

Real-time systems place special emphasis on the interaction of objects in different threads. Together's real-time interaction diagrams model object groups, or object interaction threads, and the events and messages that initiate program flow within a thread.

Often an event will cause a thread to execute. That thread may send asynchronous messages to other threads that will act on those messages. The interaction diagram is a blend of collaboration diagrams within object groups and the asynchronous communication between those groups.

Figure 47 Interaction diagram



The example diagram of Figure 47 has two object groups: Command Event and Hardware Wrapper. The Calibration Command event causes Hardware Wrapper to become unblocked at its primary wait point, a method in Command Interface. The Hardware Wrapper thread executes in response to the event and then returns to waiting on other events or messages.

As the thread executes, the Operating System object sends an asynchronous message to the Command Event group. That thread picks up the message at a wait point in the calibration object. That thread executes until it returns to wait on another message. From this example, it is easy to see that an interaction diagram consists of groups of objects that communicate internally with synchronous calls and between groups with asynchronous messages. Events are special kinds of asynchronous messages that come from outside of the system, or at least from outside the current diagram.

Interaction diagram elements

Interaction diagrams are composed of the elements listed in Table 31.

Table 31 Interaction diagram elements

Icon	Element Name	Description
	Group	Object interaction group, representing the objects in a single thread of execution. Each group is backed by a collaboration diagram that is automatically generated and kept in sync by Together. Objects in one group may communicate with objects in another group only by sending asynchronous messages.
	Object	An instance of a class. An object must be in an object group.
	Event	A shortcut to an event in the project. Events start the execution in an Interaction Diagram. In effect, they drive the action.
	Message	A synchronous message between objects in the same group. This is a normal method call.
	Self-Message	A normal method invocation from one method in an object to another (or the same) method in that object.
	Inter-Process Message	An asynchronous message sent from an object in one group to an object in another group.
	Process Delay Link	A simple execution delay. An object sets its own thread/group in a wait state for some time period.

Object groups

An **object interaction group** is made up of all of the objects that execute within a single thread. The objects within the group communicate with each other and themselves using normal synchronous method calls. Events and communications from other object groups must be asynchronous. Object groups have several properties that describe their thread interaction attributes:

- **priority**: thread priority from 1 to 32.
- **preemptiveness**: link to other object groups that this group can preempt.
- **thread safe**: group that cannot pass a message that involves delay or waiting. Blocking asynchronous calls and process delays prevent groups from being thread safe. Groups with only simple asynchronous messages that return immediately can have more than one thread executing through them at once.

Asynchronous messages

Together's real-time interaction diagrams support several kinds of asynchronous messages.

simple asynchronous

Simple asynchronous calls return immediately. The sender sends the message and does nothing else to make sure the receiver gets it. Asynchronous messages are usually

delivered through some kind of queue and are usually received at a primary wait point. See Figure 48.

Figure 48 Simple asynchronous message



Properties of simple asynchronous messages include:

- **return:** message return value
- **condition:** condition under which the message is called, usually a boolean value
- **iteration:** number of times to send the message or the iteration on which to send the message
- **execution time:** time, in ops, required for the receiving thread to execute in response to the message. (This property is used by the simulator.)

sender waiting reply

With a **sender waiting reply** message, the sender begins immediately to wait for a response. The sender waits indefinitely unless a timeout is supplied. See Figure 49.

Figure 49 Sender waiting reply message



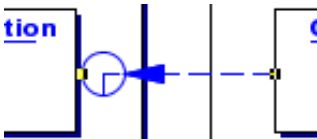
Properties of sender waiting reply messages include:

- **time out:** time the sender will wait for the message.
- **execution time:** same as for simple asynchronous messages.

receiver polling reply

A **receiver polling reply** message indicates that the sender is waiting for a response or a time out and branches depending on which one happens first. See Figure 50.

Figure 50 Receiver polling reply message

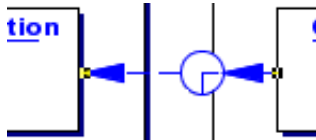


Properties of receiver polling reply messages are the same as for sender waiting reply.

time stamp

With **time stamp** message, the sender sends the message along with a timestamp and then waits for the message on a primary wait point. If the reply comes after the time stamp expires, the sender ignores the reply. See Figure 51.

Figure 51 Time stamp message

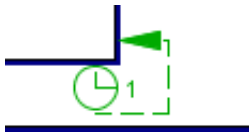


Properties of time stamp messages are the same as for sender waiting reply except that time out is replaced by time stamp (with the same meaning).

process delay link

A **process delay link** delays thread execution for a set period of time. This could be through some mechanism like blocking on a time notification or putting the thread to sleep. See Figure 50.

Figure 52 Process delay link



Properties for a process delay link include **delay**, which is the amount of time to delay.

Together real-time audits

Together's real-time module includes two new audits designed specifically for real-time projects:

1. Requirements Traceability -- RT: checks requirements traceability for each system.

- All system functional requirements/use cases should link to at least one subsystem responsibility.
- All responsibilities should belong to at least one system requirement.
- All responsibilities should link to at least one subsystem operation.
- All subsystem operations should belong to at least one responsibility in the subsystem.

2. Stereotypes Of a Diagram Subtype Audit -- SODS: verifies that only the correct stereotypes are used for a diagram sub-type.

- Only system and actor should appear on system context diagrams in Octopus.
- Only system, subsystem and actor should appear on system architecture class and sequence diagrams in Octopus.

To run the real-time audits, choose **Tools | Audit** from Together's main menu. Check one or both audits from the **Real Time** group, then click **Start**. Figure 53 shows the audit selection for Java projects with SODS selected.

Figure 53 Real-time audit selection

Title	Abbreviation	
+ Coding Style		☒
+ Critical Errors		☒
+ Declaration Style		☒
+ Documentation		☐
+ EJB Specific		☐
+ GUI Builder Specific		☐
+ Naming Style		☒
+ Performance		☐
+ Possible Errors		☒
- Real-Time		☒
Requirements Traceability	RT	☐
Stereotypes Of a Diagram Subtype	SODS	☒
+ Superfluous Content		☐

Simulation

Together's real-time module includes a concurrency simulator to aid in the design of thread concurrency schemes and thread priority. The simulator can help point out contentions that may cause performance issues, missed deadlines, and opportunities for concurrency design improvement.

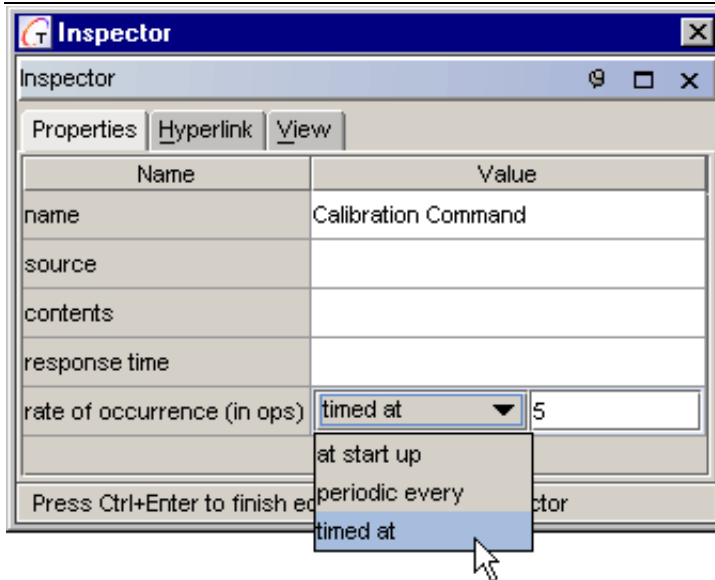
Preparing a simulation

Simulation requires an interaction diagram such as the diagram in Figure 47. Each object group in the diagram is a thread. The end product of a simulation run is a chart showing the triggering events, the messages passed between threads, and the effects of both on the states of the threads: inactive, running, ready, and completed. The smallest non-trivial simulation must include at least two object groups.

Event properties

Events are the driving force in the simulation. Each event causes a thread to run at a specified time or periodically throughout the run of the simulation. You can set the properties of an event by selecting **Properties** on its right-click menu. Figure 54 shows setting event properties for the real-time sample that ships with Together.

Figure 54 Setting event properties



The ops simulation unit

All time-based properties in Together's real-time module use the *ops* as a general unit of measurement. It is critical that ops assigned to properties are correct in terms of relative times required to complete the tasks or the times between events.

Rates of occurrence

Each event in the interaction diagram must have a configured *rate of occurrence* property.

- **at start up** - the event fires once at the start of the simulation.
- **periodic every** - the event fires at multiples of the assigned ops. For example, "periodic every 5 ops" means an event will fire at 5 ops, 10 ops, 15 ops, and so on.
- **timed at** - the event fires once at the assigned number of ops into the simulation run.

Interprocess messages

Each inter-process message has an *execution time* property. This property specifies the amount of time the target thread (object group) requires to respond to the message. If an inter-process message has no assigned time, the simulation treats the execution time as 0.

TIP: Meaningful simulation results almost always require that you assign an execution time to each message.

Preemption

The simulator can simulate cooperative multithreading using the default preemptiveness settings and process delay links. (The section “Interaction diagram elements” has a description of process delay links and other interaction diagram elements.) Modeling preemptive threading systems requires assigning object groups to preemptive possible groups. You can do this by using the *preemptiveness* property of an object group to assign other object groups to allow them to preempt the object group open in the inspector. Once groups are assigned to preemptiveness groups in this way, they can interrupt other threads as they execute, depending on thread priority.

NOTE: If you do not configure preemptiveness properties, all messages result in a thread finishing its response before other threads may run.

Priority

You can assign a thread a priority from 1 to 32. If you have already established preemptiveness groups, then priority determines whether or not a thread can interrupt another thread as well as which thread should execute first when two threads become ready at the same time. If preemption groups have not been established, the priorities are used only for scheduling two threads made ready at the same time. The Together real-time simulator uses a simple simulation model with no allowance for priority inheritance or other more complicated scheduling models.

Running a Simulation

To start the simulator, open the interaction diagram that you want to simulate. Then select **Tools | Real-Time | Simulate** from Together’s main menu.

Simulation results appear in the message pane under a new Simulator tab. If the interaction diagram is properly configured, the results show a thread execution chart explained in the next section, “Simulation results.”

Simulation controls

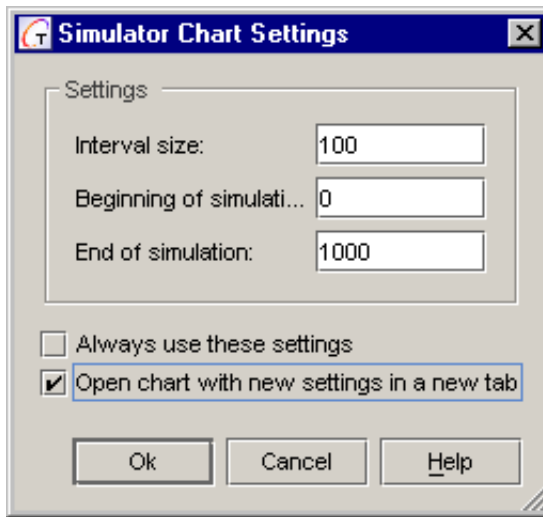
The Together real-time simulator has four controls located in the upper left hand corner of the simulation pane:

- **Run Simulation** - Runs the simulation again, including any changes made to the interaction diagram.
- **Stop Simulation** - Provides the ability to interrupt a simulator run (especially helpful for large simulations).
- **Save Chart as GIF Image** – Saves the chart as an image file.
- **Print** – Prints the chart.
- **Settings** - Pops up a small configuration panel described below.
- **Show/Hide Legend** - Toggle button to display or hide the chart legend at the bottom of the pane.

Simulation chart settings

Clicking the **Settings** button results in a dialog as shown in Figure 55.

Figure 55 Simulation chart settings



The Simulator Chart Settings dialog box has five settings. The first three settings allow adjustment to the simulator time scale:

- **Interval size** - The number of ops between time marks on the simulator chart.
- **Beginning of simulation** - The time at which to start the simulation chart.
- **End of simulation** - The op count at which to end the simulation chart.

The chart will run from beginning to end and it will be as long as end minus beginning. The end must be larger than the beginning. The interval must be greater than zero. All of the numbers must be positive.

The settings dialog box also includes two convenience checkboxes:

- **Always use these settings** - The specified settings become the new default settings for the simulator
- **Open chart with new settings in a new tab** - Provides an easy way to compare how charts look with different value ranges.

XML export

Together's real-time module provides the ability to export the interaction diagram simulator information to an XML file for use with other tools. To export the simulator information, you must first make sure that the interaction diagram to be simulated is open in the diagram pane. Then from the main Together menu, select **Tools | Real-Time | Export to XML**. Select a location for the exported file in the resulting dialog box.

Simulation results

The simulator displays a vertical chart of the thread execution states and thread interaction. From left to right, the elements in the chart are:

- **Events and Messages** - Events display with red arrows next to their names. Messages display with their operations or labels next to blue arrows. The arrows point to the time when the event occurred.
- **Simulator Timeline** - A numbered line showing marking off the ops from beginning to end at the specified intervals. (See “Simulation chart settings” for more information.)
- **Threads** - Shows the state of each thread over time:
 - **Inactive**
 - **Ready**
 - **Running**

The cyan blue “Completed” line appears when a running thread has finished executing its response to the event of message. The line is designed to make it easier to see when this has occurred versus a running thread being interrupted. It does not indicate a thread terminating.

CHAPTER 13

XML MODELING

In this chapter you will learn how to:

- Create an XML Structure diagram and populate it with elements, considering the two different formats.
- Go through the step-by-step example (“XML Modeling Step by Step”)
- Export the XML model to DTD or XSD Schema files, and import existing DTD and Schema files to the XML model (“Import and Export Operations”)
- Exchange information between the XML and data models using the DTD Interchange module (“XML Serialization”)

This chapter includes the following topics:

- “XML Structure Diagrams” on page 217
- “Import and Export Operations” on page 226
- “XML Serialization” on page 227

XML Structure Diagrams

XML structure diagram enables you to create an XML structure definition from scratch. The created XML model It is assumed that you are familiar with the basics of XML, since training is beyond the scope of this help topic. You can refer to the following books on XML:

- *Java and XML* (O'Reilly Java Tools) -- Brett McLaughlin, Mike Loukides
- *XML by Example* (By Example) -- Benoit Marchal
- *Professional XML* -- Mark Birbeck, et al

To learn more about XML Scheme, refer to XML Schema Part 0: Primer (02 May 2001, David C. Fallside), XML Schema Part 1: Structures (02 May 2001, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn), and XML Schema Part 2: Datatypes (02 May 2001, Paul V. Biron, Ashok Malhotra) at <http://www.w3.org/TR/>.

Alternatively, you can create an XML structure diagram by importing an existing DTD or XSD file. This can be very useful if you are not familiar with the DTD or XSD and want to get a quick overview of the elements it contains, and their relationships.

WARNING! If you want to use an XML structure diagram created in previous versions of Together, you have to convert the diagram to new format. Failing to do this results in wrong diagram behavior. This is how it's done:

- Right-click on the diagram background;
- Choose **Tools | Convert to New Diagram Format**.

Content of XML Structure diagram

Content of XML Structure diagram is format-dependent. There are two possible formats: DTD and XSD. As you can see, the sets of elements for DTD and XSD formats are overlapping: the majority of design elements are common. However, there are elements specific for a particular format, which are not enabled in another format. Thus, take care when exporting a diagram to a DTD or XSD file, to avoid loss of information.

Table 32 XML Diagram Components

Icon	Description	Format
	Element: Creates a visual design component of an element type.	
	Groups: Creates a visual design component of a model group.	
	Reference Link: Creates a visual design component of a link. The following cardinality values are possible for elements or groups of elements: • ? zero or one • * zero or more • + one or more • 1 required.	
	Attribute: Adds an attribute to an element. This design element can be created by a toolbar button, or from an element right-click menu.	
	Attribute Group: Creates a visual component for groups of attributes that are accessible for the elements of the diagram. This design component can be created by a toolbar button, or from an element right-click menu.	
	Data type: creates a visual design component for an arbitrary data type by means of inheritance (<i>type</i> field in the Properties tab of the inspector).	XSD
	Complex type: creates a complex data type component that allows elements in its content and may carry attributes.	XSD

Table 32 XML Diagram Components (continued)

	Entity: creates a visual design component for an entity.	DTD
	Notation: creates identification for external binary entities format.	DTD
	Notes	
	Note Links	

Creating XML Structure Diagram

XML Structure diagram is implemented as an activatable module. Thus, make sure that *XML Structure diagram* flag is checked in the list of Activatable modules.

To activate XML Structure diagram module:

1. On the main menu, choose **Tools | Activate/Deactivate Features** command.
2. In the Activatable/Deactivatable Features dialog, choose Together Features tab.
3. Check *XML Structure diagram* flag, and click **OK**.

To create an XML Structure diagram:

1. On the main menu, choose **File | New** to open the Object Gallery dialog.

TIP: alternatively, select a package in the Project Explorer, and choose **New | Diagram** on its right-click menu.

2. In the General category, choose **Diagram** icon, and click **Next**.
3. In the New Diagram dialog, click on the Together tab.
4. Choose XML Structure icon, enter package name, diagram name and description if necessary, and click OK to complete.

NOTE: You can also edit the diagram properties in the inspector. To do this, choose **Properties** command on the diagram right-click menu.

Creating design elements of XML Structure diagram

The basic technique of creating diagram elements is the same as for the other diagrams. There are several ways to create design elements:

- Use the toolbar buttons
- Choose on of the commands from the **New** node of the diagram or element right-click menu.
- Use the keyboard shortcuts, listed in the **New** node of the right-click menu.

To create an element, attribute, group etc.:

1. Click the required button on the diagram toolbar.
2. Click on the diagram background to create a design element.
3. In the in-place editor, type in the name.

NOTE: When an attribute or attribute group is created this way, the resulting design elements are Global attributes, which serve as reusable components of the diagram.

To create a link:

1. Click the Reference Link button on the diagram toolbar
2. Click on the source element.
3. Drag and drop to the target element.
4. In the in-place editor, specify the cardinality of the link.

TIP: you can also choose the cardinality value from the right-click menu of the link.

To create an attribute, attribute reference or attribute group reference of the elements:

1. Right-click on the target element or data type.
2. On the right-click menu, choose **New | Attribute** (or Attribute Reference, Attribute Group Reference).
3. In the in-place editor, type its name.

TIP: It is also possible to choose a toolbar button and click on the target diagram element.

NOTE: You can observe and edit the properties of the design elements in the inspector. To do this, choose **Properties** command on the right-click menu of a design element and do all necessary editing in the inspector.

Format of XML Structure Diagram

New XML Structure diagram is created in DTD format by default. However, you may need to change the diagram format. So doing, keep in mind that the design elements and the set of toolbar buttons depend on the selected diagram format.

To change the diagram format:

1. Choose **Properties** command on the diagram right-click menu, to open the inspector.
2. Open Properties tab of the diagram inspector.
3. In the *format* field choose DTD or XSD Schema from the drop-down list.

When converting from one format to another a warning message is displayed that the elements irrelevant to the selected format will be permanently deleted from diagram. For example, if DTD format is selected, entities and notations disappear from the resulting diagram.

The reversed situation is slightly more complicated. Complex types and data types are deleted. If there are elements that inherit certain data or complex types, then their attributes (if any) add to those elements in the resulting diagram.

Example:

1. Create XML diagram in XSD format.
2. Add an element (Element1) and a complex type (ComplexType1).
3. Add an attribute to the complex type (Attribute1).
4. Select the Element1 and change its base type (field base) in the Properties tab of the inspector to ComplexType1.
5. Change format from XSD Schema to DTD, and observe that Attribute1 adds to the Element1 in the resulting diagram.

Working with DTD-specific components

Entities, notation and processing instructions are specific for DTD format only. Entities and notations are created in a usual way using toolbar buttons. Properties inspector for an entity provides *external* checkbox. When this flag is set, additional controls for external entities show up. In particular, there is a reference to a notation.

To assign a notation for an external entity:

1. Open the Properties inspector of an external entity
2. On the *notation* field of the Properties inspector, click **File Chooser** button to open the Select Notation dialog.
3. Choose the required notation from the Model, Search/Classpath or Favorites.

Processing instructions are defined for the entire XML structure diagram in DTD format.

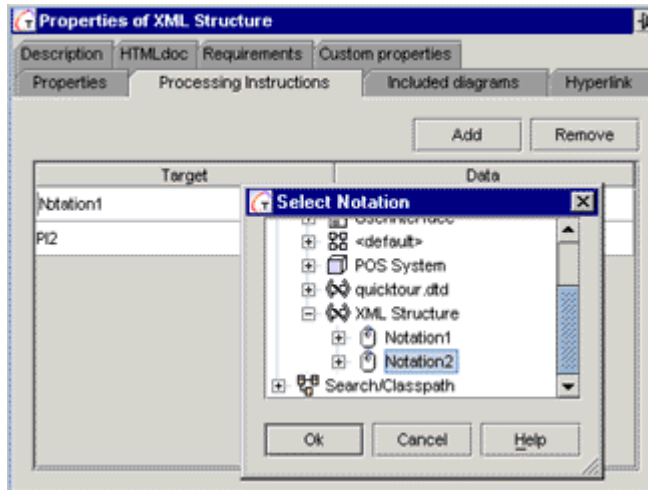
To define processing instructions:

1. Right-click on the diagram background
2. On the diagram right-click menu, choose **Properties** command
3. In the Inspector, select Processing Instructions tab.
4. Click **Add** button.

To specify notation for a processing instruction:

1. In the Processing Instructions tab of the inspector, select the processing instruction.
2. Click **File Chooser** button to open the Select Notation dialog.
3. Select the required notation.
4. Enter appropriate information in the *data* field.
5. Press Ctrl+Enter to apply the changes and close the Inspector.

Figure 56 Processing instructions and notations



XML Modeling Step by Step

Let us consider creation of a sample XML model of the personnel management data in XSD format.

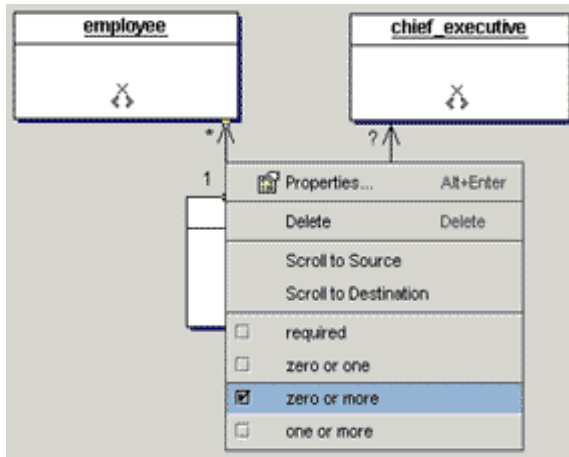
Creating XML Structure diagram

1. Activate the module *XML Structure Diagram*: On the main menu choose **Tools | Activate/Deactivate Features**, and make sure that XML Structure diagram flag is checked.
2. Create a new XML diagram in XSD format: On the package right-click menu choose **New | Diagram** and select XML Structure diagram in the Object gallery.

Creating elements, links, and attributes

1. Create an element and change its name to *personnel*. Create two elements and call them *employee* and *chief executive* respectively. Open the Inspector for each element and see that the property *content* is set to empty.
2. Create reference links from *personnel* to *employee*, and from *personnel* to *chief executive*. Right click on the *employee* link and check the box *zero or more* in the link's right-click menu, which means that there can be any number of employees. Click on the *chief executive* link and check the box *zero or one*, to specify that there can be one chief executive only.

Figure 57 Creating diagram elements and links



3. Add an attribute to each element. Select **New | Attribute** on the right-click menu of this element, or click the *Attribute* icon on the toolbar and then click inside the element to place the attribute there; change its name to *identifier*. This attribute is a unique identifier of an employee. In order to verify or change the attribute's properties, invoke its right-click menu, select *Properties*, click the *Properties* tab and set the field *occurs* to *required*, which means that this property is mandatory.

Besides unique identifier, each employee and the chief executive are characterized by name, position in the organization's hierarchy, URL, email address and postal address. Hence, create the following elements:

- *url* to a file that contains additional data, for example some graphics information.
- *hierarchy* with the attributes *subordinates* and *supervisor*. The *subordinates* attribute should have data type *idrefs*, which means that a person may have a number of people supervised; the field *occurs* of *Properties* is set to *optional*. The attribute *supervisor* should have data type *id*, which means that a person may have one immediate supervisor only; the field *occurs* is set to *optional*. Both properties are optional.
- *personal*, with the name and date of birth. The attribute *Date of birth* is mandatory (field *occurs* in *Properties* is set to "required"), and the field type is set to "CDATA".

NOTE: To insert data type of an attribute, in the *Properties Inspector* click *File chooser* button and select appropriate type in *Search/Classpath | XML data types* directory.

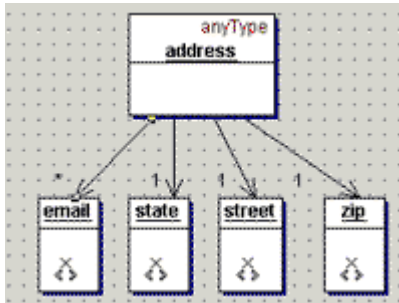
Open right-click menu of each element and see that the property *content* is set to empty.

Creating and using complex type and data type

Complex type allows to provide special customized data types containing internal elements. Consider an address type that includes zip code, state, street and several email addresses.

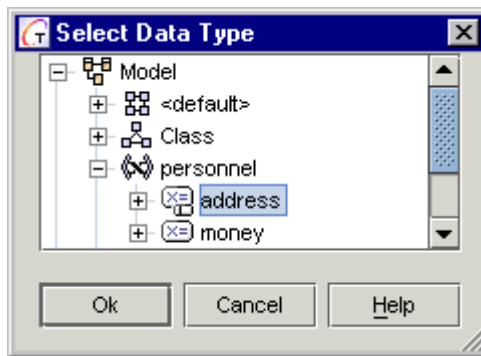
1. Create a **Complex type** visual component. Rename it to *address*. Next, create elements *email*, *state*, *street*, and *zip*, and draw links from *address* complex type to these elements. Set appropriate cardinality values for the links. *State*, *street* and *zip* code are *required*, while the link to *email* has *zero or more* cardinality, which means that a person may have unlimited number of email addresses.

Figure 58 Creating *Address* complex type



2. Now create an element *address*, open its properties inspector, select *base* field in the *Properties* tab and click *File chooser* button to open the Select Data Type dialog. Expand the *Model* node and select *address* type under *personnel* diagram:

Figure 59 Choosing data type from the Select Data Type dialog

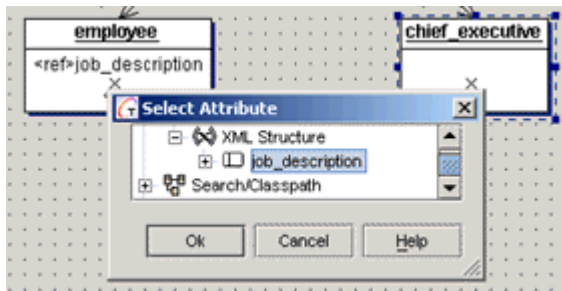


3. Create a new data type, and rename it to *money*. In the *Properties* tab of the inspector click *File Chooser* button to select data type from the Model, Search/Classpath or Favorites.
4. Create *salary* element. In the *Properties* tab of the inspector click *File chooser* button of the *base* field, and select *money* type under *personnel* node of the Model tree. You can use *derivation type* field to select the desired type of inheritance. Each derivation type displays its own set of controls. If you select *restriction*, you can assign minimum salary value.

Creating re-usable (global) attributes

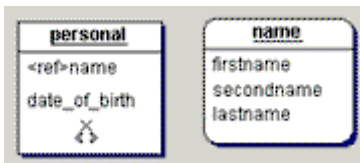
1. Choose *Attribute* button on the diagram toolbar and add *job_description* attribute to the diagram. All elements of the diagram can refer to this attribute.
2. Add this attribute to the elements *employee* and *chief executive*. To do this, choose **New | Attribute Reference** on the right-click menu of the appropriate element.
3. In the Select Attribute popup window choose the reference from the model and search/classpath tree. Same way, you can create global attribute *comments*, and add it to any element where needed.

Figure 60 Applying a global attribute to the diagram elements



4. Create the global group of attributes and call it *name*. Add the attributes *first name*, *second name*, *last name* to this group. All these attributes have String data type. *Firstname* and *lastname* are mandatory (the field *occurs* is set to required), while the attribute *second name* is optional (the field *occurs* is set to optional).
5. Apply this attribute group to the *personal* element. To do this, open its right-click menu, and choose **New | Attribute Group Reference**.

Figure 61 Applying Attribute group reference



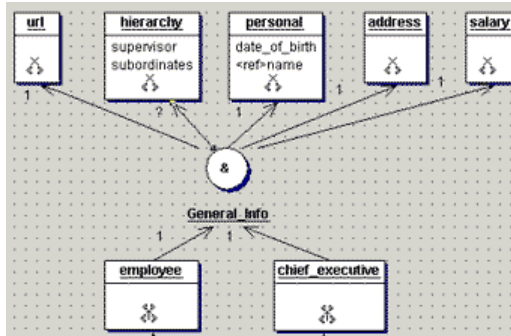
Creating groups

You can try to create links from the *employee* and *chief executive* elements to each of the elements *hierarchy*, *url*, *personal*, *address*, *salary*. However, the result looks too entangled. The more elegant way to link multiple elements to multiple elements lays with the Groups.

1. Create a new group *General_Info*. By default Sequence group is created. Link the *employee* and *chief executive* elements to this group. Open the right-click menu of *employee* link, and set the flag *occurrence* to required.

2. Link the group to each of the above elements. By default, the links have the property zero or one. Select the link to *personal* element and in the Properties set the flag *occurrence* to required, which indicates that personal information is unique and mandatory. At last, select the link to the *hierarchy* element and set the flag zero or one in its right-click menu, thus indicating that a lucky one can easily do without any subordinates or supervisors.

Figure 62 Linking diagram elements to a group of common information



Import and Export Operations

When XML structure diagram is ready, it can be stored in the local file system. So doing, the user has the choice to save the XML model in DTD or XSD format. Once created, the saved model can be imported for further use. As of this writing, the exported DTD and XSD files are stored under \$TGH\$/out/dtd or \$TGH\$/out/xsd respectively.

DTD/XSD Import-Export

To export a diagram to a DTD or XSD file:

1. On the main menu, choose **File | Export | XML diagram to DTD/XSD file**.
2. In the opened Export XML Structure Diagram dialog, click the **DTD** or **XSD** radio button to choose the target format.
3. Click **Edit** button, if you need to modify the exported file in the Edit File dialog, and click OK to return to the Export dialog.
4. Click **OK** to save the exported file in the default location.

NOTE: The generated DTD or XSD Schema file is stored by default under \$TGH\$/out/dtd/<ProjectName>/<DiagramName.dtd> or \$TGH\$/out/xsd/<ProjectName>/<DiagramName.xsd> respectively.

To change the target location:

1. Open the properties inspector of the XML Structure diagram to be exported.

2. Select the Properties tab.
3. In the file name field, specify the target location of the DTD or XSD Schema file.

To import a DTD or XSD file from the local file system:

1. On the main menu, choose **File | Import | DTD/XSD file**
2. In the Import DTD/XSD File dialog, select the required type from the *Files of type* combo box. The files of the selected type display in the tree-view.
3. In the tree-view, choose the source file and click **Import**.

WARNING! When exporting or importing DTD or schema files, keep in mind that some components are only enabled for the specific format. In particular, complex type and data type are allowed in XSD format, while entities, notations and processing instructions are allowed in DTD only. Check view formats of the diagram components to avoid loss of information.

XML Serialization

Together supports exchange of information between the data models and XML models. You can export an existing Class diagram, or database structure to a DTD file, and import information from DTD.

This exchange of information is implemented as an activatable module *DTD Interchange*.

To activate DTD Interchange module:

1. On the main menu. choose **Tools | Activate/Deactivate features**.
2. Choose Together Features tab.
3. Check the box **DTD Interchange**.

Result: XML Serialization node adds to the Tools menu. This node contains the following commands:

- Export Classes to DTD
- Export ER Entities to DTD
- Import Classes from DTD
- Import ER Entities from DTD

NOTE: These commands are only enabled when the diagram pane gets the focus. The set of enabled commands depend on the current diagram. For example in a Class diagram *Export ER Entities to DTD* command is disabled, and vice versa.

Export commands open *Export* dialogs for appropriate diagrams. The dialogs differ only in title. The DTD file is stored at `$TGH$/out/dtdinterchange/project_name/dtdinterchange.dtd` by default.

To export a class diagram to DTD:

1. On the main menu, choose **Tools | XML Serialization | Export Classes to DTD** command.

2. In the Export dialog, specify the target location.
3. If needed, set the flags to immediately create an XML structure diagram and open the generated dtd file in Together's internal editor.

The reverse operation is to import information from a dtd file to a Class diagram, or to an ER diagram. Import commands open the File Chooser dialog, where you have to select the source dtd file.

PROGRAMMING WITH TOGETHER

- Chapter 14, “Working with Code”
- Chapter 15, “Using the UI Builder”
- Chapter 16, “Using Version Control”

CHAPTER 14

WORKING WITH CODE

This chapter covers the features that allow users to work with the source code: editing, searching through the code, debugging, compiling and running. This chapter includes the following topics:

- “Editing” on page 230
This section covers the general issues of working with the Editor: configuration, basic techniques, and special features. Note that editing is not confined to the plain text or java source code. It is also possible to work with HTML, JSP and XML files (See “Integrated XML Editor” on page 244).
- “Search Facilities” on page 252
This section provides information about the search methods supported by Together (searching in files and diagrams, search for usages, creating queries and so on).
- “Compile - Make - Run” on page 258
This section describes the actions to be performed to launch applications and create makefiles on the various levels.
- “Debugging” on page 263
This section provides a detailed description of the integrated java debugger. You will learn how to start the debugging session, watch and modify the variables and expressions, step through the breakpoints and so on.

Editing

Together comes with a built-in full-featured editor that allows to work with all supported languages. The editor is powerful and flexibly configurable.

Configuring the Editor

You can configure the Editor using the Options dialog at any of the multiple configuration levels. On the main menu, choose **Tools | Options | [level]** where **level** is the configuration level at which the settings will apply (Default, Project or Diagram).

NOTE: Alternatively, choose Text Editor Options on the right-click menu of the Editor pane.

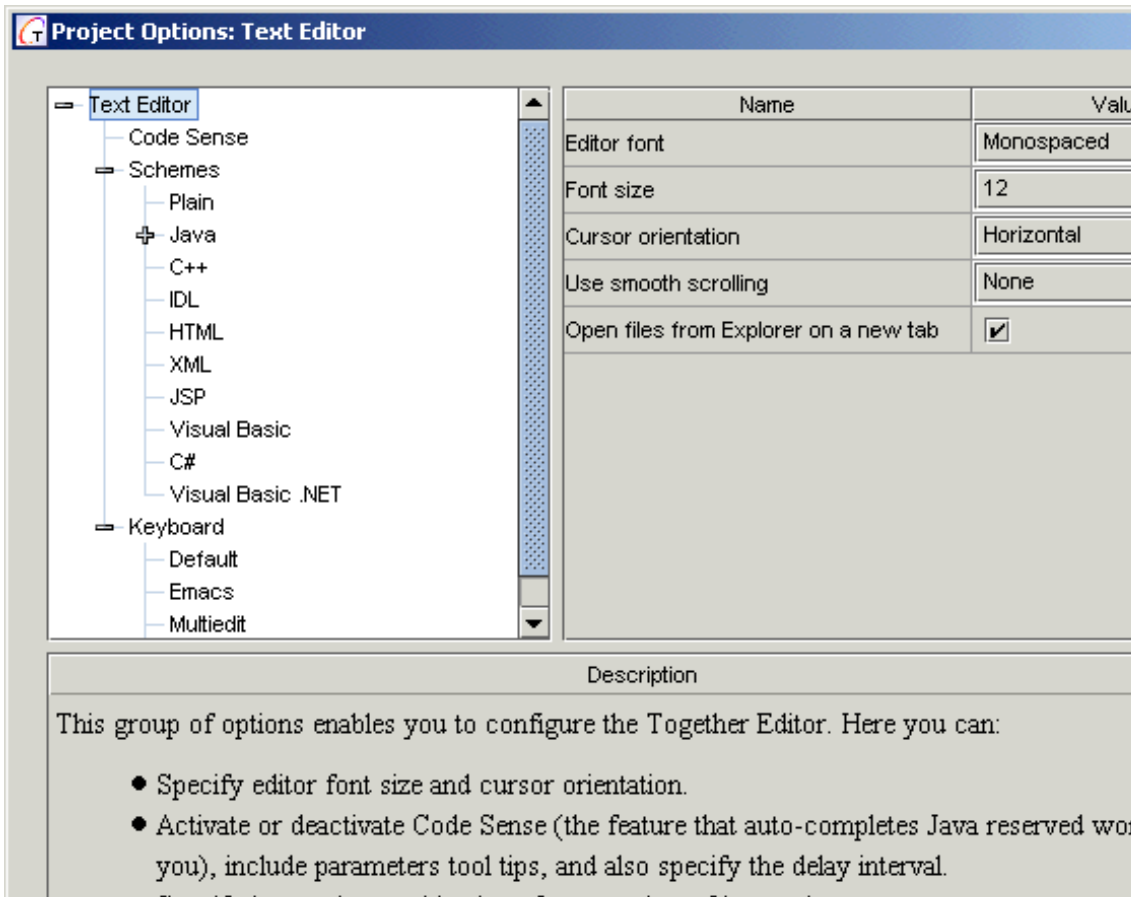
Navigate to the **Text Editor** node. You will see the following nodes representing the main categories of configuration options for the Editor:

Code Sense: auto-completes reserved words in Java source code.

Schemes: enables you to customize colors, indentation, text formatting and error highlighting. Use these options to tune the Editor for working with different contents (plain text, Java, C++, Visual Basic, IDL, HTML, JSP, C#, Visual Basic.Net). The Editor automatically picks up the particular scheme depending on the type of file loaded and focused in the Editor.

Keyboard: enables you to customize Editor hot-key combinations. Expand this node to view the current settings for keyboard shortcuts that invoke Editor commands. You can redefine the settings to fully customize keyboard shortcuts for the Editor.

Figure 63 Editor Configuration options



Using the Editor

You can open one or more files for editing. There are several ways to open files for editing:

- Select a file name in the Explorer and choose **Edit** on its right-click menu. The file opens in its own tab in the Editor pane. (Only source code, configuration, and properties files have these commands on their right-click menus.)

TIP: Opening files in new tab is a configurable option, assumed by default (*Options | Text Editor - Open files from Explorer on a new tab*). If this option is off, the files open in the current tab, replacing its content.

- Select a file name in the Explorer and choose **Tools | External Editor**. The file opens in the application configured as External Editor in Options: Tools.

- Choose **File | Open** on the main menu and select the file from the Open File dialog. The file opens in the Editor pane. (File must be text, not binary.)
- Click on a class (interface, link, member, method) in the Diagram pane. The source code opens in the Editor pane tab, highlighting the line that corresponds to the selected element.

As noted above, source code files automatically open, replacing the contents of the active tab, when you select source-generating elements in diagrams (classes for example). You can override this default behavior.

To preserve the current tab in the Editor pane:

1. In the editor pane, select the tab to be preserved.
2. On the tab's right-click menu, choose Preserve Tab command, while the active tab is selected. The current tab remains and a new tab appears named <untitled>. Your next selection of a source-generating element in the diagram opens its source in this new tab (or you can open some other file using the pane right-click menu).

Checking Preserve Tab for a selected tab does not prevent you from closing it later. It only means that if you open another file while the tab is active, the other file will open in a new tab thus preserving the flagged tab. This enables you keep several source files from the same diagram open in the Editor pane.

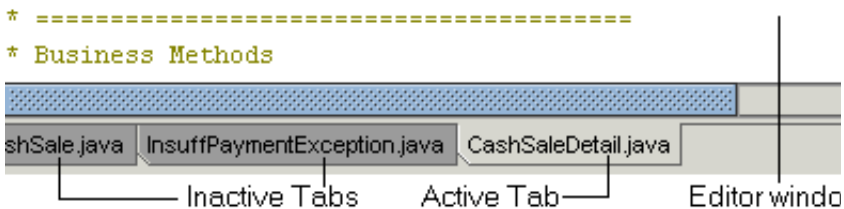
You can switch between tabs by clicking on tab's header, located right under the Editor.

To close a file:

1. In the editor pane, select the tab of the file to be closed.
2. On the tab's right-click menu, choose **Close** command.

TIP: Close All closes all opened files.

Figure 64 Tabs in the Editor window



Editor commands are available on the Edit menu, on the right-click menu and toolbar of the Editor pane. To speed up access to certain commands, use hot-key combinations.

Edit menu commands

Table 33 Edit menu commands

Command	Description	Hot-key
Undo	Rolls back last several changes.	Ctrl-Z
Redo	Reinstates last Undo operations.	Ctrl-Y
Cut	Cuts selected text to Clipboard.	Ctrl-X
Copy	Copies selected text to Clipboard.	Ctrl-C
Paste	Pastes text from Clipboard at the cursor position.	Ctrl-V
Copy Image	Copies the current diagram to Clipboard in the selected format.	(none)
Delete	Deletes selected text. (Restore with Undo)	Del
Select All	Selects all the text in the currently focused file	Ctrl+A
Insert Text File	Displays standard "Open File" dialog to pick up the file containing the text to be inserted.	(none)

Editor right-click menu commands

The Editor right-click menu contains a number of commands, some of which are self-explanatory or highly intuitive (Save, Save As, Cut, Copy, Paste, Select All, Close, Close All, while others require some brief explanation.

Table 34 Commands of the Editor right-click menu

Command	Description
Select in Diagram	Highlights the diagram element that corresponds to the current line of the code.
Browse Symbol	Displays the source code of project and library classes.
Collapse lines	Collapses groups of lines of the specified type (comments, imports, methods, fields) and displays the first line in the group marked with + sign.
Expand lines	Expands the nodes of the specified type (comments, imports, methods, fields).
Tools	Contains a sub-menu with a list of tools you can apply to active tab. You can configure tools using Tools tab of the Options Dialog
Tool Format source	Performs syntax formatting of the source code in the active tab
Tools External Editor	Opens the content of the active tab in the pre-configured external editor.

Bookmarks

Together provides two types of bookmarks: global bookmarks that apply to the entire project, and local bookmarks that are used within one file only.

You can set global bookmarks in your source code files and navigate to them from any open file that is part of your project. You can view, edit, classify, and navigate to bookmarks in the project using the Edit Bookmarks dialog, which is available on the right-click menu of the current line.

NOTE: The menu command displays a keyboard shortcut which is user-defined in Text Editor options. Default shortcut is Ctrl+M.

Once set, the bookmarks are stored in the project profile and will be re-used when the project opens next time.

Bookmarks have lower priority than the breakpoints. It means that if you set a breakpoint and a bookmark on the same line, the breakpoint will override the bookmark and executable line icon.

The other type of bookmarks provided by *Together* are the “Local bookmarks”. It is slightly different from the global bookmarks in scope of operation and the way of usage. Global bookmarks are project-wide, while the numeric bookmarks are only valid for the currently opened file.

Global Bookmarks

Setting and removing global bookmarks

To set global bookmarks:

1. Open the file to be bookmarked in the Editor.
2. Scroll to the line where you want to set a bookmark and place the insertion cursor anywhere on the line.
3. On the right-click menu of the line, choose **Toggle Bookmark**, or use keyboard shortcut.

The default bookmark icon displays in the margin. A default bookmark description is saved using up to the first 50 characters of the line. You can edit this description if you wish (see “Viewing, editing, and classifying global bookmarks”).

To remove a bookmark:

1. Navigate to the line in the open file where the bookmark has been set.
2. On the right-click menu of the current line choose **Toggle Bookmark**.

TIP: The following are additional notes for bookmarks:

- You can also remove bookmarks using the Bookmarks dialog (see next section).
- Undo/Redo does not apply to bookmarks.
- By default Ctrl+M shortcut is used to toggle bookmarks.

Viewing, editing, and classifying global bookmarks

The Edit Bookmarks dialog displays a list of all the bookmarks currently set in files in the current project. In this dialog you can:

- Edit the default bookmark description

- Reorder the list of bookmarks
- Visually classify bookmarks using different icons
- Delete bookmarks

To open the Edit Bookmarks dialog:

1. Open a file in the Editor.
2. On the right-click menu, choose **Edit Bookmarks**.

TIP: By default Ctrl+D shortcut is used to open Edit Bookmarks dialog.

Editing global bookmark descriptions

By default, the *Description* field for a bookmark contains the characters of the bookmarked line (up to maximum 50 characters). This field allows in-place editing.

NOTE: You cannot edit the *Line* and *File* fields.

Reordering bookmarks

To change the order of bookmarks in the list:

1. In the Edit Bookmarks dialog, select the bookmark to be repositioned.
2. On the right-click menu of this bookmark, choose **Move-Up** or **Move-Down** command.

TIP: There are alternative ways to move bookmarks:

- On the toolbar of the Edit Bookmarks dialog, click up-arrow or down-arrow icons.
- Use keyboard shortcuts Ctrl+Up or Ctrl+Down.

Classifying global bookmarks

The Edit Bookmarks dialog enables you to classify the various bookmarks in your project using various icons. You can use this icon set to devise any sort of classification scheme that is meaningful to you.

For example, you might use one icon only for bookmarking constructors, another for start of business methods, another for JavaBean getter/setter methods, etc.

To change associated icon of a bookmark:

1. Choose the required icon in the Edit Bookmarks dialog.
2. In the *Icon* field's right-click menu, choose **Change Icon** node.
3. Select the required icon from the drop-down list of available icons.

The icon you select for each bookmark displays in the margin of the Editor next to the bookmarked line. Now you can organize the bookmarks as required. One way to sort the bookmarks is in the order stipulated by the list of icons.

To sort the bookmarks:

1. Open Edit Bookmarks dialog.

2. On the right-click menu of the table of bookmarks, choose **Sort by Icons** command.

Deleting bookmarks

You can delete one or several bookmarks from the list.

To delete bookmarks:

1. Open Edit Bookmarks dialog.
2. Select the bookmarks to be deleted from the list.
3. On the right-click menu, choose **Delete** command.

TIP: There alternative ways to delete bookmarks:

- On the toolbar of the Edit Bookmarks dialog, click waste-bin button.
- Use Delete keyboard shortcut.

Navigating with global bookmarks

Once bookmarks have been set, you can use them to navigate from any open file in the project, to any bookmarked line in other project files.

To navigate to a bookmarked line:

1. Open the Edit Bookmarks dialog (on See “Viewing, editing, and classifying global bookmarks” on page 235.as previously described).
2. Select the target bookmark in the list.
3. Click the **Go To** button.

If the target file is not open, it opens in a new Editor tab and the cursor moves to the bookmarked line. No diagram is opened... just the file.

If the target file is already open in the Editor, the insertion cursor moves to the start of the bookmarked line.

If you have a small number of bookmarks, you can navigate within a single file using hotkeys. Ctrl/Grey+ moves forward, Ctrl/Grey- moves backwards. Hotkeys do not work for the entire project.

Local bookmarks

Setting and removing local bookmarks

Local bookmarks are very fast and handy to operate. They are numeric, in the sense that they are numbered from 0 to 9. Hence, there can be only ten local bookmarks per file.

To set local bookmarks:

1. Open the file to be bookmarked in the Editor.
2. Scroll to the line where you want to set a bookmark and place the insertion cursor anywhere on the line.
3. Press Ctrl+Shift+number.

The numbered bookmark icon displays in the margin. Each new bookmark gets the next number. The local bookmarks do not add to the list of bookmarks and do not display in the Bookmark dialog.

To remove a bookmark:

1. Navigate to the line in the open file where the bookmark has been set.
2. Use the same keyboard shortcut: Ctrl+Shift+number displayed on the bookmark icon.

Navigating with the local bookmarks

Navigating with the local bookmarks is nice and easy. All you have to do is press CTRL+number, and the cursor moves to the start of the bookmarked line.

TIP: Local bookmarks work with the main keyboard only. Small numeric keypad buttons produce no result.

Breakpoints

Breakpoints specify where to stop code execution during debugging to permit inspection of variables, expressions, class members, etc. This feature of the integrated Debugger is accessible from the Editor. Find a brief outline of the basic functions here and refer to the “Debugging” section for in-depth information.

Setting and removing breakpoints

There are several ways to set breakpoints in the Editor. In every case, place the insertion point cursor at the beginning of the line of code where the execution should stop and then...

- Click on the extreme left margin next to the line, or...
- Press F5, or...
- Choose **Toggle Breakpoint** from the Editor right-click menu.

When a breakpoint is set, a red rectangle appears in the extreme left margin, and the entire line is highlighted red.

To remove a breakpoint, place the insertion point cursor at the beginning of the line that has the breakpoint, then use any of the techniques listed above to remove the breakpoint.

Working with breakpoints

Once set, a breakpoint can be disabled or re-enabled using the right-click menu commands **Disable Breakpoint** and **Enable Breakpoint**. Disabled breakpoints display as grey rectangles.

You can edit properties of a breakpoint by placing the insertion point in its line and choosing **Breakpoint Properties** command on the right-click menu.

Related topic:

“Using the Integrated Debugger”

Advanced Editor Features

The Editor's toolbar provides a number of buttons that significantly speed up the coding process. These buttons are:

- **Code completion:** Code Sense and Advanced Code Sense allow users to automatically complete the code. See “Code Completion” for details.
- **Parameter tooltip:** Displays the possible lists of parameters of the method under the cursor.
- **Surround with...:** Surrounds the selected lines of code with one of the specified constructions (for, if-else, try-catch).
 - Select the required lines of code.
 - Click **Surround** button on the Editor toolbar.
 - Choose construction from the list.
- **Toggle comments:** Allows users to comment and uncomment the selected line of code.
 - Select the required lines of code.
 - Click **Toggle Comments** button on the Editor toolbar.
- **Override/ implement methods:** Allows users to choose methods of the parent class to be implemented or overridden.
 - Click **Override/Implement** button.
 - In the dialog window, choose the method to be overridden or implemented.
- **Expand snippet:** Allows users to expand one of the pre-defined snippets. See “Snippets” for details.
- **Browse symbol:** Allows users to view the source code of the library classes.
- **Previous/Next Declaration:** Allows users to navigate through the list of declarations within the current class.

Code Completion

Smart code completion feature includes two functions - Code Sense and Advanced Code Sense. Both functions are activated by the toolbar icons or hotkeys.

Code Sense

The Code Sense feature is currently available in products with Java language support. It helps you to auto-complete references to standard Java classes in Java code, significantly speeds up your coding and helps you avoid syntax errors.

Code sense works in two modes:

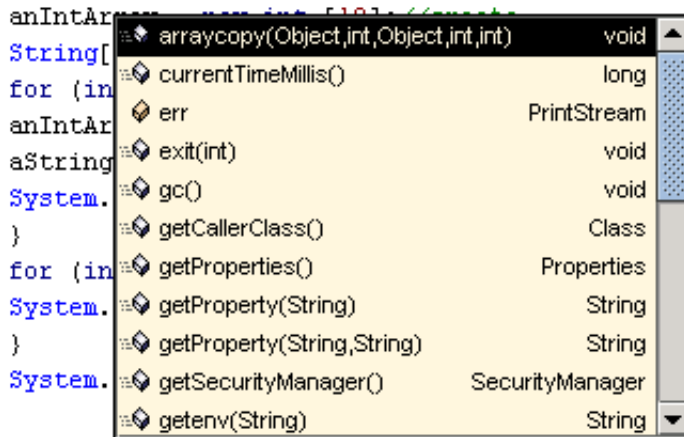
- invoked by a hotkey stroke (Ctrl-Space by default) or  button on the Editor toolbar
- activated upon delay. This is defined in the Options dialog:
 - Check **Tools | Options | <level> -Code Sense: Activate**.

- Enter duration of the delay in milliseconds in the **Tools | Options | <level> -Code Sense: Delay**

Example 1

1. Type `System.`
2. With the insertion point cursor placed after the second period character, press `Ctrl-Space`. The list of available methods for this class is displayed.

Figure 65 Code sense



3. Select the desired method with mouse or arrow keys and press **Enter**. The name of the selected method is inserted into the line.

Example 2

1. Type `System.out.p`
2. With the insertion point cursor placed after the letter `p`, press “**Ctrl-Space**.” The list of available methods for this class beginning with the letter `p` is displayed.
3. Select the desired method with mouse or arrow keys and press **Enter**. The name of the selected method is inserted into the line.

Advanced Code Sense

This function is activated by `Shift+Space` hotkey. Depending on the current caret position, the advanced code sense completes the code appropriately. Using advanced code sense, you can perform type casting, complete expressions with the new keyword, and suggest commonly used names.

Type casting

The Smart Code Completion enables to cast an expression value to the required type.

Example:

```
String s = (<caret is here>
```

After pressing the assigned hotkey or toolbar button, the code fragment will be completed to:

```
String s = (String)
```

Instantiating new objects

Applying advanced code sense after the new keyword enables users to instantiate an object of the expected type.

Example:

```
StringBuffer buffer = new <caret is here>
```

After pressing the assigned hotkey or toolbar button, the code fragment will be completed to:

```
StringBuffer buffer = new StringBuffer(<caret is here>);
```

If the instantiated object has parameters, the list of possible parameter sets displays next to the opening parenthesis.

Suggesting commonly used names

Advanced code sense suggests commonly used names for the fields and variables depending on their type.

Example:

Entered type - `FileInputStream`, suggested names - `fileInputStream`, `inputStream`, `stream`.

Import Assistant

The import assistant feature is supported for Java only. When the source code contains references to a class that belongs to a package or library in the projects, but has not been properly imported, this class name is highlighted in red, and a tooltip shows up prompting the user to perform the required import.

To run the import assistant:

1. Navigate to the highlighted name to get the tooltip.
2. Press **Alt+Enter**.

Result: The import statement adds to the source code and removes the red highlight.

Snippets

Using the snippets significantly speeds up the process of application development. You can define as many snippets as you want. Then, while coding in the Editor, type the name of the snippet and press **Ctrl-J** to insert the entire block of code contained therein. There are already several default snippets for common constructs in each supported language: `if`, `for`, `while`, etc.

Defining Snippets

1. In the Options Dialog, open *Text Editor-Schemes* node.
2. Expand the node for the language you want to configure.
3. Select *Snippet* option node and click **Edit** button to display the Snippets Editor dialog.
4. Select one of the existing snippets listed at left, or click the **New** button to add a new snippet. If adding a new snippet, give it a description in the *Title* field, and name in the *Abbreviation* field.
5. In the lower editing area, type the actual code that you want inserted in your source files when you use this snippet later.
6. Check Space Expand flag to enable expanding of a snippet by hitting space bar rather than Ctrl+J.
7. In the *Where to expand* field, select the area where the snippet is applied.
8. Click **OK** when done and close the Options dialog.

Deleting Snippets

1. Perform steps 1-3 listed above.
2. Select the snippet name at the left you no longer need.
3. Click the delete button. The selected snippet is permanently deleted.

Using Snippets

To make use of a snippet, type in the name keyword and press **tab (or Ctrl-J)**. The code completes automatically, leaving you a chance to do some job of replacing the placeholders with the required expressions.

TIP: You can press **Ctrl+J** or click the snippet button on the Editor toolbar, and select a snippet from the drop-down list.

Browse Symbol

Browse symbol feature allows users to view the source codes of project and library classes. When you click on a class name or attribute name, Browse Symbol opens the source code of this class and highlights declaration. When you click on an operation name, the source code of the corresponding class opens, highlighting the method signature.

Adding library classes to the sourcepath

By default, Browse Symbol works with the project classes only. To work with the library classes, it needs additional customization of the sourcepath.

In the Builder page of the Options dialog expand the node *Compiler Options*, select the field *Sourcepath* and specify path to the JDK source jar file. If your JDK home directory is `C:/jdk1.3`, then the sourcepath will be `$SOURCEPATH$PS$C:/JDK1.3/src.jar!/src`.

Alternatively, you can unzip `src.jar` and specify path to the source library:
`$$SOURCEPATH$$P$$C:/JDK1.3/src.`

You can press the Browse button to invoke the Sourcepath dialog. Click **Add Path** and select the required path from the file chooser.

- If you wish to add a jar file this way, edit the field and append “!/src” to the end of the sourcepath string.
- Apply the changes and close the Options dialog.
- The library classes open in read-only tabbed pages.

TIP:  and  buttons in the Editor toolbar allow users to switch between the file tabs and easily return to the original point of the browse sequence.

Context Help

The text editor enables you to create and use context help that provides access to the specification of JDK (for example, JDK 1.3 downloaded from <http://java.sun.com/j2se/1.3/docs.html>), and other documentation at the user's discretion (EJB specifications, J2EE, WebLogic or project documentation generated using the doc generation facilities of Together).

WARNING! Context Help does not display HTML files with applets. If you generate the project documentation using Generate HTML, do not switch to the frame mode.

Creating a help database

To enjoy the advantages of this feature, you have to create the database of libraries. Help libraries are not supplied with Together. It is the user's responsibility to create context help system according to the specific requirements. Each library is by default available for all projects. You can skip some libraries from the global search, and make them available to the selected projects only.

To create a help database:

1. On the main menu, choose **Help | Context Help libraries**.
2. In the Context Help Libraries dialog, choose Libraries tab and click **Add** to create a library.
3. In the Library dialog, specify the library name and click **Add** button.
4. In the Select Folder or File dialog, pick the required resource and click **OK**.

NOTE: You can repeat this step to add as many files or folders to the library as needed.

5. Check *Skip* flag, if you wish to make certain libraries unavailable on the global level.
6. In the Context Help Libraries dialog, choose Assign Libraries to Project tab.
7. In the *Assign* column, check the libraries to be assigned to the current project.
8. Click **OK** to close the dialog.

Each library adds its own tab to the Context Help.

You can modify the help database as needed. The Context Help Libraries dialog makes it possible to add new resources, edit and delete libraries, assign or unassign libraries to projects.

To modify the context help libraries:

1. On the main menu, choose **Help | Context help libraries**.
2. Change the set of libraries as required.

Example:

Let us consider creating a set of libraries for a project, including JDK and project documentation.

1. Open the project.
2. In the main menu, choose **Project | Generate HTML** to create the project documentation.
3. Choose **Help | Context Help libraries**.
4. Create the library that contains JDK documentation:
 - Click **Add** button to open the Library dialog.
 - In the Library dialog, click **Add** to add specific resource to the new library.
 - In the file chooser dialog, select the folder that contains JDK documentation, and click **OK**.
5. Create another library, that contains Project documentation:
 - Click **Add** button to open the Library dialog.
 - In the Library dialog, click **Add** to add specific resource to the new library.
 - In the file chooser dialog, select the folder that contains the generated project documentation (for example, `$TGH$/out/webpublish/<project name>`), and click **OK**.
6. Assign both libraries to the current project.

Using context help

When the database is successfully created, pressing F1 with the cursor on a certain keyword will put in action a search throughout the context help database.

You can find detailed description of controls of the Context Help dialog in the online help system.

Integrated XML Editor

The XML Editor allows you to create and edit XML files, providing their structure view on the XML Structure tab of the Explorer. The XML Editor includes syntax highlighting and other useful features of the Together text editor (refer to “Advanced Editor Features” on page 239 for details):

- Bookmarks
- Snippets
- Surround with

- Context help
- Go to line
- XML-specific feature: toggling between textual and table views of XML files.

These features are accessible via the editor toolbar and right-click menu.

NOTE: The XML Editor also supports working with XSL, JSP, and HTML files.

To enable XML Support:

1. Choose **Tools | Activate/Deactivate Features** on the main menu.
2. On the Together Features tab, check the box *XML Support*.

NOTE: If XML Support is deactivated, it is still possible to edit XML, HTML, and JSP files in the text editor.

An XML file is displayed in a special XML tab of the Editor pane, and the structure is dynamically reflected on the XML Structure tab of the Explorer. The Explorer tree-view enables adding and deleting elements, attributes, processing instructions, CDATA sections, comments, and namespace declarations. Selecting a node in the tree-view automatically navigates to the corresponding line of the source code.

The XML file structure takes into consideration the underlying DTD or XSD Schema file (if any), and enables or disables actions according to the specified cardinality of elements.

Editor settings for XML, JSP, and HTML files are configurable in the relevant nodes of the *Text Editor* options in the Options dialog. Follow the same procedures as described in the section “Configuring the Editor” on page 231.

Working with the XML Editor

XML files are created using the Object Gallery. Note that to use the XML Editor you need to activate the feature *XML Support*.

To create an XML file:

1. Choose **File | New** on the main menu.

TIP: Alternatively, click the New button on the main toolbar, or use the Ctrl+N keyboard shortcut.

2. In the Object Gallery, select the **XML** node.
3. Click on the **XML file** template to create a skeleton for an XML 1.0 file, and click **Next**.
4. In the New XML File Properties dialog, specify the file name, encoding, and the name of the root element, and click **Finish**.

Result: The XML Structure tab is added to the Explorer, displaying the structure of the new file, and the source code opens on the new XML tab in the Editor pane.

To open a file for editing:

1. Choose **File | Open** on the main menu.
2. In the Open File dialog, navigate to the required file and click **OK** to open it.

TIP: There are some helpful tricks for opening files in the XML Editor.

- Double-click the file name on the Directory tab of the Explorer.
- Use the **File | Recent Files** command to quickly locate frequently used files.
- It is possible to work with multiple files opened on different tabs. The XML Structure tab in the Explorer shows the structure of the active Editor tab.

Developing XML file structure on the XML tab

Having created a skeleton XML file, you can continue developing its structure.

The XML Structure tree-view provides a set of right-click menu commands, depending on the node type. All changes to the structure are immediately reflected in the source code, and vice versa.

To add a child element (CDATA, comment, processing instruction):

1. Right-click the element where you want to add a child node.
2. Point to **Add Child** on the right-click menu.
3. Choose the type of child node to add from the submenu.

Result: The child node is added under the root element, and the appropriate section is added to the source code.

To add a peer element (CDATA, comment, processing instruction):

1. Right-click the element where you want to add a peer node.
2. Point to **Append** on the right-click menu.
3. Choose the type of child node to append from the submenu.

Result: The new node is added on the same level as the selected element, and the appropriate section is added to the source code.

To add an attribute:

1. Right click on the element where you want to add an attribute.
2. Choose **Add Attribute** on the right-click menu.

Result: The new attribute is added under the selected element, and the appropriate section is added to the source code.

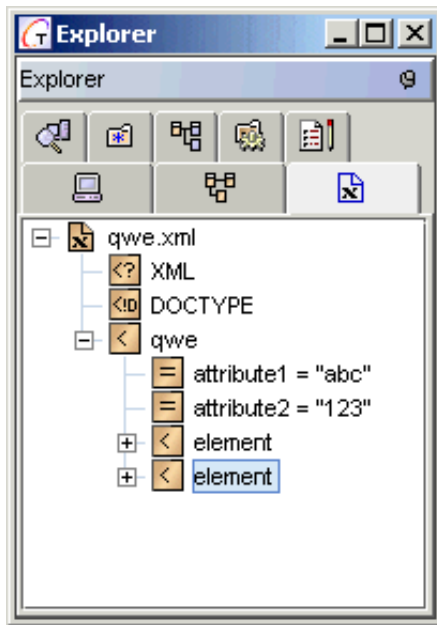
NOTE: Adding a new element or attribute immediately enables the in-line editor. To edit the names and values later, click twice on the node name or press F2.

To add a document type declaration or XML type declaration to the XML file:

1. On the XML Structure tab of the Explorer, point to **Add Child** on the root element's right-click menu.
2. Choose **XML Declaration**, or **Document Type Declaration** as required.

TIP: A doctype declaration can be appended to the XML type declaration, using the **Append | Doctype declaration** command on the right-click menu for the XML type declaration.

Figure 66 XML Structure tab in the Explorer



There are several ways to edit names and values in an XML file:

- Using the in-line editor
- In the source code
- In the Inspector

You can modify the structure of an XML file by moving elements among levels.

To move an element one level up or down:

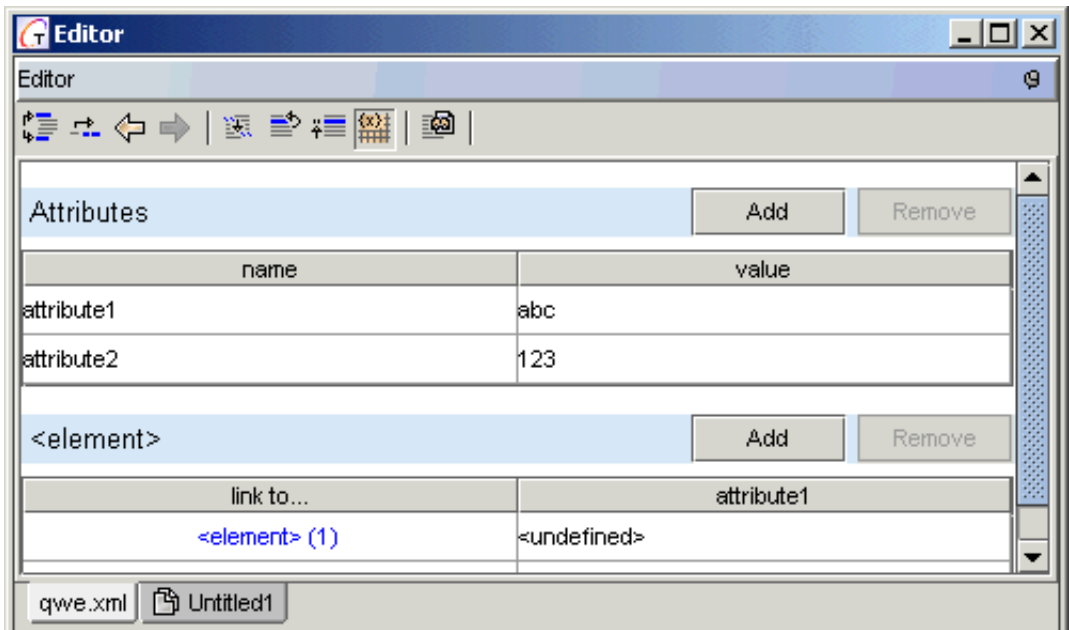
1. Right-click on the element to be moved.
2. Point to **Convert To** on the right-click menu.
3. Choose **Promote** or **Demote** from the submenu, as required.

Developing XML file structure using the table view

To switch to the table view:

1. On the XML Structure tab of the Explorer, select an element or an attribute.
2. On the Editor toolbar, click the Toggle XML Table View button  to switch between the textual and table representation of the source code.

Figure 67 Table view of an XML file



The table view displays dynamic contents depending on the element or attribute currently selected in the Explorer.

You can manage the child elements and attributes using the Add and Remove buttons of the table view. The values are editable in the *Value* columns. All changes are immediately reflected in the Explorer.

Editing XML, JSP, and HTML files with XML support disabled

When the XML Support feature is deactivated, it is still possible to edit XML, JSP, and HTML files, but the structure view in the Explorer and table view in the Editor are not available.

Code Sense in JSP Editor

Code Sense is also available for JSPs. It works basically the same way as for Java source code. However, to make use of this feature, you have to modify the Search/Classpath of your project.

To set up Code Sense for JSP:

1. Choose **Project | Project Properties** on the main menu.

Alternatively, right-click the project node on the Explorer's Model tab and choose **Project Properties** on the right-click menu.

2. In the Project Properties dialog add `$TGH$\bundled\tomcat\lib\servlet.jar` to the Search/Classpath.

This resource is not actually required for JSP debugging, but it makes Code Sense functional.

Viewing HTML files

There are two ways to view an HTML file:

- Choose **Tools | View in Browser** on the Editor's right-click menu.
- Navigate to the desired file in the Explorer and choose **View** on the node's right-click menu.

The HTML file is displayed in the default browser window.

Tag Library Helper

The Tag Library Helper command is available on the Tools submenu of the Editor right-click menu for JSP files. The Tag Library Helper enables you to re-use tags in JSP code. Provided that the tag library is already created, this command displays a list of available tags. See “Taglib diagram” on page 548 for details.

Tips and Tricks

Rectangular Blocks

The Editor allows users to select rectangular blocks of text. Ctrl-L key combination toggles rectangular block capabilities on and off.

If you need to work with a rectangular block:

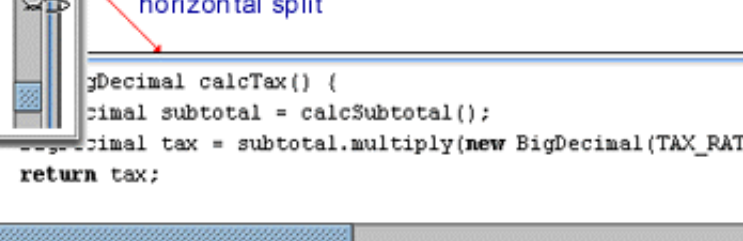
1. Press **Ctrl-L**,
2. Select the desired block of text with the mouse, or hold down the Shift key and use arrow keys.
3. Handle the selection as usual.

When rectangular block is no longer needed, press **Ctrl-L** again to switch to normal block mode.

Split Pane

For the ease of editing large files, split the Editor pane into two or four segments, each one with its own scroll bars. To split the Editor pane horizontally, grab the upper right corner of the pane and drag the splitting line down. If you grab the lower left corner of the pane and drag the splitting line to the right, you can split it vertically.

Lower left and upper right corners are marked with double arrows. Typing in any split pane is reproduced in all the other panes.



Drag down for horizontal split

```
ATE));  
  
    BigDecimal calcTax() {  
        BigDecimal subtotal = calcSubtotal();  
        BigDecimal tax = subtotal.multiply(new BigDecimal(TAX_RATE));  
        return tax;  
    }  
  
    /**  
    public class CashSale {  
  
        /** Poor way to do taxes (but it's easy <g>). */  
        public final static double TAX_RATE = 0.06;  
  
        /**
```

CashSale.java InsuffPaymentException.java



The Editor pane may be hidden by default depending on the *Role after restart*, selected in the **Tools | Options - General**.

Using the Editor with an open project

When a project is open, you can have one or more source-generating diagrams open concurrently in the Diagram pane. As you click on source-generating elements in the current diagram, the contents of the Editor updates to display the source code for the selected element. It displays the same file until you select a different source-generating element in the same diagram or another open source-generating diagram.

You may also open non-source diagrams such as Use Case or State concurrently with source-generating diagrams. When you first open a project, the following default Editor pane behavior is in effect:

- When you then open or select the tab of a non-source diagram, the Editor pane hides automatically.
- If multiple diagrams, some source-generating and some not, are open concurrently, the Editor pane shows when you select a source-generating diagram in the Diagram pane and hides when you select any other diagram.

This default behavior prevails unless you override it by using the Main Toolbar or View menu to show the Editor pane while you are focused on a non source-generating diagram. From that point on, until the end of the current session, control the display of the Editor pane using the View menu or Main Toolbar.

NOTE: If you have already have a file opened in some of the Editor's tabs, Together will open the existing tab in an attempt to open that file again.

Using the Editor with no open project

When you launch Together, the Editor pane fills the entire right side of the Main Window and displays a single tab. A new file "<Untitled>" is open. You can immediately edit and save this file, or you can use the Directory tab in the Explorer or the Editor right-click menu to open one or more other files (see Opening Files below). Files supported by the Editor pane include the source files of the supported languages, text type files, and configuration files such as *.properties or *.config.

Although you can open and edit source files without opening any project, most of the time you will probably work with diagrams and files in the context of an open Together project.

External Editor

In addition to the built-in editor, you can use one of the popular editors as an external tool. Win Vi is the default external editor and comes bundled with Together (\$TGH\$/bundled/winvi). The other editors should be installed separately.

Configuring the external editor

You can assign a different external editor. This is how it's done:

1. Open Options dialog on the required level.
2. Choose **Tools | External Editor**.
3. In the *Language* field, choose the target language from the drop-down list.
4. In the *External Editor* field, choose the editor from the drop-down list.

5. Choose *External Editor - Show in menu* node. Select the menu types, where external editor's command will be shown. It is possible to add the external editor command to the right-click menus of files and source-generating visual items on diagrams (classes, interfaces etc.)

Using the external editor

To invoke the currently assigned external editor, do one of the following:

- Choose file in the Explorer, open its right-click menu, and click **Tools | External Editor**.
- Choose a source-generating visual element in the Diagram pane, open its right-click menu, and click **Tools | External Editor**.
- In the Editor pane, open right-click menu and choose **External Editor** from the **Tools** menu node.

Search Facilities

Together allows users to search for elements and strings throughout the diagrams and projects. Commands are available under the Search item of the main menu. Search results display in separate tabs. Toolbar icons at the left of search result tabs enable expanding /collapsing the tree-view nodes and saving the search results.

Find and Replace

These are the most basic search facilities. You can find the specified string in the current file and replace it with the other string.

To find a string:

1. On the main menu, choose **Search | Find** to open Find dialog.
2. In the *Text to find* field, enter the search string.
3. Click **Search**.

TIP: To open the Find dialog, use Ctrl+F keyboard shortcut, or click  icon on the main toolbar.

To replace a string:

1. On the main menu, choose **Search | Replace** to open *Replace* dialog.
2. In the *Text to find* field, enter the search string.
3. In the *Replace with* field, enter the replace string.
4. Click **Replace**.

TIP: To open Replace dialog, use Ctrl+H keyboard shortcut.

To navigate to the next occurrence of the search string:

- Press **F3** or choose **Search | Find Next**.

To navigate to the previous occurrence of the search string:

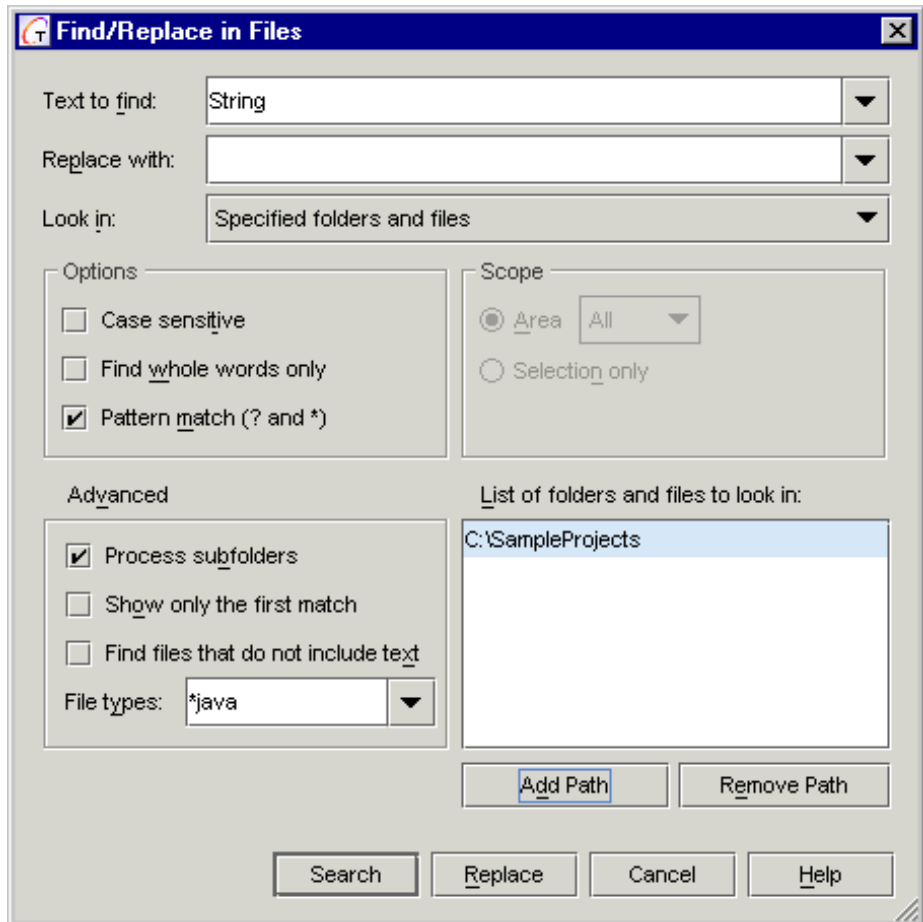
- Press **Shift+F3**, or choose **Search | Find Previous**.

Find and replace in files

This feature enables you to find and replace occurrences of the search string in the specified range.

To replace a string in the specified range:

1. On the main menu, choose **Search | Find/Replace in Files**.



2. In the *Text to find* field, enter the search string.
3. In the *Replace with* field, enter the replace string.
4. From the *Look in* combobox, choose the range where the encountered search strings will be replaced. You can search in the current file, in the source files of the whole

project, in all opened files, or in the specified locations. Further behavior of the dialog depends on the selected range.

5. In the *Options* section, check the following flags if necessary:
 - *Case sensitive*: Enables case sensitive search;
 - *Find whole words only*: Matches the whole words and ignores substrings;
 - *Pattern match*: Enables usage of the regular expressions. Pattern match may include “*” symbol to replace any number of characters, and “?” to replace a single character.
6. In the *Scope* section, choose *Area* radio-button to search in the specified direction, or *Selection only*, to search within the selected fragment. These controls are only enabled when *Current File* is selected in the *Look In* combobox.
7. Check *Process subfolders* flag if the nested directories should be searched. Note that this flag is only enabled for the search in the specified folders and files.
8. Check *Show only the first match* flag if you want to confine your search with the first occurrence of the search string.
9. Check *Find files that do not include text*, to perform inverted search. This flag is enabled for all search ranges, except for the current file.
10. Using **Add Path** button, specify the folders and files to be searched. Press **Remove Path** button to delete the selected path. These buttons are only enabled for the search in the specified folders and files.

Search on diagrams

This function allows users to search the current diagram or all opened diagrams for the specified string in a certain scope.

To search a string on diagrams:

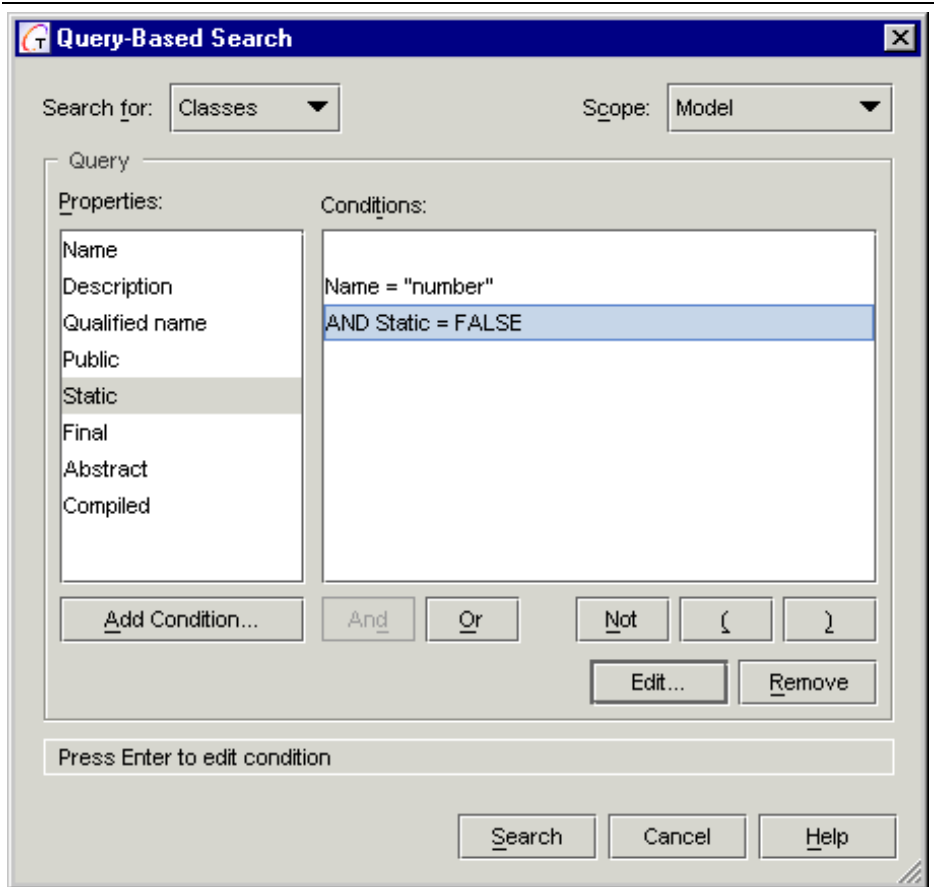
1. On the main menu, choose **Search | Search on diagrams**, or use Ctrl+Shift+D keyboard shortcut, to open the dialog.
2. In the *Text to find* field, enter the search string.
3. In the *Options* section, check the following flags if necessary:
 - *Case sensitive*: enables case sensitive search;
 - *Find whole words only*: matches the whole words and ignores substrings;
 - *Pattern match*: enables usage of the regular expressions. Pattern match may include “*” symbol to replace any number of characters, and “?” to replace a single character.
4. In the *Scope* section, choose the current diagram or all opened diagrams.
5. Click **Search**.

Query-based search

This advanced features enables you to find project elements (packages, classes, variables etc.) on the base of filters, or queries. The filters are created using logical expressions. *Query-based Search* dialog provides basic facilities to build the queries.

NOTE: Since Together supports modeling without source code, it is possible to create diagrams where the elements have no source code representation. Such elements cannot be found using the query-based search. This relates to the elements of the non-source generating diagrams (for example, use cases or actors), and to the class diagrams created in projects with the Design language selected as default.

Figure 69 Query-Based Search dialog



Query, or filter is a logical expression that consists of a set of conditions. The *Query* section of the dialog includes the *Properties* list and *Conditions* field.

For each selected element, the list of properties contains specific parameters. You can choose any of these parameters to build up a set of conditions, which are presented in one of the following forms:

<property> <operation> <value> ,

or

hasProperty<property>

where <property> is the name of element's property, and <operation> is "=", "!=", "<", or ">".

If the conditions are applied to strings, it is possible to use a pattern match. Pattern match may include "*" symbol to replace any number of characters, and "?" to replace a single character.

Conditions may use brackets and logical binary operations (AND, OR, NOT). The dialog provides relevant buttons for each logical operation.

When creating queries, follow the usual rules of logical operations: logical negation (NOT) has the highest priority, logical multiplication (AND) is the next, and logical addition (OR) has the lowest priority. For example, in the expression

Name = "Class1" OR "Class2" AND "Class3"

the AND operation has higher priority and will be performed first. Such expression returns the value Class1.

If you need to change the priorities of operations, create expressions with brackets.

To filter out an element:

1. On the main menu choose **Search | Query-Based Search**, to open the Query-Based Search dialog.

TIP: You can also use Ctrl+Shift+Q keyboard shortcut.

2. In the *Search for* combobox, choose which element to find. The possible options are: packages, classifiers, classes, interfaces, operations, attributes.
3. In the *Scope* combobox, choose where the element is sought for. The possible options are: model, classpath, whole project, current package or result set.
4. Create the query that will help you filter out the required element:
 - In the *Properties* list, choose the specific property to be added to the filter.
 - Click **Add Condition** to open the *Condition* dialog. The form of this dialog depends on the type of the selected property.
 - Use **And**, **Or**, **Not** and bracket buttons to make up logical expression.
5. Click **Search**.

You can find a detailed description of the dialog controls in the online help system.

Go to line

This is the most basic command that enables users to navigate directly to the specified line in the Editor.

To navigate to a line:

1. On the main menu, choose **Search | Go to Line**, to open Go to Line dialog.

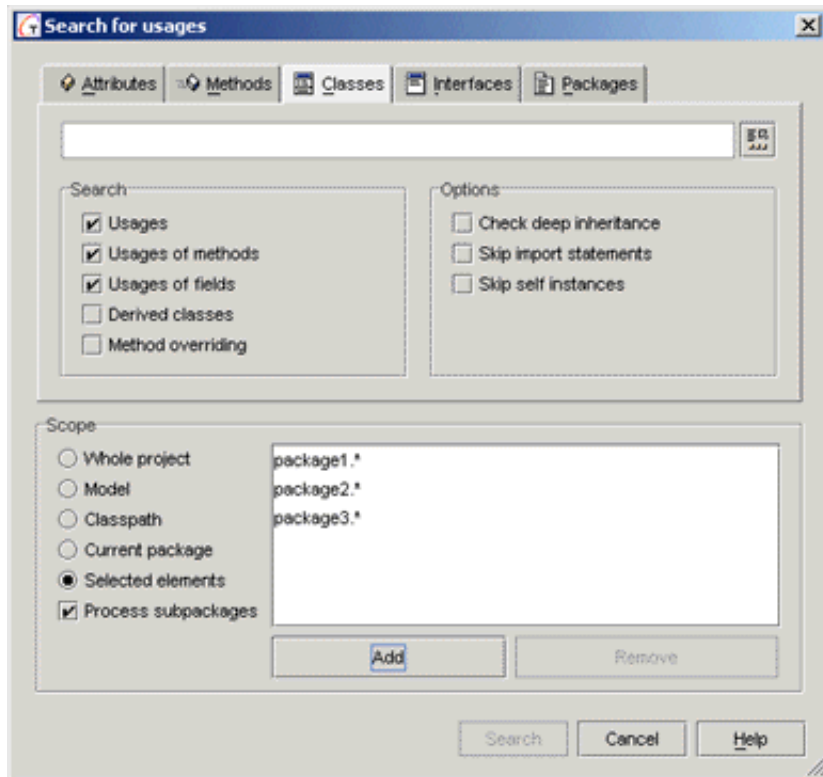
TIP: Alternatively, click  button on the Editor toolbox, or use Ctrl+G keyboard shortcut.

2. Enter the target line number.

3. Click **OK**.

Search for Usages

Search for Usages command enables you to track how an element is used in a project. The Search for Usages dialog contains tabs that allow users to organize searches for packages, classes, operations and attributes. Each tab has its own set of controls, described in detail in the appropriate page of the online help system.



To find the usages of an element:

1. On the main menu, choose **Search | Search for Usages**.

TIP: You can also choose Search for Usages on the right-click menu of the Diagram or Editor pane, or use Ctrl+Shift+U keyboard shortcut.

2. In the *Search for Usages* dialog, choose the tab for the required element.

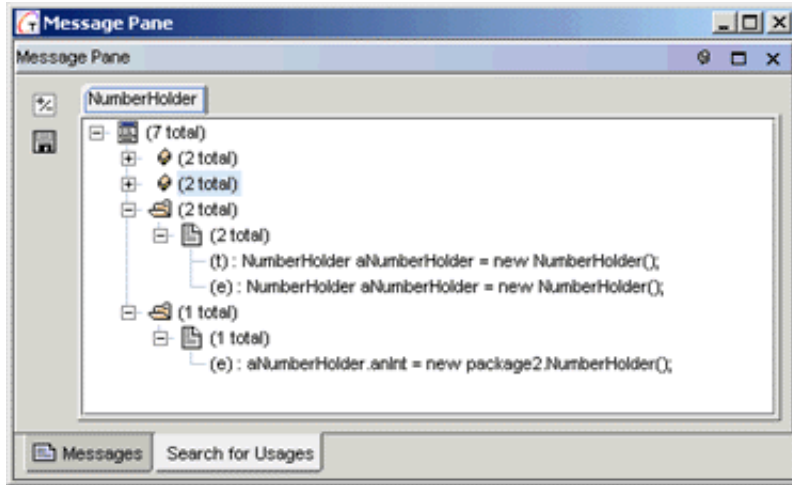
3. In the *Search* section, check the type of usage to be sought.

4. In the *Options* section, check specific usages to be skipped, if necessary.

5. In the *Scope* section, choose one the radio buttons to specify the range of search.

6. Click **Search**.

The results display in *Search for Usages* tab and present a tree-view, each node containing all usages of an element in a certain class member. If the sought element is used as a type, the appropriate line in the result set is marked with (t). If it is used as an element, the line is marked with (e).



Compile - Make - Run

Using Compile and Make from Together

When developing applications using Together you can compile classes and make your project without leaving the Together environment. There are several options:

- **Using the default compile/make tool:** Executes the default Java compiler and make utility installed along with Together.
- **Using another (external) Java compile/make tool:** Executes a user-defined external Java compiler and/or make utility. You need to modify the compiler specification in your configuration options (see below) to point to a different compiler.
- **Using a C++ compiler/make tool:** Compiles a C++ project using your preferred external tools. You need to modify the compiler specification in your configuration options (see below) to point to a different compiler.

Compile and make tools are configurable via the Options dialog (Tools | Options on the required level).

Executing the compile and make tools

Execute the default Java compile/make tool from any of the following places:

- **Selection menu**
 1. Click on the diagram background or on a class icon to be compiled.
 2. Choose **Selection | Tools | Make Node** (or **Rebuild Node**) on the main menu.
Note that the **Selection** menu is enabled only when the Designer pane gets the focus.
 - **Explorer pane:** Project, package, or class right-click menu.
 1. Open the Model tab in the Explorer.
 2. Navigate to the class or package to be compiled.
 3. Choose **Tools | Make Node** (or **Rebuild Node**) on the right-click menu.
 - **Designer pane:** Project, package, or class right-click menu in the Class diagram.
 1. Right-click the diagram or class to be compiled.
 2. Choose **Tools | Make Node** (or **Rebuild Node**) on the right-click menu.
 - **Editor pane:** Current file in the Editor.
 1. Open the required file in the Editor.
 2. Choose **Tools | Make Node** (or **Rebuild Node**) on the right-click menu of the current file.
 - **Builder pane:** Click Make  and Rebuild  buttons on the Builder pane.
- TIP:** Use keyboard shortcuts Shift+F8 for Compile and Ctrl+Shift+F8 for Rebuild.

Compile and make tools are pre-configured to output messages to the Builder tab of the Message pane. In you encounter errors or warnings during compilation, click on a message and navigate directly to the line of source code that caused the error or warning. The Message pane also displays executed commands and status of tool execution.

Configuring and using the standard compiler

On all platforms, Together installs the built-in compiler from Java 2(tm) SDK version 1.3 as the default compiler. This compiler resides in \$JDK_HOME\$/lib/tools.jar.

In general, the default compiler is pre-configured and ready to work. However, you can reconfigure it using the Options dialog.

To tune the default compiler:

1. Choose **Tools | Options | <level>** on the main menu.
2. In the Options dialog, navigate to *Builder - Built-in Javac*. Specify the required settings.
3. Click **OK** to apply changes and close the dialog.

TIP: If you need to ignore certain files during a build, you can specify them in the Options dialog (*General - Ignore files and folders*).

Compiler output

Compiler errors and warnings are displayed in a navigable overview. You can choose between full and reduced output format. The maximum allowed number of compiling errors is also user-defined.

To configure the compiler output:

1. Choose **Tools | Options | <level>** on the main menu.
2. In the Options dialog, navigate to *Builder - Built-in Javac*.
3. If necessary, check the box *Reflect compiling process in the status bar*, keeping in mind that it may increase compilation time.
4. For the *Show javac output in full format* option choose full or reduced output format.

NOTE: To change the format of compiler output on the fly, choose **Full Output | Reduced Output** on the right-click menu of the Builder tab.

5. For the *Maximum number of errors* option, specify the permissible number of compiling errors.
6. Apply changes and close the Options dialog.

Cross-compilation

By default, classes are compiled against the bootstrap and extension classes of the JDK that the javac tool shipped with. But javac also supports cross-compiling, where classes are compiled against a bootstrap and extension classes of a different Java platform implementation.

In order to perform cross-compilation, you have to select the target platform. Together provides three possible virtual machine (VM) options:

- 1.1 Generated class files are compatible with 1.1 and VMs in the Java 2 SDK. This is the default.
- 1.2 Generated class files run on VMs in the Java 2 SDK v 1.2 and later, but do not run on 1.1 VMs.
- 1.3 Generated class files run on VMs in the Java 2 SDK v 1.3 and later, but do not run on 1.1 or 1.2 VMs.

In addition, it is necessary to define the *-bootclasspath* and *-extdirs* javac options. All these options are configured in the Options dialog.

To enable cross-compilation:

1. Choose **Tools | Options | <level>** on the main menu.
2. In the Options dialog, navigate to *Builder - Built-in Javac - Compiler options - Cross-compilation*.
3. Check the *Cross-compilation* box.
4. Specify the platform, boot classpath, and extension directories.
5. Click **OK** to apply the changes and close the dialog.

Configuring and using an alternative compiler

You are free to use any compiler that meets your needs for all the languages supported by Together. As an alternative to the default compiler, configure your favorite compile / make tool in the *Tools* node of the Options dialog and assign it in the *Builder - User Compiling Tool* options.

To set up an alternative compiler tool:

1. Choose **Tools | Options | <level>** on the main menu.
2. In the Options dialog, select *Tools | Tool#x*. Configure the tool:
 - Specify the command name to be displayed in the menu, command line parameters, compile/make output, menu settings, and other options.
 - Check the required boxes in the *Show in menu* node to specify the menus where this command can be invoked.
 - Use information from the *Description* section of the Options dialog to help you create the correct configuration.

TIP: To avoid overwriting the default compiler/make settings, use empty *Tool#x* slots.

3. Navigate to *Builder - User Compiling Tool*.
4. Select the required tool from the list in the *Tool* option.
5. To activate the alternative compiler instead of the default compiler, check the box *Builder - User compiling tool - Use by default*.

Having specified your own compile and make commands, you can run them from any menu that you have checked.

Running programs in Together

In order to run your project, you have to choose one of the Run commands on the main menu. Running a program requires specifying main class name, arguments and VM options.

To run a program:

1. On the main menu, choose **Run | Run** (or click F9)
2. In the Run Arguments and Parameters dialog, choose run configuration or just specify main class, VM options, and program arguments.
3. Click **OK** to run the program.

Run/Debug Configurations

Large-scale projects may have numerous main classes and significant number of parameters. Together helps simplify the process by means of Run/Debug Configuration feature, which allows users to store various sets of runner parameters for classes, applets and servlets.

To create a run configuration:

1. On the main menu, choose **Tools | Run/Debug | Run/Debug Configurations**.

TIP: Use Shift-F10 keyboard shortcut.

2. Click **Add** to open the Arguments and Parameters dialog.
3. Click on Class, Applet, or Servlet tab as required.
4. Enter the run configuration name.
5. Specify arguments and parameters:
 - for applications: class with the main method, program arguments and VM options;
 - for applets: applet class and parameters, VM options and frame size;
 - for servlets and JSP: start page or servlet name, query string, context parameters, and VM options.
6. Click **OK** to close the Arguments and Parameters dialog. The new configuration adds to the list.

You can add as many configuration as needed, and assign one of them as the default one. Default configuration is highlighted in bold.

Makefile Generation

Together provides the possibility to generate makefile for your projects. This feature is especially precious for large-scale projects. Having once created a Together project and generated a makefile for it, you can further install this project anywhere, independently from Together.

Makefile contains all necessary command line commands and parameters, required for the independent run of your project. You can generate makefile on various levels: for a project, a package, or even a class. Default Makefile destination is specified in the Builder node of the Options dialog, and you can specify any other location on the required level.

To configure the makefile destination:

1. On the main menu, choose **Tools | Options - <level> - Builder**.
2. In the *Makefile destination directory* field, click the file chooser button.
3. In the Select Path dialog, navigate to the target folder intended to store the makefile, and click **OK**.
4. Apply the changes and close the Options dialog.

To generate makefile:

1. On the main menu, choose **Project | Generate Makefile**.

NOTE: To generate makefile for the selected object, choose **Selection | Tools | Generate makefile** on the main menu, or object right-click menu in the Diagram pane or in the Model tab of the Explorer.

2. If a makefile has already been generated, confirm whether you are going to overwrite it.
3. You can choose to open the generated makefile immediately in the Editor. Click **Yes** in the dialog box to open the makefile for edit.

Debugging

Using the Integrated Debugger

Full-featured integrated Java debugger allows you to debug your projects right inside *Together*:

- **Breakpoints:** Enables you to stop at any line of the source code including "Logging" and "Pass Count" features. Provides 5 kinds of breakpoints: line, exception, class, method and attribute breakpoints.
- **Command execution:** "Run", "Pause", "Continue", "Stop" commands available. For debugging methods there are "Step over", "Step into", "Step out" and "Run to the end of method" facilities.
- **Watch:** You can watch/modify expressions, variables, and class members.
- **Evaluate:** Enables you evaluating variables and expressions, and changing their values in course of run in Debugger.
- **Threads:** Enables you to browse the state, methods and variables of a thread.
- **Frames:** You can work with the information of the current frame.
- **Skip classes:** Enables you to specify classes that should be skipped. Shows the classes that are already loaded.
- **Remote process:** Enables you to attach to a remote process by specifying address and transport.

There are several ways to access the debugger commands:

- **Run | Run/Debug** commands on the main menu;
- Debugger toolbar, which can be accessed using **Run | Run in Debugger**,
- Keyboard shortcuts.

Starting a Debugger session

TIP: Before starting a debugging session, be sure you've set up all the needed breakpoints. If the breakpoints are not in place, the Debugger session does not start. For more information see "Working with breakpoints" on page 238.

To start the debugging session:

1. Select Run/Debug configuration, if necessary;
2. Choose **Run | Debug** on the main menu or use Ctrl+F9 hotkey.

This starts your projects in debug mode and automatically shows the Debugger Tab. Debugger Tab significantly speeds up access to the information you could need during the debugging process.

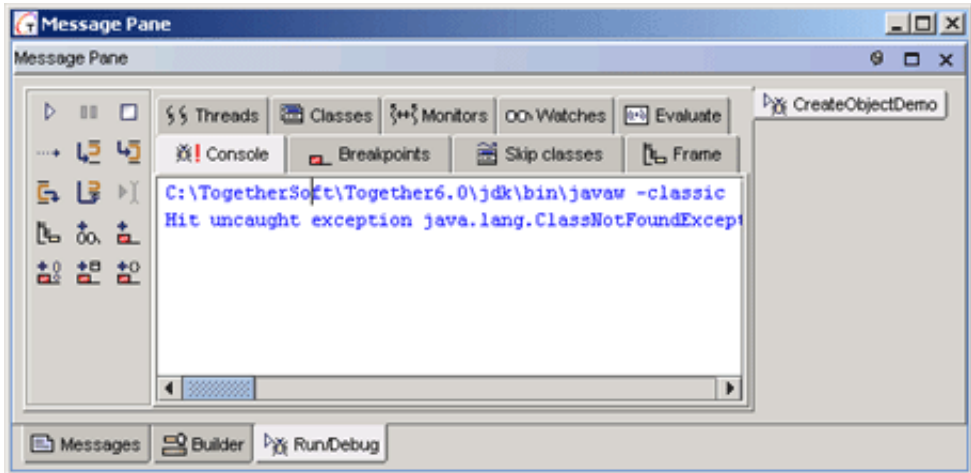
Run/Debug Tab

Run/Debug tab includes the following elements:

- **Console:** Displays Java console

- **Threads:** Threads viewer. Displays in details all the running threads.
- **Classes:** Displays a hierarchy of loaded classes.
- **Monitors:** Displays synchronization monitors.
- **Watches:** Watch window for class members/expressions.
- **Breakpoints:** Displays list of breakpoints.
- **Skip classes:** Contains list of "disabled for debug" classes.
- **Evaluate:** This tab enables evaluating expressions and modifying values (including method executions) when the Debugger is in suspended state.
- **Frames:** Frames viewer. Displays the frame stack of the current Thread.
- **Debugger toolbar:** Provides mouse-click access to the debugger commands such as *Pause*, *Resume*, *Reset*, *Step over*, *Step into*, etc.

Figure 70 Run/Debug tab



Controlling program execution

When a debug session starts, you can control the program execution using the following set of control commands:

Table 35 Debugger commands

Command	Icon	Description	Hot-key
Continue Program		Resumes program execution.	
Pause Program		Pauses program and passes control to the debugger.	

Table 35 Debugger commands

Command	Icon	Description	Hot-key
Continue Program		Resumes program execution.	
Stop Program		Terminates the program and debugging session.	Shift-F2
Toggle Smart Step		Debugger performs a “smart step.” Set up smart step behavior in Tools Options - Run/Debug - Debugger - Smart Step behavior .	
Step Over		Skips debugging of a method that is currently under cursor. The method executes and returns results.	Shift-F8
Step Into		Forces debugging of a method that is currently under cursor. Debugger stops at the first line of this method.	Shift-F7
Step Out		Forces current method execution and stops in the current method's caller at the next line after call.	
Run to End of Method		Forces current method execution and stops before return.	
Run to cursor		Resumes program and breaks before cursor.	F4

Breakpoints

Breakpoints provide the most powerful debugging facility. It is possible to break program execution at the specified place, in order to inspect variables, class members, etc.

Together's Debugger provides the following types of breakpoints:

- line
- exception
- class
- method
- attribute

When staying at a breakpoint, you can examine data with the aid of watches. Refer to “Using Watches” for details.

The toolbar of the Breakpoint tab contains the following buttons:

Table 36 Breakpoint buttons

Icon	Breakpoint control function
	Disable all breakpoints
	Enable all breakpoints
	Remove all breakpoints
	Edit breakpoint properties
	Go to breakpoint
	Disable/Enable breakpoint
	Remove breakpoint
	Add line breakpoint
	Add exception breakpoint
	Add class breakpoint
	Add method breakpoint

Setting breakpoints

There are several ways to set breakpoints. Initially, the breakpoints are set in the Editor, to enable the debugging session. While debugging, you can set breakpoints of specific types using buttons and commands of the Run/Debug tab.

To set a breakpoint at the selected line:

- In the source file
 - Press **F5** or click on the left margin next to the selected line;
 - On the Editor's right-click menu choose **Toggle Breakpoint**;
- On the main menu choose **Run | Add Breakpoint**.

To set a breakpoint in the Run/Debug tab:

- Click one of the buttons on the Debugger toolbar, or on the Breakpoints tab.

- On the Run/Debug right-click menu, choose one of the Add breakpoints commands.

The attribute breakpoint has no icon on the Debugger toolbar and can be added when an attribute is activated in course of the program execution in the Watches, Threads, Frames tabs. When an attribute is encountered, Add Attribute Breakpoint command appears in the right-click menu of these tabs.

To set an attribute breakpoint:

1. Click Watches tab.
2. Select the watch where an attribute breakpoint should be added.
3. On the right-click menu of the watch, choose **Add Attribute Breakpoint** command.

NOTE: Attribute breakpoints can be also added in the other tabs of the Debugger.

Modifying breakpoint properties

After the breakpoints are set, you can disable or enable them. To do this, check or uncheck *Enabled* flag for the appropriate breakpoint in the Breakpoints tab. If a breakpoint is enabled, it is possible to modify its properties.

To modify a line, class, method or exception breakpoint:

1. Choose Breakpoints tab in the Debugger.
2. Right click on the breakpoint to be modified.
3. Choose **Breakpoint Properties** command on the right-click menu of the selected breakpoint.
4. In the opened Breakpoint Properties dialog specify stop execution if necessary, number of passes through the breakpoint before stop, conditions, and also whether this stop should be logged or not.

TIP: You can also open Breakpoint Properties dialog using the Editor's right-click menu, and appropriate icon on the Breakpoints toolbar.

Apart from the Breakpoint Properties dialog, it is possible to modify all types of breakpoints in the Debugger tab. The controls of the dialog are replicated in the table of the Breakpoints tab. Note that *Pass count* and *Condition* fields are editable in place. The flags *Stop Execution* and *Log Message* may not be disabled simultaneously. At least one of these flags must be set. Refer to the description of the Breakpoint properties dialog in the Online Help for detailed description of the fields.

Attribute breakpoint has two properties that are modified in a slightly different way. These properties are: *Stop on Read* that allows users to break application when an attribute is about to be read, and *Stop on Write* that allows users to break application when an attribute is about to be written. By default both options are enabled.

To modify an attribute breakpoint:

1. Choose Breakpoints tab in the Debugger.
2. Choose the attribute breakpoint to be modified.
3. On the right-click menu of the attribute breakpoint, choose **Disable (Enable) Stop on Read/ Disable (Enable) Stop on Write** command.

Evaluating and Modifying Variables

Together allows to access the variables in course of program execution.

To enable variables evaluation:

1. Set breakpoints in the desired locations of your code.
2. Choose **Run | Run in Debugger** on the main menu.

Evaluate/Modify command becomes enabled on the Run menu, and on the Editor's right-click menu.

Displaying structured context

While your program runs in the Debugger, you can check the value of each object. To do this, you need only navigate the cursor to the desired object, and after a small delay the description of this object shows up.

The form of presentation is defined by the *Show variable as tooltip* flag in the Run/Debug | Debugger node of the Options dialog. If the flag is set (by default), evaluation window shows the entire structure and value of an object. Expand the nodes to reveal the constituent properties.

If this flag is cleared, the evaluation window shows the object's address or value only.

Evaluating arrays

When evaluating an array, you can specify the bounds of display range. To do this, right click on the array name in the evaluation window and choose *Change Display Range* command on the right-click menu, enter starting and ending element numbers, or an asterisk to display the entire array.

Evaluating and modifying objects

It is also possible to evaluate and modify objects in the Evaluate tab of the Debugger. To do this, select Evaluate command on the Editor's right-click menu and specify the variable or expression to be evaluated/modified.

Appropriate entry adds to the Evaluate tab, showing location, type and value of the object in question.

Expressions in the list are evaluated every time you choose Evaluate/Modify command on the right-click menu of the current row, or press the right-arrow icon in the second column.

You can change the value of variables, typing directly in the *Expression* field of the tab. Click arrow icon for the changes to take effect.

Note, that you can only evaluate and modify objects within the current debugging context. If you try to check an object past the breakpoint, the Result field will report that the variable is out of range.

Watches, Threads and Frames

Using Watches

Watching class members, inspecting objects etc. is an important side of the debugging process. Select **Tools | Run in Debugger** on the Editor right-click menu or on the main menu. This will open Run/Debug tab with the full set of elements, including Watches tab. The Add Watch command on the Editor right-click menu becomes enabled, when the debugging session is suspended (paused). You can create new watches both in the Editor and in the Debugger.

To add a new watch in the Editor:

1. In the source code, select the expression you want to watch.
2. Open right-click menu.
3. Choose **Add Watch** command. Add Watch dialog opens with the selected expression displayed in the *Expression* field.
4. Enter watch description if necessary.
5. Click **OK** to complete.

To add a new watch in the Debugger:

1. Choose Watches tab.
2. Right click on the table to open the menu.
3. On the right-click menu choose **Add Watch** command. Add Watch dialog opens.
4. In the *Expression* field, specify the expression to be watched.
5. In the *Description* field, enter description of the watch if necessary.
6. Click **OK** to complete.

You can add as many watches as needed.

A watch has its right-click menu that provides commands for adding, removing and modifying watches. Note that the content on the right-click menu is different for variables and arrays.

Using the right-click menu, you can:

- Add a watch for a local variable (Create local variable watch).
- Change value of the watch. This command is not available for the arrays.
- Show selected elements of arrays (Change display range). This command is available for arrays only.
- Add a watch for an element of the array (Create array component watch). This command is available for array elements only.
- Toggle between decimal and hexadecimal display of the watched expressions (Show Hex Value / Show Decimal Value).
- Modify watches (Change Watch Expression, Change Watch Description).
- Add and remove watches.
- Add attribute breakpoints

Change Display Range

If the expression being watched refers to an array, you can confine its range to the elements that are of interest for you. Change Display Range command of the right-click menu serves this purpose. You can explicitly specify the numbers of the starting and ending array elements to be displayed in the watch.

To display the required portion of an array:

1. Add watch for an array.
2. On the right-click menu of the watch, choose **Change Display Range** command.
3. In the Change Range dialog, specify the array bounds. Entering an asterisk (*) displays the entire array.

Change Values

Having selected a watch in the *Watches* tab, you can change the value of the variable. The values are changed in-place, or by means of *Change Value* command on the watch right-click menu. Note that the string values must be entered in quotes. Otherwise any modifications will be ignored.

Change display format

For the integer variables, it is possible to toggle between decimal and hexadecimal representation. Use complementary commands *Show decimal value* / *Show hexadecimal value* on the watch right-click menu.

Similar modification features are available in *Evaluate*, *Threads* and *Frames* tabs. There you can also edit variables in-place and toggle between decimal and hex views for the integer values.

Using Threads and Frames

It is possible to observe the various threads in course of program execution. Using the two buttons on top of the *Threads* tab, you can display all current threads and frames, display the current thread only or display all threads.

Frames tab represents a frame stack of the current thread being resumed. This tab displays information which is a subset of the *Threads* tab. This spares the user from unnecessary navigating to the *Threads* tab, and allows to operate with the current frame.

Combo box on top of the *Frames* tab allows users to choose the location of the currently suspended thread.

Monitors

The Monitors feature, implementing the synchronization monitors. The *Monitors* tab of the Debugger displays the monitors owned by the threads, and helps detecting the deadlocked threads, or the threads that are awaiting for certain monitors.

A monitor is owned by a thread, if it has been entered via the synchronized statement or entry into a synchronized method and has not been released through `Object.wait()`.

Sometimes the monitors cannot be displayed. It depends upon the VM in use. For example, the default Client VM of the HotSpot technology does not support this feature.

To see the monitors, add `-classic` option to the VM options.

Monitors in deadlock and non-deadlock state

The monitor in a non-deadlock state, or the thread that owns this monitor, are denoted with a yellow key. The thread object of synchronization and the owner threads are expandable, as in the *Threads* tab of the Debugger.

The monitor in a deadlock state, the thread that owns this monitor, and the waiting threads are denoted with a grey key.

Attaching to a Remote Process

You can remotely debug Java programs with the integrated debugger. Start the external Java program to be remotely debugged (debugged per attach) in the following way:

```
java ... -Xdebug -Xnoagent -Djava.compiler=NONE --> -  
Xrunjdwp:transport=dt_socket,address=8787,server=y,launch="%1\bin\win32\  
display.bat %1"
```

If port 8787 is not convenient for you, you could allow the JVM to define the port:

```
-Xdebug -Xnoagent -Djava.compiler=NONE --> -  
Xrunjdwp:transport=dt_socket,server=y,launch="%1\bin\win32\  
display.bat%1"
```

In the latter case, the JVM prints the address. Note the address and enter the value in the Attach to Remote Process dialog.

To start remote debugging:

1. On the main menu, choose **Run | Attach to Remote Process** (or click Shift+F5).
2. In the Attach to Remote Process dialog, type in the host name of the computer where the external application runs, the socket name and port number, and click **OK**.

CHAPTER 15

USING THE UI BUILDER

The UI Builder enables you to visually design and build a graphical user interface (GUI) for a Java™ application. This chapter includes the following topics:

- “Overview of the UI Builder” on page 272
- “Activating the UI Builder” on page 273
- “Setting UI Builder options” on page 274
- “Creating visually editable UI classes” on page 275
- “Using the UI Designer” on page 277
- “Designing Menus” on page 288
- “UI Builder Audits and Metrics” on page 299
- “Generating user interface documentation” on page 300
- “Customizing the Toolbox component palette” on page 300

Overview of the UI Builder

The UI Builder is “hooked” to source code, meaning that the relevant Java source code automatically generates and updates in real time as you add visual, nonvisual, and menu components to your user interface. You can optionally develop user interfaces by writing code and view the result using the UI Builder.

The UI Builder is one of Together’s “activatable” features, meaning that you can turn it off when not needed to conserve system resources. When activated, the UI Builder commands appear on the main View menu, an icon is added to the Designer pane toolbar, and two additional views are available in the Designer pane:

- **UI Design:** This is a WYSIWYG¹ design view that enables you to see frames and other containers, and the components these contain.
- **Menu Design:** This is a specialized view enabling WYSIWYG design and implementation of menu bars and right-click menus.
 1. (“What-You-See-Is-What-You-Get”)

When you work in one of the above Designer pane views, the **UI Builder** tab is added to the Explorer. This tab displays the components of your user interface, enables you to navigate the components of your user interface, and switch between UI Designer and Menu Designer views in the Designer pane. This tab is referred to as the *Component Explorer* throughout this chapter.

There is a sample Together project that you can use to explore the various kinds of things you can construct with the UI Builder. See: `$TGH$/samples/java/UIBuilder/UIBuilderSamples.tpr`.

Activating the UI Builder

The UI Builder is one of Together's *activatable features*. It is activated by default. This section explains the steps for activation/deactivation, and shows you what to look for in the user interface to tell when the UI Builder is activated.

To activate the UI Builder feature:

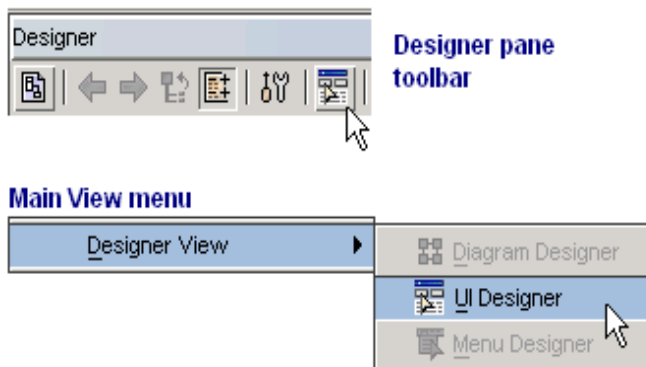
1. On the main menu, choose Tools > Activate/Deactivate Features.
2. On the Together Features page of the dialog, check *UI Builder*.

NOTE: In some pre-release builds, the UI Builder may appear on the Early Access page of the Activate/Deactivate Features dialog.

3. Click **OK** to activate the UI Builder module.

The UI Builder icon is now enabled on the Designer pane toolbar and the UI Builder command is added to the main View menu:

Figure 71 When activated, icons show on the Designer pane toolbar and main menu



Deactivating the UI Builder

When you no longer need to work with the UI Builder, it is advisable to deactivate it. Deactivating decreases the use of system resources and helps optimize performance.

Deactivate the UI Builder following the same procedure for activating it, except clear rather than check the UI Builder checkbox in the Activate/Deactivate Features dialog.

Setting UI Builder options

There are a number of configuration options for the UI Builder feature. The default settings enable you to get started creating new graphical user interfaces. When you have existing user interface code, it may be necessary to change some of the settings.

To access the UI Builder options:

1. Choose Tools | Options | Default on the main menu to open the Options dialog.
2. Select the UI Builder node in the options explorer

Choosing a profile

Choose one of several pre-defined profiles in the *IDE profile* option, including one user-defined profile. (All the profiles are individually customizable.) The profile settings define what the UI Builder looks for in a UI class, and enable the UI Builder to handle source code created with other IDE products.

If you select a profile for a competing IDE product, the UI Builder recognizes UI classes in source code that were created with the competing product; such classes will be treated as *visually editable* and will load into UI Designer view in the Designer pane. Note that any new UI classes you create in Together are not created according to the IDE profile setting.

Customizing a profile

The default values of the various profiles should suffice in most cases. You might want to customize an IDE profile if:

- A new version of a competing product changes the way it handles UI initialization, etc. in the source code it creates or generates, and you want to use Together to work on code created with that version.
- Your organization's coding standards call for something different from the default values of the *Together* or *User-defined* profiles.

Profile options

The following options are available for each profile:

- **InitGUI operation:** defines the operation name(s) that UI Builder will require to be present in any class in order to treat it as a visually editable class that can be loaded into UI Designer view in the Designer pane. Specify multiple names separated by semi-colons. When multiple names are specified, a class is treated as visually editable if it contains any of the specified operations.

- **Visibility:** defines the visibility scope of new attributes added to UI classes as you work on them.
- **Event handler style:** defines the way UI Builder will generate code for event handlers in new UI components you create. Choose between standard or anonymous inner class.
- **Object initialization point:** defines the way UI Builder will generate initialization code for new objects. Choose to have the object initialized at its declaration point or at the beginning of the class's initialization method.

Of the above options, the first one is most likely to be significant for you initially. For example, if you are accustomed to JBuilder™, you might write a new UI class with a *jbinit()* operation. But if you have the *Together* profile selected, and you try to load the class into UI Designer view, the UI Builder rejects the attempt because the profile tells it to expect an *InitGUI()* operation.

Creating visually editable UI classes

In order to develop a user interface using the UI Builder complete the following:

- A project must be open. For instructions, see “Opening a project” on page 43 and “Creating new projects” on page 95.
- A visually editable UI class must be the currently focused source code file in the Editor (normally you would do this by selecting the relevant class in a Class diagram in the Designer pane). More information for creating visually editable UI classes is provided in the next section.

What is a visually editable class?

As discussed in the section on UI Builder options, a visually editable class must contain the initialization operation defined in the currently selected IDE profile (Tools | Options | Default (or Project): UI Builder > IDE Profile). The class scope must also be declared *public*.

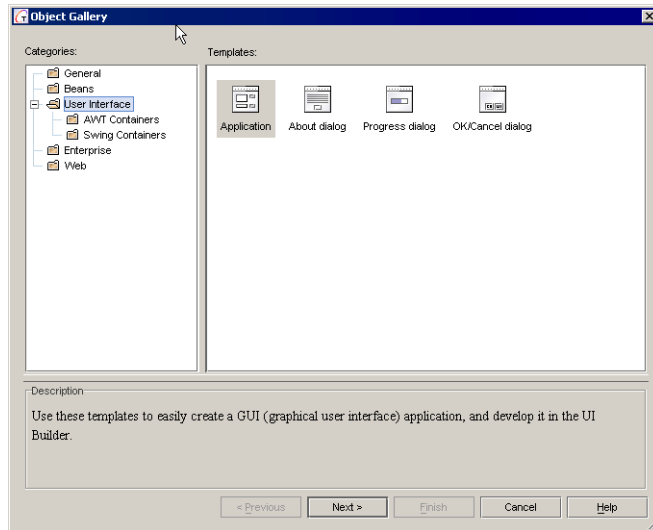
Creating UI classes

The User Interface category of the Object Gallery (File | New: User Interface) provides a quick way to create a number of visually editable UI classes. There you will find both AWT and Swing containers as well as a number of templates for more robust things such as applications and dialogs.

TIP: If you are creating a user interface without a template, the Object Gallery is the easiest way to get started.

NOTE: The User Interface templates are just patterns. They are not dynamic, and therefore do not look to the current IDE profile for the name of the initialization operation. The default initialization operation is *InitGUI()*.

Figure 72 Object Gallery categories for UI components



Using the UI Designer

Working on a user interface visually requires the Designer pane, which may or may not be visible depending on which Workspace you currently use. When you are ready to use the UI Builder, switch to a Workspace that shows the Designer pane or create a Workspace in which it is visible. (You can find the Workspaces feature on the main View menu and main Toolbar. For more information, see “Setting-up workspaces” on page 71. When you create a visually editable class using the Object Gallery, it appears in the default Class diagram of the package containing the source code. Open that diagram in the Designer pane (Diagram Designer view) and visually add attributes and operations to the class in the same way you would for any other class. You can also write code using the Editor.

The UI Designer view

When it comes to adding visual and nonvisual UI components within a container, you may decide you prefer to work on the class visually in the Designer pane in UI Designer view.

To work on a visually editable class in UI Designer view:

1. Open the Class diagram containing the visually editable class if that diagram is not already open or open the visually editable class in the Editor.
2. If working from the Class diagram, select the editable class to load its source into the Editor.
3. If working with the Class diagram, click the *UI Designer* icon on the Designer pane toolbar. If using the Editor, choose *UI Designer* on the Editor’s right-click menu.

NOTE: You can also use View | Designer View | UI Designer

The UI Builder looks to the *Editor* (not the Class diagram or Explorer selection) for the class to work on. It accepts the class for visual editing only if it is declared *public* and contains the required initializing operation (see “What is a visually editable class?” on page 275).

The Menu Designer view

When a container class you are working on contains a menu component, develop the menu visually in Menu Designer view.

To work on a menu component visually in Menu Designer view:

1. Select the menu component in the UI Component Explorer (Explorer pane). The Menu Designer icon is enabled in the Designer pane toolbar and the main menu (View | Designer View | Menu Designer).
2. Click the Menu Designer icon on the Designer pane toolbar, or choose *Activate Designer* on the menu component’s right-click menu in the UI Component Explorer.

TIP: Double-click on a menu component in the UI Component Explorer to switch to Menu Designer view. To switch back to UI Designer view, double-click on a visual UI component instead of a menu component.

Adding components from the Toolbox

The Toolbox is available in UI Designer and Menu Designer view. It contains the available set of components for building a user interface. If the Toolbox is hidden, display it using **View | Main Panes | Toolbox**.

For a reference for the standard list of available UI components provided in the Toolbox, search Online Help for *Toolbox*.

Let's look at how to add components from the Toolbox. Suppose, for example, that you are starting with a visually editable class that extends *JDialog*, and you want to add a panel to the dialog frame.

1. Click the Swing Containers page of the Toolbox (in this case, the component we want is found on that page).
2. Click the JPanel icon in the Toolbox.
3. Move the pointer over the Designer pane. The pointer changes to the Drop shape, and the JPanel changes to show the current Layout (the default layout is BorderLayout).
4. Click inside the container frame to place the panel.

NOTE: You can customize the contents of the Toolbox. For more information, see “Customizing the Toolbox component palette” on page 300.

Editing component properties

Every UI component has a set of properties... component attributes that control how a component behaves at runtime. Properties can be edited at design time and manipulated with code at runtime.

Properties of UI components appear in the Inspector when a component is selected in UI Designer or Menu Designer view in the Designer pane. Use the Inspector to edit default property values during development.

To view the Inspector:

1. Select a visual component in the Designer pane (UI Designer view), or a visual, nonvisual, or menu component in the UI Component Explorer.
2. Press **Alt+Enter** or choose *Properties* on the right-click menu.

Exposure levels of properties

The Inspector displays several levels of properties for UI components:

- **General:** The Inspector displays properties not flagged as *Hidden* or *Expert* in the BeanInfo class of the selected component.
- **Expert:** The Inspector displays properties flagged as *Expert* in the selected components BeanInfo class, plus all the properties shown in the General level.

- **Hidden:** The Inspector shows only properties flagged *Hidden* in the selected component's BeanInfo class.
- **Read-only:** The Inspector shows only properties flagged *Read-only* in the selected component's BeanInfo class.

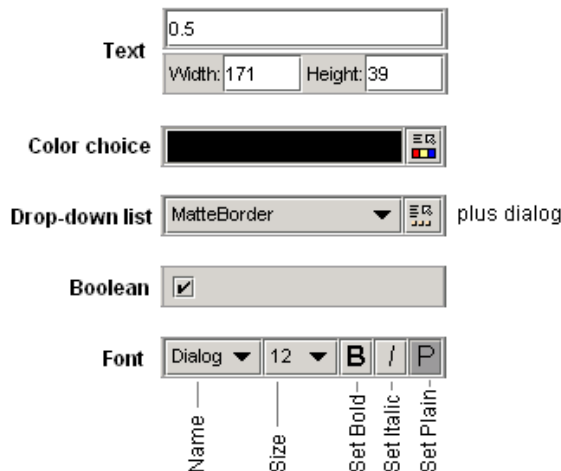
To change the exposure level in the Inspector:

1. Right-click on any property in the *Name* column of the Inspector.
2. Choose *Property Filter* on the right-click menu, followed by the desired exposure level on the fly-out menu.

Property editors

There are several types of property editors ranging from simple text fields to drop-down lists, to color choosers, to entire dialogs (for Layout Manager for example). The following figure illustrates the different types.

Figure 74 UI Builder property editors



As you edit properties, relevant changes are written to the source code in real time.

Some property editors have buttons leading to dialogs: Bean Chooser or File Selection for example. Online Help is provided for these dialogs.

Changing the layout manager for UI components

Java UI containers use a special *layout manager* object to determine how components are placed and sized at runtime. This means you don't have to worry about something looking good on one operating platform but not on another.

When you add a component to a container (or Applet), the container uses the layout manager to determine where to put each component. Java defines the `java.awt.LayoutManager` interface that is implemented by the five main layout manager classes *BorderLayout*, *CardLayout*, *FlowLayout*, *GridLayout*, and *GridBagLayout*. (There are several other layout managers, but these are the most often used.)

The *layout* property

In Together, the layout manager used by a container component (frames and panels, for example) is shown visually as a property of the component. The *layout* property is shown in the Inspector whenever you select a container in the UI Designer or the UI Component Explorer.

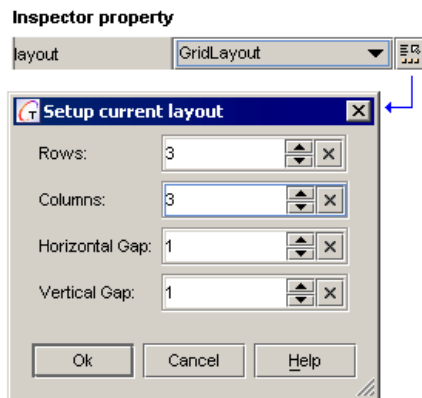
To view and change the layout property:

1. Select the component in the UI Components Explorer, or in the UI Designer view in the Designer pane.
2. If the Inspector is not visible, use the View menu or press **Ctrl+Alt+I** to show it.
3. Scroll the Inspector properties list and locate the *layout* property.
4. Choose the desired layout manager from the drop-down list.

Using layout setup

The *layout* property includes a special editor dialog you can use to visually set the parameters for each layout. For example, if using *GridLayout* set the number of rows and columns and the size of the horizontal and vertical gaps between them.

Figure 75 The layout setup dialog



Layout-specific setup dialog

The visual setup in shown in the figure results in the following statement in the source code (when used for a JPanel component):

```
jPanel1.setLayout(new java.awt.GridLayout(3, 3, 1, 1));
```

Supported layout managers

The following layout managers are supported as a property of UI containers and display as choices in the *layout* property of the inspector:

- **BorderLayout:** (java.awt.BorderLayout) Organizes the container into 5 regions: *North*, *South*, *East*, *West*, and *Center*. It is not necessary to use all regions. Default layout manager for frames.
- **GridLayout:** (java.awt.GridLayout) Divides a container into a specified number of rows and columns to form a grid of cells. Each cell is limited to a single component. Most useful when all components in the container are nearly equal in size. Good for laying out panels.
- **GridBagLayout:** (java.awt.GridLayout) The most robust of the layout managers. It enables users to use components of different sizes and precisely lay them out in the container.
- **CardLayout:** (java.awt.CardLayout) The concept is similar to breaking the container into a deck of cards, each card having its own layout manager. Flipping through the cards displays a different set of components. Conceptually analogous to HyperCard on the Macintosh, and Toolbook on Windows.
- **FlowLayout:** (java.awt.FlowLayout) Arranges components from left to right until no space remains, then begins a new row. Mainly useful for button layouts. Default manager for panels and Applets.
- **OverlayLayout:** (javax.swing.OverlayLayout) Allows components to be physically placed one on top of another.
- **BoxLayout:** (javax.swing.BoxLayout) Aligns components from either left to right or top to bottom in a container. The class is not generally used directly as the Box class provides methods for screen design.
- **Null:** A special design-time layout that disables any automatic control of the component layout, allowing for easier visual prototyping at design time. Not generally suitable for runtime.

You are not limited to these layout managers. Use your own custom layout managers or third-party layout managers, but you will need to use code to implement them rather than using a visual property in the Inspector.

Placing components into containers with different layouts

Once you have set the *layout* property for a container, place components into it (see “Adding components from the Toolbox” on page 279). Visually resize containers such as panels by selecting the component and dragging the corners or sides.

When you select a component in the Toolbox and then hover the pointer over the container, the visual image of the areas of the container that can accept the component change to a cross-hatch pattern. Figure 76 shows an example of a JFrame using BorderLayout. You can clearly see the North, South, East, West, and Center regions of the container.

The mouse pointer changes to a “drop” shape whenever you hover over an area of the container that can accept the component you are placing. Figure 77 shows the same JFrame as Figure 76, with a button being placed into the JPanel in the South region of the frame. As you can see, only the pane displays the cross-hatching. If the pointer were moved over the JFrame, cross-hatching would show for the other regions of the frame, but the JPanel would not, as in Figure 78.

The main point to remember when working with different layouts in a container is that Together will only allow you to place a component in a location that the current layout manager supports, and those locations will show the white cross-hatching pattern (see Figure 79.)

Figure 76 JFrame using BorderLayout shows where components can be placed

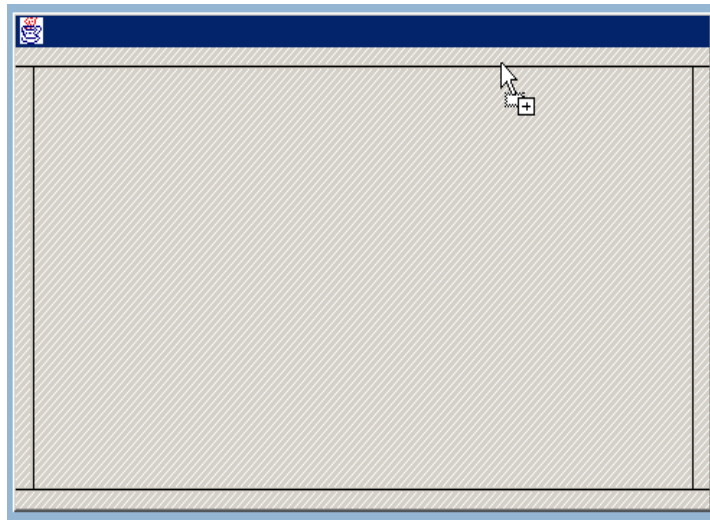


Figure 77 Placing a second JButton into a JPanel using FlowLayout

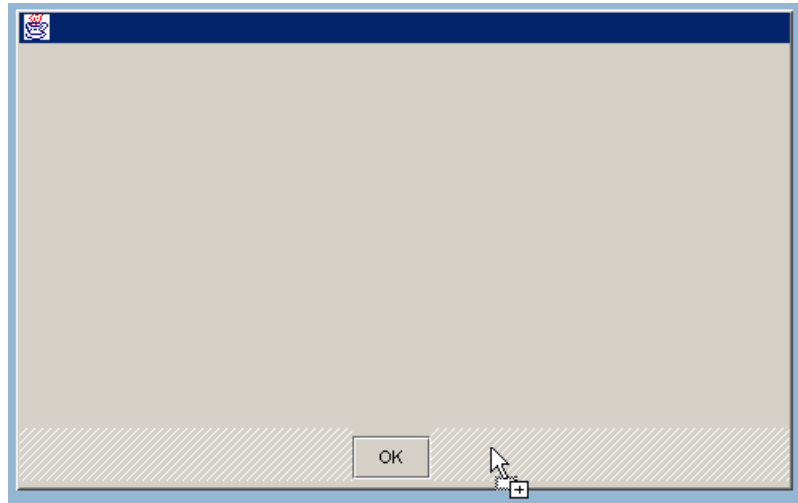


Figure 78 Pointer hovering outside the JPanel

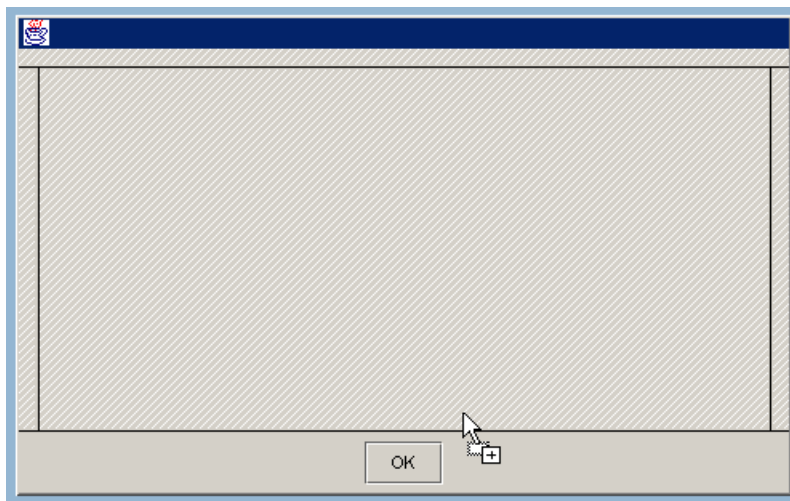
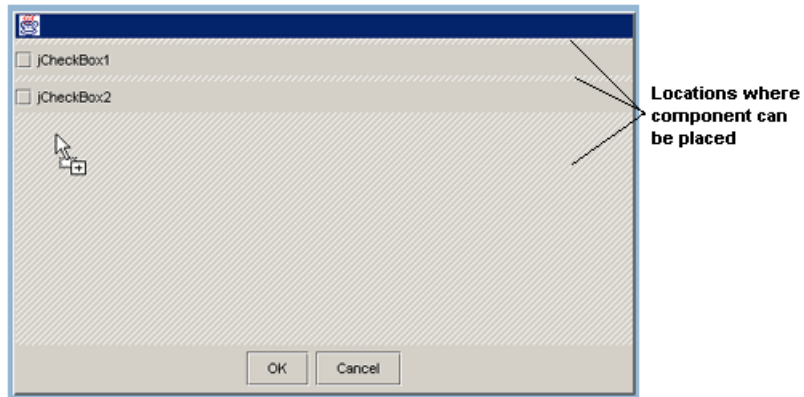


Figure 79 Valid placement locations for any layout manager always display the cross-hatching pattern



Where to learn more about layout managers

There is a great deal of information published on the Web about the basics as well as advanced techniques for using layout managers in Java. One useful tutorial is available (at the time of publication) on Sun's website at:

<http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html>

Other tips and tricks for working on UI components

This section provides some information that you may find useful as you learn your way around the UI Builder.

Selecting multiple components

To select multiple components in the UI Designer, click to select the first component, press and hold down **Ctrl**, and then click on the other components you want to select.

In the UI Component Explorer, use **Shift + click** to select multiple contiguous components, or **Ctrl + click** to select multiple non-contiguous components.

Changing the look and feel

Change the look and feel of frames using *Look and Feel* item on the right-click menu of the container in the UI Designer. The look and feel options are:

- Metal (the default)
- CDE/Motif
- Windows

Changing the order of components

Change the appearance order of components or containers using the *Order* item right-click menu of a selected component or container.

Changing alignment, spacing, and distribution

Change the alignment, spacing, and distribution of components in a frame container using the *Layout* item on the frame's right-click menu.

TIP: To get the right-click menu for a frame when it contains other components such as panels, right-click on the title bar of the frame in the UI Designer.

The *Layout* menu present the following options:

- **Align:** Presents a submenu enabling you to specify horizontal alignment values of Left, Center, and Right, and vertical alignment values of Top, Middle, Bottom.
- **Space Evenly:** Spaces components evenly either horizontally or vertically.
- **Distribute in Container:** Evenly distributes components in the entire container either horizontally or vertically.

Creating event handlers for UI components

Each component has a set of event handlers. Code defines what behavior the component should have when and if an event occurs.

To create an event handler:

1. Select the component in UI Designer view or the UI Component Explorer.
2. Click the *Events* page of the Inspector.
3. Click in the edit field next to the event for which you want to write event handler code. A default identifier for the event handler block appears in the edit field.
4. Press **Enter** to create the event handler block in the source code. The Editor scrolls to the newly inserted event handler line.

Designing Menus

Use the UI Builder feature to create both Swing menu components (*javax.swing.JMenuBar* and *javax.swing.JPopupMenu*) and AWT menu components (*java.awt.MenuBar*, *java.awt.PopupMenu*) in applications. When the UI Builder feature activates, the Designer pane has a special **Menu Designer** view for visually developing menu components.

This section covers the following topics:

- Creating a new main menu
- Creating a new popup menu
- Defining menu items including checkbox and radio button items
- Creating nested menus (submenus)
- Enabling, disabling, hiding, and showing menu items
- Defining menu item properties such as icons, mnemonics, and keyboard shortcuts

Creating a new menu bar component

Create main menu (menu bar) components in much the same way you create other components. There are minor but important differences which are covered in detail in this section.

To create a new menu bar component:

1. Open the container frame that will contain the menu in UI Designer view in the Designer pane (see “The UI Designer view”).
2. Open the *AWT Menu* or *Swing Menu* page depending on the type of components you are using in the Toolbox.
3. Click *MenuBar* (AWT) or *JMenuBar* (Swing)
4. Move the pointer over the Designer pane.
5. Click anywhere on the *background* of the Designer pane. (You cannot click inside the frame component).
6. Expand the *Menu Components* node and select the new menu component (*MenuBar1* or *JMenuBar1*) in the UI Component Explorer.
7. Optionally change the name of the menu component in the Explorer (see “Renaming menu component identifiers visually” for more information).

Setting a menu bar into the container frame

A frame component does not display a menubar until an appropriate *set* statement is placed into the source code (see “Understanding the code” on page 289). The Inspector provides an easy way to set a menu bar component into a container frame.

The *menuBar* property specifies which menu bar component should be set in the frame. The editor for this property presents a dialog enabling you to select a menu bar component for the frame from a drop-down list of all menu bar components that have

been declared in the current source code file. When you select a menu bar component, an appropriate *set* statement generates or updates in the code and the selected menubar appears in the visual representation of the frame in UI Designer view.

TIP: View an example of how a menu bar is set in the main JFrame class created by the Object Gallery's *Application* pattern (File | New: User Interface > Application)

Understanding the code

When you first place a menu component visually, its declaration is written in source. For example, if you place a JMenuBar component, the following line is written:

```
private JMenuBar JMenuBar1 = new JMenuBar();
```

When you add menu items to the menu bar, their declarations are also written to the source file. You might end up with the following code for a small menu:

```
private JMenuBar JMenuBar1 = new JMenuBar();
private JMenu JMenu1 = new JMenu();
private JMenu JMenu2 = new JMenu();
```

The code will also contain some *add* and *set* statements such as:

```
JMenuBar1.Add(JMenu1);
JMenuBar1.Add(JMenu2);
JMenu1.SetText("File");
JMenu1.SetMnemonic("F");
JMenu2.SetText("Edit");
JMenu2.SetMnemonic("E");
```

At this point the *JMenuBar1* component appears as an attribute of the container class when the Designer pane is in Diagram Designer view. But the menu does *not* appear on the WYSIWYG image of the frame in UI Designer view. In order for the menu to appear there, add an appropriate *setJMenuBar* statement to the code. In the foregoing example, the following statement is required after the *add* statements:

```
JMenuBar1.Add(JMenu1);
JMenuBar1.Add(JMenu2);
setJMenuBar(JMenuBar1);
JMenu1.SetText("File");
JMenu1.SetMnemonic("F");
JMenu2.SetText("Edit");
JMenu2.SetMnemonic("E");
```

Write the code yourself or use to the *menuBar* property of the frame component which generates the necessary code.

Creating a new popup menu component

The steps for creating a new popup menu are the same as for creating a new main menu, except that you select a popup menu component in the Toolbox instead of a menu bar component.

Displaying a popup menu at runtime

The code for calling a popup menu from a component is somewhat more complex than that required to set a menubar component. Thus, there is no Inspector property for components that “attaches” a popup menu. The following code fragment demonstrates one way to invoke popup menus at runtime:

```
public void commentFieldMouseReleased(MouseEvent e) {
    if(e.isPopupTrigger()){
        jPopupMenu1.show(commentField,e.getX(),e.getY()) ;
    }
}
```

(where *commentField* is the name of component that pop up menu was called on, and *jPopupMenu1* is the name of a popup menu component for *commentField*).

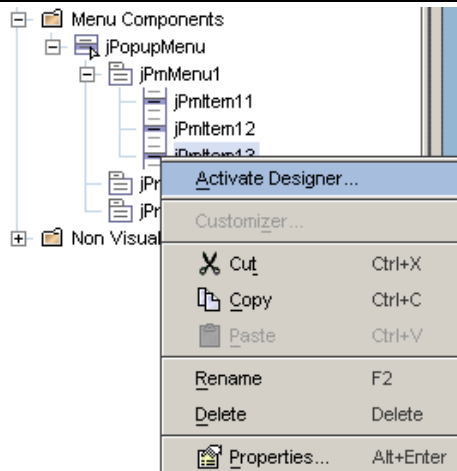
Also add a mouse listener using the Inspector (See “Creating event handlers for UI components” on page 287.). For example:

```
commentField.addMouseListener(
    new MouseAdapter() {
        public void mouseReleased(MouseEvent e) {
            commentFieldMouseReleased(e); }
    });
```

Defining menus visually

When opening a UI container class in the Designer pane, a visual representation of the container appears in the Designer, and the contained components appear listed in the UI Component Explorer. The *Menu Components* node of the Explorer lists all the menu and menu item components.

Figure 80 The UI Component Explorer and its right-click menu



To activate the *Menu Designer* view:

1. In the Explorer, select any menu, popup menu, or menu item component
2. Right-click and choose *Activate Designer* or click the Menu Designer icon on the Designer pane toolbar.

Defining new menu items

When you create a new menu component, the first item is automatically created and labeled *<new item>*. To define the new menu item:

1. Double-click *<new item>* to activate in-place editing of the item name.
2. Type the text of the menu item optionally using the syntax described in the next section.
3. Press **Enter** to accept the entry.

For second-level main menu items and popup menu items, a *<new item>* item is added to the end of the menu. Repeat the above steps to define new menu items.

For top-level items in a main menu component, *<new item>* items are created to the right of each item.

To create checkbox or radio-button menu items, see the respective sections in this chapter: “Defining a checkbox menu item” and “Defining a radio-button menu item”

Menu definition syntax and syntax helpers

Define a number of properties of new menu items in the items’ in-place edit field using a special syntax. This is very productive when creating a menu structure from a specification. The properties you can define are:

- Menu text
- Mnemonic (accelerator)
- Keyboard shortcut
- Enabled
- Visible
- Checked (for checkbox and radio button items only)

The syntax looks like this:

```
<menu &text>;<shortcut>;<enabled>;<visible>[;<checked>]
```

Examples:

```
&New file;Ctrl+F;0;1
```

In the above the menu text is *New file*, the mnemonic accelerator is *N*, the shortcut is *Ctrl+N*, the item is *disabled* (0, or Boolean *false*), and the item is *visible* (1 or Boolean *true*).

```
&Toolbox;Ctrl+Alt+T;1;1;0
```

In the above, the menu text is *Toolbox*, the mnemonic accelerator is *T*, the shortcut is *Ctrl+Alt+T*, the item is *enabled* (1), *visible* (1), and *unchecked* (0).

NOTE: Menu items defined as *not visible* still display in the Menu Designer view, but their background color is white instead of gray.

Using the menu definition syntax helpers

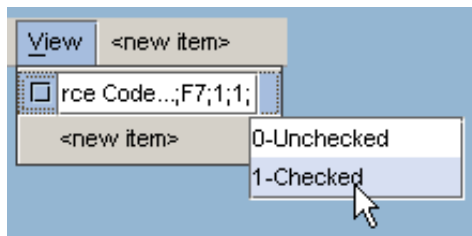
You do not have to memorize the in-place menu definition syntax. Together provides a set of syntax helpers that guide you through your in-place editing options.

To use the syntax helpers:

1. Type in the menu text followed by a semi-colon. A context-sensitive pop-up helper menu appears after a short pause.
2. Select the desired option from the menu navigating with the mouse or arrow keys.
3. To set further properties, type another semi-colon and select the desired option from the next helper.
4. Repeat step 3 until no further pop-up appears.

TIP: Press **Enter** in the helpers to select the default

Figure 81 Design syntax helper for checkbox item



Defining a checkbox menu item

Optionally define menu items below the menu bar level and popup menu items as checkbox items in both AWT and Swing. Checkbox items provide on/off true/false toggles.

To define a checkbox menu item:

1. In Menu Designer view, select the *<new item>* you want to define as a checkbox menu item.
2. Right-click and choose *Checkable Item*
3. Double-click to activate in-place editing and proceed as you would for any menu item.

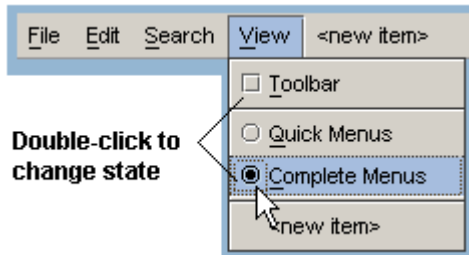
TIP: To revert to a regular menu item, right-click on the item in the designer and choose *Non-checkable item*.

Changing the default state of a checkbox

To set the checked state of a checkbox type menu item, use the in-place editing field. Double-click to activate editing, then enter the desired Boolean value (0 or 1) in the syntactical position for *checked* state. For more information, see “Menu definition syntax and syntax helpers.”

Once a checkbox menu item has been created, change the checked state of the checkbox by double-clicking on the checkbox itself.

Figure 82 Changing menu item state at design time



Defining a radio-button menu item

The menu items of a menu bar, and popup menu items can optionally be defined as radio-button items (Swing only). Radio-button items provide mutually exclusive choices.

To define a radio button menu item:

1. In Menu Designer view, select the <new item> that you want to define as a radio-button menu item.
2. Right-click and choose *Toggle Radio Item*
3. Double-click to activate in-place editing and proceed as you would for any menu item.

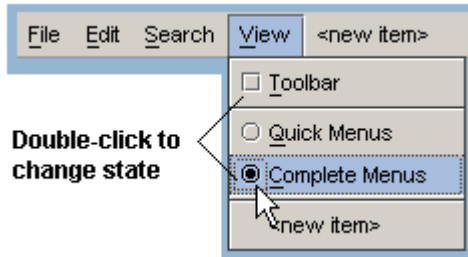
TIP: To revert to a regular menu item, right-click and choose *Non-checkable item*.

Changing the default state of a radio-button

Use the in-place editing field to set the selection state of a radio-button type menu item. Double-click to activate editing, then enter the desired Boolean value (0 or 1) in the syntactical position for *checked* state. For more information, see “Menu definition syntax and syntax helpers”.

Once a radio-button menu item has been created, you can quickly change the selection state of the radio button by double-clicking on the button itself.

Figure 83 Changing menu item state at design time



Creating nested menus

You can create any number of nested menu levels (sometimes called “submenus”) beneath any menu item or popup menu item. (Note that good user interface design practice generally limits the number of levels to about three.)

To create a nested menu:

1. Select the existing menu item you want as the parent of the nested menu.
2. Right-click on the parent menu item and choose *Insert Nested Menu*. This creates the new level and inserts a *<new item>* menu item.
3. Define the menu items for the nested menu in the same manner as for any other menu item.

Inserting separators

Insert separators between menu items to create groups of related commands. There are three ways to do this:

- Using in-place editing
- Using the menu item right-click menu
- Using the Toolbox

Creating separators using in-place editing

This is the quickest way to create a separator when creating a new menu.

1. Create or select a *<new item>* item at the place where the separator should be.
2. Activate in-place editing and type a dash (-) and then press Enter.

Creating separators using menu item’s right-click menu

1. Select the menu item you want to appear immediately *below* the new separator.
2. Right click and choose *Insert Separator*.

A separator is inserted *above* the selected menu item.

TIP: Select the menu item and type a minus sign “-”

Creating separators using the Toolbox

Select a separator component in the AWT Menu or Swing Menu page of the toolbox and drop in into the menu at the desired location. See “Defining menu items non-visually” on page 297 for more information about using the Toolbox.

Moving menu items

Once you have a menu structure developed, move menu items to different locations using drag-and-drop in the Menu Designer view (Designer pane). As you drag an item, a red bar appears at valid drop target locations.

Defining menu item properties

Every menu item has a set of properties that you can modify at design time. As we saw in earlier sections, some properties can be set as you visually create new items, and some can be modified using the menu item right-click menu. The full set of properties is listed in the Inspector when a menu item is selected in Menu Designer view.

To view the Inspector properties of a menu item:

1. Select the item in the Menu Designer view (Designer pane).
2. Press Alt+Enter or click *Properties* on the Main Toolbar to display the Inspector.

TIP: You may find it convenient to undock and resize the Inspector.

Disabling and hiding menu via the right-click menu

As mentioned in, “Define a number of properties of new menu items in the items’ in-place edit field using a special syntax. This is very productive when creating a menu structure from a specification. The properties you can define are:”, you can use a special syntax in the in-place edging field of to enable/disable or show/hide menu items at design time.

Modify the settings in the same way, but you may find it more convenient to change the default state of menu items at design time using the right-click menu of any menu item.

To make a menu item hidden by default:

1. Select the item in the WYSIWYG menu hierarchy and right-click.
2. Choose *Hide Item* on the right-click menu.

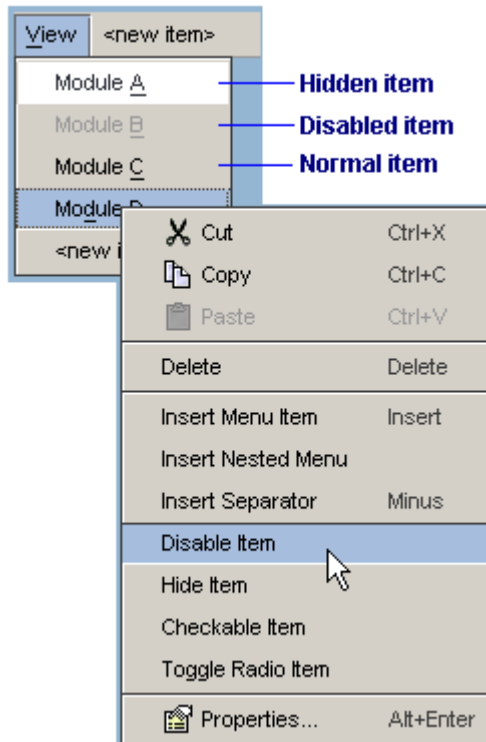
Menu items defined as hidden are still visible at design time; the background color of these items turns to white.

To make a menu item disabled by default:

1. Select the item in the WYSIWYG menu hierarchy and right-click.
2. Choose *Disable Item* on the right-click menu.

The foreground color of items defined as disabled turns to dark gray.

Figure 84 Setting design time properties for menu items using the right-click menu



Adding icons to menus

Menu items display icons next to them. You can show static and rollover images using existing image files in BMP, GIF, or Jpeg format. Images are defined as properties of menu items in the Inspector.

TIP: One fairly widespread convention for the use of icons in menus is that menu items having a parallel function on a toolbar should display the same icon as the toolbar button. Other menu items do not show icons.

Icon properties

Optionally define in the Inspector the following icon properties :

- ***disabledIcon***: the image to display when the item is disabled but not selected
- ***disabledSelectedIcon***: the image to display when the item is disabled and selected
- ***icon***: the image to display when the item is enabled but not selected
- ***rolloverIcon***: the image to display when the mouse pointer hovers over item the item when the item is not in the selected state. Note that the *rolloverEnabled* property must be checked (i.e., *true*) in order for this to work at runtime.

- ***rolloverSelectedIcon***: the image to display when the mouse pointer hovers over the item and the item is in the selected state. Note that the *rolloverEnabled* property must be checked (i.e., *true*) in order for this to work at runtime.
Note that the *selected* property must be checked (i.e., *true*) in order for this to work at runtime.
- ***selectedIcon***: the image to display when the item is enabled and selected. Note that the *selected* property must be checked (i.e., *true*) in order for this to work at runtime.

To display one or more images for the menu item:

1. Select the menu item in the Menu Designer view (Designer pane)
2. Display the item's properties in the Inspector (Alt + Enter or main toolbar).
3. In the Inspector, select one of the iconic properties described earlier in this section. Use the browse button in this field to select an image file.
4. Repeat this process for any or all the other iconic properties as required to get the effect(s) you require.

TIP: According to Sun's user interface guidelines, the standard size for menu icons is 16x16 pixels.

Defining menu colors and font

Modify the default foreground and background colors and the default font properties of menu items.

- Use the *foreground* property in the Inspector to change default foreground (text) color
- Use the *background* property in the Inspector to change default background color
- Use the *font* property in the Inspector to change default font properties of the selected menu item.

Defining menu items non-visually

There is an alternate technique to creating new menu items once you have defined a menu bar or popup menu component. Use the Toolbox to select different types of menu items and place them into the menu hierarchy displayed in the UI Component Explorer. This technique may be easier if you have to add a new item to a large existing main menu system.

To add a menu item in the Explorer with the Toolbox:

1. Expand the Menu Components node of the UI Builder tab of the Explorer (i.e. the UI Component Explorer), and locate the menu item that you want as the parent of the new item.
2. Select the Toolbox page that contains the menu component you want to add (Swing Menu, for example).
3. In the Toolbox, click on the menu component you want to add to the hierarchy.

4. Move the mouse over the parent menu component in the Explorer. (The pointer changes to the Drop shape when you hover over a valid target.)
5. Click on the node of the parent item to create the new menu item with a default identifier.

Renaming menu component identifiers visually

As you create new menu items either visually or non-visually, new menu components are declared in source code with default identifiers. If desired, opt to change the default component names (identifiers) to names that are more meaningful and easier to maintain in the future.

To rename a component identifier:

1. Select the menu component in the UI Component Explorer
2. Press **F2** or right click and choose *Rename*. In-place renaming is activated.
3. In the menu component node, type in a legal Java identifier for the component and press **Enter**.

NOTE: Do not confuse this process with in-place editing of the menu text and properties as described in, “Define a number of properties of new menu items in the items’ in-place edit field using a special syntax. This is very productive when creating a menu structure from a specification. The properties you can define are:”

Renaming components this way automatically changes every occurrence of the component name in the source code file.

Renaming component identifiers in source code

You can optionally rename menu components manually in the source code. However, you will have to change *every instance* of the name in the file. Use the main menu’s Search | Replace command when the Editor pane has focus to accomplish this task.

UI Builder Audits and Metrics

When the UI Builder feature is activated, special audits and metrics are provided to help you check the quality of your user interface classes. In general, these work like all other audits and metrics. See Chapter 25, “Audits and Metrics” for information on how to use audits and metrics.

UI Audits

When the UI Builder feature is activated, a special *UI Builder Specific* category is added to the Java Audits dialog. It contains a special set of Audits for UI classes that are used to check for problems including conflicting mnemonics in the same menu, use of nonstandard mnemonics in menus, duplicate context menu items, etc. Detailed descriptions of each individual audit are provided in the Java Audit Dialog.

To use UI Builder audits:

1. Make sure your project includes Java GUI classes as defined in this chapter under “Creating UI classes” on page 275.
2. Make sure the UI Builder feature is activated as described in this chapter under “Activating the UI Builder” on page 273.
3. On the main menu choose: **Tools** | **Audits** to open the Java Audits dialog.
4. Locate the *UI Builder Specific* category in the list of audit titles, and expand the category node to see all available audits.
5. Check the audit(s) you want to run against your project and click the **Start** button.

UI Metrics

When the UI Builder feature is activated, a special *UI* category is added to the Java Metrics dialog. The first release of the UI Builder feature provides a single metric *Number of Controls in Form* that returns a number count of controls in a form type component (such as a JFrame derived class). Future releases will provide additional metrics.

To use UI Builder audits:

1. Make sure your project includes Java UI classes as defined in this chapter under “Creating UI classes” on page 275.
2. Make sure the UI Builder feature is activated as described in this chapter under “Activating the UI Builder” on page 273.
3. On the main menu choose: **Tools** | **Metrics** to open the Java Metrics dialog.
4. Locate the *UI* category in the list of audit titles, and expand the category node to see all available metrics.
5. Check the metric(s) you want to run against your project and click the **Start** button.

Generating user interface documentation

Together's documentation generation feature provides a special documentation template that you can use to generate documentation for a project's user interface. Generate documentation in HTML, RTF, text format using the standard template, modify the standard template, or design your own template.

See Chapter 17, "Generating Documentation for Together Projects" for more detailed information on documentation generation features.

To generate documentation for your user interface:

1. Open the project containing the UI code you want to document
2. On the main menu, choose Project | Documentation | Generate Using Template. The *Generate Documentation Using Template* dialog appears.
3. Choose the scope of the information to include in the generated documentation.

TIP: To document only user interface classes, open the Class diagram that contains the UI classes and choose *Current Diagram*. If UI classes are in a specific package, select that package in the Model tab of the Explorer and choose *Current Package* (or *Current package with subpackages* if there are classes further down in the package hierarchy).

4. Use the drop-down list of templates and choose `$TG_HOMES\modules\com \togethersoftware\modules\gendoc\templates\UIBuilderReport.tpl` (where `$TG_HOMES` is the root of your Together installation).
5. Choose the output format and output location, then click **OK**.

Customizing the Toolbox component palette

User interface components are just JavaBeans™. The toolbox comes with a default set of standard AWT and SWING bean components (search Online Help for *Toolbox* for a full listing). The bean components are divided into named categories (Swing General, AWT Menu, for example). Each category has a scrolling "page" in the Toolbox that contains the various bean components assigned to that category. You can fully customize the Toolbox's categories and the components each one contains. You can reorder existing categories and add new categories. You can add your own custom user interface components to the new categories you create or to any existing category.

Creating a component archive

Before you try customizing the Toolbox, create one or more JAR files containing the bean components to add to the Toolbox. These can be your own components or components of a third-party library.

To create new categories, a good rule of thumb is to create one appropriately named JAR file for each category page you want to add to the Toolbox. This helps to organize your components... it not required. Add specific beans from any JAR file to any Toolbox category. The JAR file(s) containing custom bean components can reside on any drive accessible to your system.

TIP: Consider adding a *Custom* directory to `$TG_HOMES$/lib` and placing your custom component archives there.

Once you have your component archive(s) ready, you can begin customizing the Toolbox.

Invoking the Toolbox customization dialog

The *Palette Customization* dialog is available from the right-click menu of any category header in the Toolbox.

To access the Palette Customization dialog:

1. Load a visually editable class in the Editor as described in “What is a visually editable class?” on page 275 and “Creating UI classes” on page 275.
2. Activate UI Designer view by clicking the UI Designer icon on the Designer pane toolbar (see “The UI Designer view” on page 277 for more information).
3. Display the Toolbox by using View | Main Panes | Toolbox (or the parallel icon on the main toolbar).
4. Right-click on any of the category headers and choose *Customize Palette* in the Toolbox.

Customizing toolbox categories

Reorder existing component categories, add new categories, or remove existing categories in the *Palette Customization* dialog.

Reordering existing component categories

To reorder existing categories:

1. In the *Categories* list, select the category name you want to move. **Shift+click** to select multiple categories to be moved at once.
2. Using the toolbar immediately above the *Categories* list, click the up arrow or the down arrow to change the position of the selected category in the list. Continue clicking until the selected item(s) are in the desired position.

Creating new component categories

To create a new component category:

1. Using the toolbar immediately above the *Categories* list, click the *New Category* icon.
2. Edit the default name in the *Category Name* field. Press **Enter** to accept the edit.
3. Optionally change the appearance order of the new category as described in the previous section.

Removing component categories

You can remove any existing component category. Removing a category automatically removes all the bean components it contained. The JAR file containing the components is not deleted by the remove operation.

To remove a component category:

1. In the *Categories* list, select the category name you want to remove. **Shift+click** to select multiple categories to be removed at once.
2. Using the toolbar immediately above the *Categories* list, click the *Remove* icon.

Restoring the default component categories

You are not constrained from removing any component category, including the default categories that ship with Together. Should you remove any of the default categories and later want them restored, you can use the *Restore Defaults* button in the *Palette Customization* dialog.

IMPORTANT: If you click the *Restore Defaults* button, you will lose any customizations you may have made to any of the default categories.

Adding custom bean components

Add custom bean components to any existing component category by selecting a category and then selecting a JAR file that contains the components you want in the selected category. Once added to the category, you can change the appearance order of the individual bean components, remove individual bean components from the category, and customize the component names and icons of each bean component.

Adding components to a category

To add bean components to a component category:

1. Select the desired category in the *Categories* list box
2. Click the button labeled *Add Beans to Category*.
3. Click the browse button next to the *Path to JAR* field and in the file chooser dialog, select the JAR file containing the components.
4. If any components in the bean component JAR file require classes in other JAR files in order to work correctly, use Advanced mode of the dialog to add these archives.
5. Click **OK** to close the Add Beans dialog and add the selected JAR file's bean components to the category.

NOTE: All components contained in the archive are added to the category. If there are some you do not want to use, you can remove them as described in the following section.

Removing components

You can remove any component from the category. Removed components do not appear in the Toolbox. Components are not removed from the underlying JAR file.

To remove a component from a category:

1. In the *Categories* list, select the category from which you want to remove components.
2. In the *Components* list, select the component name you want to remove. Shift+click to select multiple contiguous components to be removed at once. Ctrl+click to select multiple non-contiguous components to be removed at once. (Or use the multi-selection operations for your operating system if different).
3. Using the toolbar immediately above the *Components* list, click the *Remove* icon.

Changing component display names

You can change the display name of any component. The naming in the underlying JAR file is not affected.

To change a component's display name:

1. In the *Components* list, select the component you want to rename. Its name is shown in the *Component Name* field.
2. In the *Component Name* field, edit the display name and press Enter to accept the change.

Changing component icons

Each component has two associated icons, termed *large* and *small*. The large icon is displayed in the Toolbox. The small icon is displayed in the UI Component Explorer when the component is used in a user interface you are building in your project. Normally the icons are provided by the developer of the component.

If the component developer does not provide default icons, or if you want to use different icons, you can specify the icon image file you want displayed for each component.

To use custom icon images:

1. In the *Components* list, select the component for which you want to use custom icon images.
2. In the *Choose Icon* group, click the *Select icons* option button.
3. Click the browse buttons next to the *Large icon* and *Small icon* fields in turn, and select the GIF or Jpeg image you want to use for each of the two icon sizes.

CHAPTER 16

USING VERSION CONTROL

Systems that implement version control maintain the current and previous versions of artifacts. This chapter provides an overview of how Together facilitates version control support. In addition, it provides instructions for integrating your version control system with Together, configuring version control settings, enabling version control for a project, and interacting with a version control system.

This chapter includes the following topics:

- “Multi-user development” on page 304
- “Using Together with a Version Control System” on page 305
- “Product-specific VCS notes” on page 314
- “Using Together with ClearCase” on page 318.

Multi-user development

Together delivers support for true multi-user development. You can use version control to:

- Protect your company’s software assets.
- Help team members work together.
- Unobtrusively impose team-wide or enterprise-wide standards.

Together delivers seamless integration with various version control systems without requiring you to artificially and manually split up your model into submodels, and subsequently split those submodels into files for your version control system. Together frees you from the efforts of maintaining proprietary internal repository.

Once you have configured your version control options and associated your project with a version control project, you can easily interact with your VCS through Together. Version control commands are displayed on right-click menus for source elements (such as classes), both in diagrams (as shown below) and on their nodes in the Model tab of the Explorer.

Overview of version control support

Together provides built-in integrations for the leading multi-user version control systems: CVS, SCC, Rational[®] ClearCase[®], and generic providers. The SCC standard is supported by many popular version control products.

CVS 1.11 comes bundled with Together, pre-configured for immediate use. Both an integrated Java CVS client and native CVS clients are included. Together comes pre-configured for CVS pserver mode using the native client. Compiled CVS binaries for the various supported OS platforms can be found in the following locations in the Together installation:

- **Windows:** %TGH%\bin\win32\cvs.exe
- **Linux:** \$TGH\$/bin/linux/i686-unknown/cvs
- **Sun Solaris:** \$TGH\$/bin/sunos/sun4u-sparc/cvs
- **HP-UX:** \$TGH\$/bin/hp-ux/cvs
- **Compaq Tru64:** \$TGH\$/bin/osf1/alpha-alpha/cvs
- **Mac-OS:** \$TGH\$/bin/darwin/cvs

Using Together with a Version Control System

From the Together environment, you can get, add, check in, and check out source code, examine differences, and view properties and history. All of these commands are on right-click menus for visual source elements in diagrams and the Model tab of the Explorer.

IMPORTANT: Version control in Together provides a unified interface for all supported VC providers. This interface contains some abstract VC commands that are not necessarily related to any specific tool. To learn about the typical use of all VCS commands, you must consult the vendor manuals.

Getting started with version control

To work with your version control system (VCS) from Together, complete the following steps:

1. Configure Together for version control.
 - Enable Together's version control integration support in your configuration.
 - Set Version Control configuration options for the VCS you will use with Together.
2. Enable version control for your Together projects, specifying the VCS project or repository to use for each.
3. Interact with the selected VCS. You can use the various right-click menu commands and dialogs for specific CVS operations (add, get, and so on), and/or use the System dialog which Together provides as a client, enabling you to more fully interact with your CVS from the Together environment.

Each of these steps is described in more detail in the sections that follow.

Configuring Together for version control

The Version Control node of the Options dialog provides configuration settings that enable Together to work with your version control system. Your VCS should already be installed and operational before you configure Together.

Remember that Together's configuration system is multi-level. This means you can set up versioning options to apply globally (Default level), or to a specific project (Project or Diagram level). For more information on multiple configuration levels, see Chapter 2, "Setting Your Personal Preferences".

Use the following steps to access version control options:

1. Choose **Tools | Options | <level>** from the main menu.
2. Select **Version Control** among the options listed on the left pane of the dialog box.
3. Click the **Levels** button to select the configuration level (a project must be open).

Alternatively, you can access the options with these steps:

1. Select **Project | Project Properties** from the main menu.
2. Click the **Options** button near the bottom of the dialog box. (If that button is not visible, click **Advanced** to open the full dialog window.) The resulting Version Control Options dialog box displays. (See Figure 85.)

Enabling version control support

To enable or disable Together's version control support:

1. Choose **Tools | Options | <level> - Version Control** on the main menu.
2. Click the **Levels** button to select the configuration level (a project must be open).
3. Check the *Version Control enabled* box to enable version control support in Together. (Clearing that box disables versioning support.)
4. Click **Apply** to immediately effect the change, or wait until you set the options for your version control system as described in the subsequent sections.

NOTE: Version control support is enabled by default.

Choosing which VCS to use

To use CVS pserver mode with the native client, skip this section (it is set up by default). Otherwise, choose which system you want to work with. You should still be in the Version Control node of the Options dialog.

To choose a version control system:

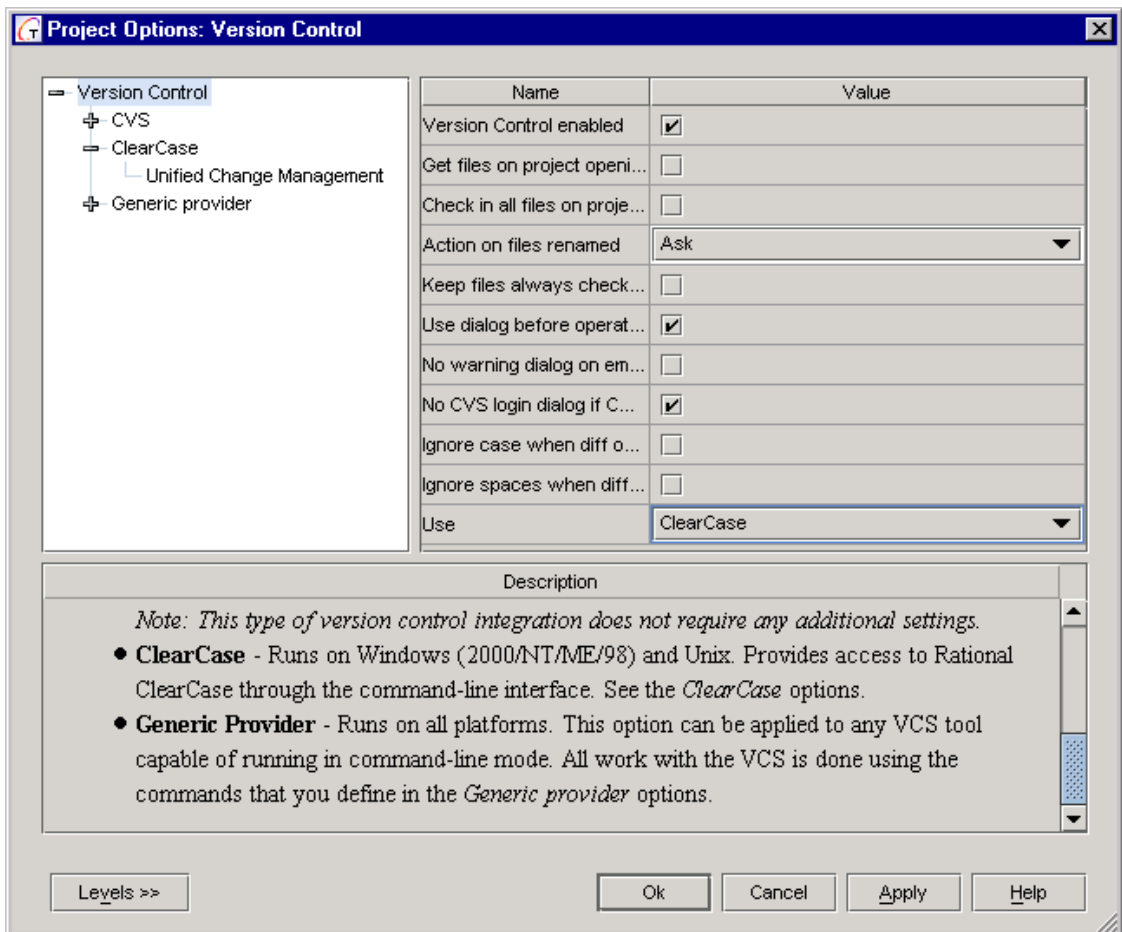
1. In the Options dialog, go to the *Version Control - Use* option.
2. Select the desired VCS from the drop-down list. The choices include:
 - **CVS:** Installed with Together and set up by default for version control integration; runs on all platforms.
 - **SCC:** Runs only on Windows (NT/2000/98). This option causes Together to use the SCC provider registered on your machine (INTERSOLV PVCS Version Manager, Microsoft Visual SourceSafe, Rational Software ClearCase, or another

SCC provider). This option will work only if the additional library called "coroutine" is installed (it is installed automatically with Together on Windows systems). This library is used as a Java-to-Win32 bridge to work with the Win32 SCC provider's API.

NOTE: This type of version control integration does not require any additional settings.

- **Generic Provider:** Choose this option for custom VCS interaction. Runs on all platforms. This option can be applied to any VCS tool capable of running in command-line mode. All work with the VCS is done using the commands that you define in the Generic Provider options.
- **ClearCase:** Runs on Windows (2000/NT/ME/98). Provides access to Rational ClearCase through the command-line interface.

Figure 85 Version control project options.



Configuring system-specific version control options

After choosing the VCS, set system-specific options so that Together works with the selected versioning system. The *Version Control* node of the Options dialog presents several option subnodes corresponding to the different choices in the *Use* option. Set options only for the system you are using and ignore the others. For example, if you choose CVS in the *Use* option, set the options under the CVS node, ignoring the other nodes.

Setting up CVS options

Refer to CVS documentation to learn more about CVS settings. You can find documentation on CVS in the file `$TGH$/bin/win32/cvs.html`, or refer to http://www.gnu.org/manual/cvs/html_chapter/cvs_20.html.

To set up CVS options:

1. If you connect to the repository via dial-up or similar non-persistent connection, check *Go offline by default*. Otherwise, leave this option unchecked.
2. Choose the connection method and client used in your version control integration in the *CVS type* field:
 - Local or LAN-based repository using the native client
 - Pserver protocol using the native client
 - Pserver protocol using built-in Together support
 - Server protocol using built-in Together support

NOTE: When using pserver method with Win 95/98, keep in mind that environment variables required to locate `.cvspass` file (`CVS_PASSFILE`, `HOME`, `HOMEDRIVE`, `HOMEPATH`) are not provided in these operating systems. Thus, to make it possible to work with cvs, it is necessary to define `HOMEDRIVE` and `HOMEPATH` (or `CVS_PASSFILE`, `HOME`) in the `autoexec.bat` file. Concatenation of these strings allows Together to locate the `.cvspass` file.

3. If you access the CVS server remotely and want to compress traffic to and from the server, check the *Compress traffic* box and specify the compression level (minimum 1, maximum 9) in the *Compression level* field.

NOTE: As of this writing, traffic compression applies to the integrated Java CVS client only.

4. Check the box *Use external Diff tool* to examine differences in your favorite application (instead of using the CVS diff feature), and specify the fully qualified path in the field *Name of External Diff executable*. The default external tool comes bundled with Together: `$TGH$/bundled/examdiff/examdiff`.

Setting up CVS Local or LAN-based repository

1. In the Options dialog, choose *Version Control - CVS - Local repository (via native client)*.
2. In the *Name of executable cvs* field, specify the fully qualified path to the cvs client. Note that cvs clients are installed under `$TGH$/bin/` in folders corresponding to the supported operating systems.

3. In the *Repository* field, specify the fully qualified path to the repository.

Setting up CVS Pserver connection

Together enables pserver connection to the remote repository using the native or built-in CVS client. The settings for both options are similar.

1. In the Options dialog, choose *Version Control - CVS - Pserver protocol (via native client)*, or *Version Control - CVS - Pserver protocol (via built-in Together client)*.
2. For the native client, specify the fully qualified path to the CVS client in the *Name of executable cvs* field. If the built-in client is selected, skip this step.
3. In the *Repository* field, enter the CVS repository as you would in the pserver section of the CVS login command. For example, if your login command begins with:

```
cvs login :pserver:/cuthbert@ourCVS-host:/repository_alias
```

then in the *Repository* field, enter

```
/repository_alias
```

NOTE: Be sure to include the initial forward slash character.

4. In the *Server name* field, specify the host name or a valid alias for your CVS server. This is the same host for your CVS login command. For example, if your login command begins with:

```
cvs -d :pserver:/cuthbert@ourCVS-host
```

then in the *Server name* field you must enter

```
ourCVS-host
```

5. In the *Port* field, specify the correct port for your CVS server. Consult your CVS administrator for the correct port number if you are not sure.

Setting up direct connection to the server

1. In the Options dialog, choose *Version Control - CVS - Server protocol (via built-in Together client)*.
2. In the *Host* field, enter the host name or a valid alias for your CVS server.
3. In the *Port* field, specify the correct port for your CVS server. Consult your CVS administrator for the correct port number if you are not sure.
4. In the *Command* field, specify the server command.

Setting up an SCC-compliant versioning system

Together provides support for SCC-compliant version control systems. This support is limited to Windows operating systems.

Together has been tuned and tested to support the following SCC-compliant version control systems:

- PVCS

- StarTeam
- Perforce
- Continuous

The other SCC-compliant versioning systems may be used with Together, but only those mentioned are currently tested.

This type of version control integration does not require any additional settings. To configure other versioning tools, choose the *Generic provider* option.

Coroutine classes

SCC version control support requires installation of Coroutine classes and .dll libraries. The Together installation program for Windows automatically installs these files and updates the environment classpath.

NOTE: Make sure you log on to your Windows NT or Windows 2000 computer with full Administrator rights before installing Together.

After Together installation, you can check to confirm that Coroutine classes have been installed in the directory: `$TGH$/lib/coroutine/com/neva/`. Check to be sure your classpath includes this path.

If, when using the SCC version control feature, you get an error message that Coroutine cannot be initialized, it means that Coroutine classes were not found where expected. If for some reason Coroutine is not installed, you can install it separately by running: `$TGH$/bin/win32/jcinst.exe`. If Coroutine is installed and the error message persists, check that your classpath points to the Coroutine directory.

Setting SCC Version Control options

To set global SCC options, do so with or without opening a project. To set SCC options for a project, open the project.

To set SCC options:

1. Choose **Tools | Options | Version control: Use**.
2. Choose SCC from the drop-down list.

NOTE: As already mentioned, the supported SCC-compliant versioning tools do not require any additional tuning.

3. If using another (non-supported) SCC system, expand the *Generic Provider* node.
4. Set the options of the selected node to conform to your version control system. Explanations are provided in the *Description* area of the Options dialog for each selected option.

Switching among different SCC providers

If you have installed several SCC providers on your system, for example SourceSafe and PVCS VM, you probably have to change registry entries in order to work with a specific source control system.

All SCC providers are listed in the values of the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\
InstalledSCCProviders

Using Copy and Paste commands, copy the value of your preferred SCC provider key to the following value:

- Key: HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider
- Value: ProviderRegKey

The one that is stored in this value will be used by Together as well as by other tools such as DevStudio.

Tuning SCC support for Visual SourceSafe

Before setting up support for Visual SourceSafe, be sure you have installed VSS Explorer. You must have SSSCC.DLL in order to use SCC. Check your SRCSAFE.INI file in the directory where VSS Explorer is installed. If you are using several SourceSafe databases you might want to select one of them to work from Together. If you are using only one database, you can skip this section.

SCC interface functions ignore the settings kept by VSS Explorer in the registry and use the ones stored in the SRCSAFE.INI file instead. If you ever want to switch the default database used by the source safe SCC interface, you should modify the values the of following keys:

- Data_Path
- Users_Path
- Users_Txt

Each of these points to a directory where the SourceSafe database is located or to a file in this directory. You must type the path to your preferred database in each of these keys in order to work with it.

WARNING! Visual SourceSafe version 5.0 is not supported any more. Use version 6.

Enabling version control for projects

Enabling version control and setting up system-specific configuration options in the Options dialog only activates version control integration and prepares Together to interact with your VCS. The configuration process does not set up version control interaction in new or existing projects – this is a separate step.

New projects

Whether or not you immediately enable version control for a new project depends on your project development plan. If you plan to spend quite some time brainstorming, modeling, and designing, and if you do not need to preserve any artifacts of this process in version control, you can wait until you are ready to begin “real” work on the project to enable version control. On the other hand, if your design concept is already formulated, or if you are creating a new project for an existing code base, you should enable version control for the new project as you create it.

To enable version control in a new project:

1. Make sure your version control system is already installed and running and that it is configured in the Options dialog as described earlier.
2. Choose **File | New** on the main menu.
3. In the Object Gallery, choose **New Project**.
4. Specify project name, location, language, and other options, and click **Next**.
5. On the Resources page, check the *Version control project path* checkbox.
6. Click the **Options** button to the right, and specify the VCS settings as described above.
7. Use the browse button to the right of the field below the checkbox to specify which VCS project or repository in your version control system the Together project should access. The Select Project dialog opens. Choose the folder or repository that is relevant for the project you are creating.

For more information on creating projects, see Chapter 4, “Working with Projects”.

NOTE: If you open a project associated with CVS that lacks CVS administration files, Together tries to determine whether the associated CVS project exists in the repository. If it does not find the project, Together displays the following message:

This appears to be the first time you have opened this CVS project. Do you want to create it in the repository? (Yes/No)

On selecting Yes, the project is created in the repository using the import command. Otherwise, the version control session is terminated.

Existing projects

To enable version control in an existing project:

1. Make sure your version control system is already installed and running and that it is configured in the Options dialog as described earlier.
2. Choose **Project | Project Properties** on the main menu, to open the Project Properties dialog.
3. Click the **Advanced** button to display the Resources pane.
4. Check the *Version Control Project* checkbox.
5. Use the browse button to the right of the field below the checkbox to specify which VCS project or repository in your version control system the Together project should access. The Select Project dialog opens. Choose the folder or repository that is relevant for the project you are creating.

Together files to include in version control

To protect your visual modeling information, place diagram and project files under source control, in addition to your source code files. The following table describes the types of files you should add to source control:

Table 37 Together files under version control

Files	Description	Location
*.tpr	Together project file	Project primary root folder
.df	UML diagram files. .dfPackage is the default package view diagram generated by Together. All other diagrams are user-created.	There will be one .dfPackage file in every folder in the project that has been parsed. Other diagram files are wherever users have created them.
*.tws	User-specific desktop settings. Probably not a source-control item, especially in multi-user environments, but you can back it up in VCS if you want to.	Project primary root folder

Interacting with version control

After you set Version Control options and set up version control in project properties, you can use Together to interact with your system. You can add, get, check in, check out, and so on.

To use version control in your project:

1. Make sure that version control is enabled in the Options and Project Properties.
2. Select the object under version control using one of the following methods:
 - Choose a source-generating diagram element in a diagram (such as a class or interface).
 - Choose a source-generating element in the Explorer.
 - Focus the Editor with a class source file loaded.
3. Choose **Version Control** from the right-click menu. This displays a submenu of the commands available for the currently configured VCS.
4. Choose the desired action from the submenu.

You are now presented with a dialog that enables you to complete the chosen action. The layout and content of the dialog varies depending on the action selected and the version control system you are using. If you are experienced with your VCS, the dialog should be fairly self-explanatory.

Alternatively, you can choose the **System** command on the right-click menu to open the System dialog. This dialog is an integrated tool that enables all version control actions.

For information about the various dialogs for different systems, see Version Control dialogs in the online help.

Examining differences

Once you have selected the type of version control to use, set up the desired tool to view differences. Together provides a built-in difference viewer, which is used by default. For CVS, however, checking the option *Use external Diff tool* enables you to invoke any other external tool, whose fully qualified path is specified in the field *Name of External Diff executable*.

External visual diff tools

To configure Together to work with another visual difference viewer installed on your computer:

1. Choose **Tools | Options | <level>** on the main menu, and choose the *Version Control* node in the Options dialog.
2. With version control enabled, choose the CVS node.
3. Check the *Use External Diff tools* box.
4. In the field *Name of External Diff executable*, specify the fully qualified path to your favorite tool (for example, Araxis Merge).

The currently assigned external tool is invoked from the System dialog.

NOTE: The `examdiff` tool that comes bundled with Together can be used in the Windows environment only.

Product-specific VCS notes

Together has been tuned and tested with a number of providers with the SCC interface. These are: PVCS (VM and Dimensions), StarTeam, Perforce, and Continuous version control systems. Other versioning systems may be used with Together, but only those mentioned here are currently tested.

PVCS command line tools

PVCS command line tools are no longer supported. SCC is now used instead. If PVCS VM 6.5 is used as an SCC interface, PVCS command line tools works through it. However, it is recommended to use PVCS VM 6.6

PVCS Dimensions

SCC integration with PVCS Dimensions version 6.0 is supported. The product is fixed for JDK 1.3.

Before starting to use Dimensions with Together, Dimensions must be configured to support Together as an IDE. For details, see “Configuring PVCS Dimensions to work with Together's IDE” on page 315.

The directory path set in the default Work Set must be the parent directory of the Together project root associated with the Dimensions project.

The initial name of the Dimensions project, as entered in the *Version Control Project* field of the Project Properties dialog, must be unique among all directories in the Work Set.

Problems and workarounds

- When running Version Control commands from Together, Dimensions might display a modal window with command execution status, which is inactive and cannot be closed. To avoid this problem, set the property in the `vcs.config` file to `vcs.scc.usecallback=true` (false is the default). This causes Dimensions to display all status messages in the Together message window.
- After adding or checking in a file to Dimensions, independent of the state of the *Keep Checked Out* checkbox, Dimensions deletes the files from the local directory. To get these files back, execute *Get* or *Checkout* on the directory where the files were located.
- On first running a Get or Checkout command from Together with PVCS Dimensions, it may crash unexpectedly without any warning; the process terminates, killing the Java machine. To avoid this problem, set the property in the `vcs.config` file to `vcs.scc.queryinfobeforeget=true`. This option tells VCS to query info on files before proceeding with Get.

Configuring PVCS Dimensions to work with Together's IDE

Together initializes the `sccpcms.dll` with the IDE name *Together*, so the IDE environment Together must be set up correctly.

To configure PVCS Dimensions:

- In \$GENERIC product, add an object type PROJECT with attribute IDE_VALIDSET.
- Define Valid Set IDE_PROJECTS, which by default contains definitions for IDEs certified to work with Dimensions, created by IDE Setup.
- Add a value Together, tg.
- Attach lifestyle SOURCE to the object type.
- Define file formats and MIME types for file types you want Together to upload to Dimensions, if not yet defined. These types are:

Table 38 File formats and MIME types

Name	Format	MIME type
Java	Ascii text	text/plain
text/plain	Ascii text	text/plain
C++	Ascii text	text/plain
PASCAL	Ascii text	text/plain
IDL	Ascii text	text/plain
PROJECT	Ascii text	text/plain
DIAGRAM	Ascii text	text/plain
WMF	Binary	binary/wmf
GIF	Binary	binary/wmf

- In the IDE Setup tool, define item types used by Together. These items are:

Table 39 Together item types

Pattern	Format	Type
%.java	JAVA	SRC
%.cc	C++	SRC
%.hpp	C++	SRC
%.h	C++	SRC
%.idl	IDL	SRC
%.pas	PASCAL	SRC
%.dfActivity	DIAGRAM	SRC
%.dfBusinessProcess	DIAGRAM	SRC
%.dfClass	DIAGRAM	SRC
%.dfComponent	DIAGRAM	SRC
%.dfDeployment	DIAGRAM	SRC
%.dfEJBAssembly	DIAGRAM	SRC
%.dfER	DIAGRAM	SRC
%.dfPackage	DIAGRAM	SRC
%.dfSequence	DIAGRAM	SRC
%.dfUseCase	DIAGRAM	SRC
%.dfXMLType	DIAGRAM	SRC
%.tpr	PROJECT	SRC
%.twc	PROJECT	SRC
%.wmf	WMF	DAT
%.gif	GIF	DAT

Continuus/CM

Together supports Continuus/CM with the SCC interface. The SCC integration with Continuus version 5.0 was tested with SCCI generic integration.

Known issues

- The CASE property of the database must be set to PRESERVE and the FILENAMESLIMIT must be set to NONE in order to prevent renaming of files that are created by Together. If this is not done, numerous parser errors will appear in the message window.
- The working-area path specified for this project must be a Together project root that is associated with the Continuus project. Refer to the Continuus manual for details on how to set up a working area. The name of the project to be associated with the project root is returned from the Select Project dialog, accessed from the Select button in the project's Properties dialog.

NOTE: A Continuous project must already exist and its working-area path must be set to the same value as the Together project root path before associating the root with Version Control.

- Continuous is very sensitive to the Work Area Templates. The directory template should be set to the path of the parent directory of the project root to be associated with VCS. The project subdirectory should be set to an empty string.
- There may be a crash on the second attempt to add a file if the first attempt failed with the message `SCC_E_NONSPECIFICERROR`. This can be due to the lack of necessary privileges. To prevent this, before taking any action with files in Together, ensure that the Continuous GUI is running under a login/role that allows file-adding privileges.

Perforce

Together supports Perforce with the SCC interface. The SCC integration with Perforce was tested with client version 99.1.10232, and server version 99.1/11533 (1999/07/29).

Known issues

- A created Perforce project maintains the same name as the Together project directory, even if that is modified through the Together project properties. Perforce cannot create a subproject with an arbitrary name. All Perforce project names are the same as the local directory names where the corresponding files are stored.
- Perforce does not support Remove and Get commands. Although these appear to be available from the Version Control System dialog, attempting to use them does nothing.

Problems and workarounds

- The default Perforce project name, produced by Together, starts with a `\\depot` prefix. The Browse VCS Project button does not work with this provider.

In order to work around this limitation, locate all Together projects that are to be used with version control under the client root directory, which can be modified in Perforce Client. Refer to the Perforce documentation for details about this process.

PVCS Version Manager

Together supports the PVCS Version Manager with the SCC interface. The SCC integration with the PVCS Version Manager was tested with version 6.6 (build 775).

Known issues

- The default PVCS project name produced by Together, is incorrect. For the PVCS VM SCC interface, this name should contain the path to the PVCS Project Database. To resolve this when starting work with this provider, use the Browse VCS Project button in the Together project properties for every project that has not yet been under version control. In the Select Project dialog provided by PVCS VM, select any

existing project or create a new one with a default name the same as that of the current Together project name.

- The VCS project name specified in the Together project properties should exist. However, if you create a correct PVC project name and place it manually into the Together Project Properties' Version Control Project field, the corresponding project will not exist in the PVCS Project Database and an error message is shown in the message window.

StarTeam

The SCC integration with StarTeam was tested with version 4.1 (build 4.1.537).

Known issues

- Since Together uses the SCC interface for access to the StarTeam database, the interface should be installed during the StarTeam installation process. To do so, choose the Custom StarTeam setup type and install one of the following components:
 - Developer Studio Integration
 - VB and Developer Studio Integration
 - Sybase PowerBuilder Integration
- Because only StarTeam VirtualTeam servers, which use the TCP/IP (Sockets) protocol, can be used in the SCC interface provided by StarTeam, you must configure your StarTeam client to use this protocol before working with Together.
- Problems can occur if you add files from a package whose parent does not exist in the StarTeam project. We recommend adding files to the version control project package by package. This means ensuring that the package's parent is already created in the Version Control Database before adding any of its files. You should invoke any VCS command for each new package from the project. For example, if you create a new physical package, invoke the System dialog from the default diagram of this package.
- Because StarTeam does not add read-only attributes to its work files, Together project elements remain writable after Add or Check In commands.

Using Together with ClearCase

ClearCase is one of the many options for version control systems (VCS) within Together ControlCenter™. ClearCase maintains the work of many developers on one or more servers which developers may access using the ClearCase client.

There are two ways to access a ClearCase client from Together ControlCenter:

- Utilize the SCC (Source Code Control) interface, a generalized interface on Windows which provides access to many of the commercially available version control systems. A drawback of this interface is that it is a least common denominator approach since it presents a general interface to many different version control systems. It is also available only for clients on Windows.

- Use the native ClearCase interface. This interface has a command line interface to deliver functionality unavailable through the SCC interface. The native ClearCase interface also works on platforms other than Windows where Together ControlCenter and ClearCase both operate. A description of this interface is in this section.

Setting up the interface

The first step to creating the link between Together ControlCenter and ClearCase is to create a Together project. This project must be created in a snapshot or dynamic view under a versioned object base (VOB). You cannot check in and check out artifacts in ClearCase unless they are in a view associated with a VOB.

There are two types of ClearCase projects: those that use Base ClearCase and those that use Unified Change Management (UCM). Your ClearCase administrator knows which type of project you are working on based upon the way that VOB is set up. Understanding the type of project being created is important to configuring the options in Together.

Large projects can require artifacts to be placed in multiple VOBs. Multiple VOBs can be accessed simultaneously by placing their view in Project Paths under the Advanced Options dialog. You can also use Advanced Options to configure the interface between Together and ClearCase.

ClearCase options

See the instructions in “Configuring Together for version control” on page 306 for accessing Version Control configuration options from the Project Properties.

To set up options in the Version Control Options dialog box:

1. Select **Version Control** from the tree view on the left pane to set the overall options.
2. In the *Use* field, select **ClearCase** from the dropdown box.
3. Click **ClearCase** in the tree view in the left pane and set the ClearCase options as described below.
4. Click the **Ok** button to return to the project properties dialog.
5. Check the box to the left of *Version Control project* on. The directory name in which the project displays in the textbox below the label *Version Control project*.
 - If the name is displayed and the checkbox remains checked, configuration is successful.
 - If the checkbox remains unchecked, the Message pane displays the error messages on a *Version Control* tab.

There are seven configuration options for ClearCase.

- **Mode:** Base ClearCase or Unified Change Management (UCM). If you select UCM from the dropdown menu, you must be working on a project in a UCM view and a project VOB (PVOB). The UCM mode provides all of the functionality of the Base mode in addition to activity management.

- **Set development view on start:** Available only on platforms such as Unix. On these platforms, you typically perform a setview when you begin to work in ClearCase. When the checkbox is checked, a setview will be performed when your project starts.
- **Development view:** The name of the set view (using setview) on these platforms. The name must be a valid view name or an error will be returned. The two development view options have no effect on Windows platforms (or any other platform in which setview is not supported).
- **Check if ClearCase directory:** Confirms if the directory of the opened project is a ClearCase directory or if the ClearCase version control system is enabled. Leaving this option checked off results in better performance on startup but no initial license check will be attempted.
- **Show version ids:** Turns on the version id capability in the System pane (see Figure 86). Querying for version ids can be performance intensive. The default is to turn version ids off.
- **Default checkout to reserved:** Reserves all checkouts unless overridden in the advanced options on each checkout. However, you may wish the default to be unreserved. When the box is not checked, all checkouts default to unreserved. You may use the advanced options to reverse this default on the individual checkout.
- **Use version tree for history:** The version tree view is used instead of the event log view.

In addition to the configuration options, the following are two comments:

- **Use default comment:** When this option is set, the default comment is used for implicit commands such as class and package renaming. Additionally, the default comment is set as the default comment on checkins and checkouts.
- **Default comment:** Used when the *Use default comment* box is checked for implicit ClearCase commands.

There are two UCM configuration options:

- **Prompt when changing activities:** turns on and off the prompting mechanism when changing activities (described later).
- **Allow new activity creation:** toggles the ability to create new activities within Together ControlCenter. This option is often found unchecked when ClearQuest is used to create and track new activities.

Basic Version Control

Since Together ControlCenter provides a framework for version control, most of the basic operations (such as add, checkout, and so on) behave the same way in Together ControlCenter. The dialogs that perform these operations look the same among different version control systems. The results of using the dialogs, however, may be very different. For example, a checkout in CVS does something very different from a checkout in ClearCase.

This section discusses the semantics of the various operations and the advanced options for ClearCase.

Add

Files and directories must be made ClearCase elements before they can be put under version control. The *add* command makes a file or a directory into a ClearCase element. Once they are ClearCase elements, files and directories may then be checked in and out of ClearCase. After an *add*, the file is checked out to the view in which it was added.

Update

There are two types of views in ClearCase: snapshot and dynamic. In a snapshot view, elements (files and directories) are written to a local disk. Periodically, the view must be updated to get the latest versions of the elements according to the rules in the view. Perform an update to incorporate the changes of others or when you are ready to integrate your work.

Check-in

When they are checked in, elements are locked and read only. Therefore, in order to change an element, you must check it out. (Adding a new file or directory to a directory is considered a change.) You must provide a comment when checking in an element. In addition, when in UCM mode, you must specify an activity (see “ClearCase options” on page 319) under which the check-in is made.

You can perform a diff of the element during check-in. The diff compares the current version against other versions of the element.

Uncheckout

An alternative to checking in an element which has been checked out is to *uncheckout* the element. The artifact reverts to the last checked in version. Any changes that you have made are saved to a .keep file. This command is used when you have decided to discard changes that you have made for some reason.

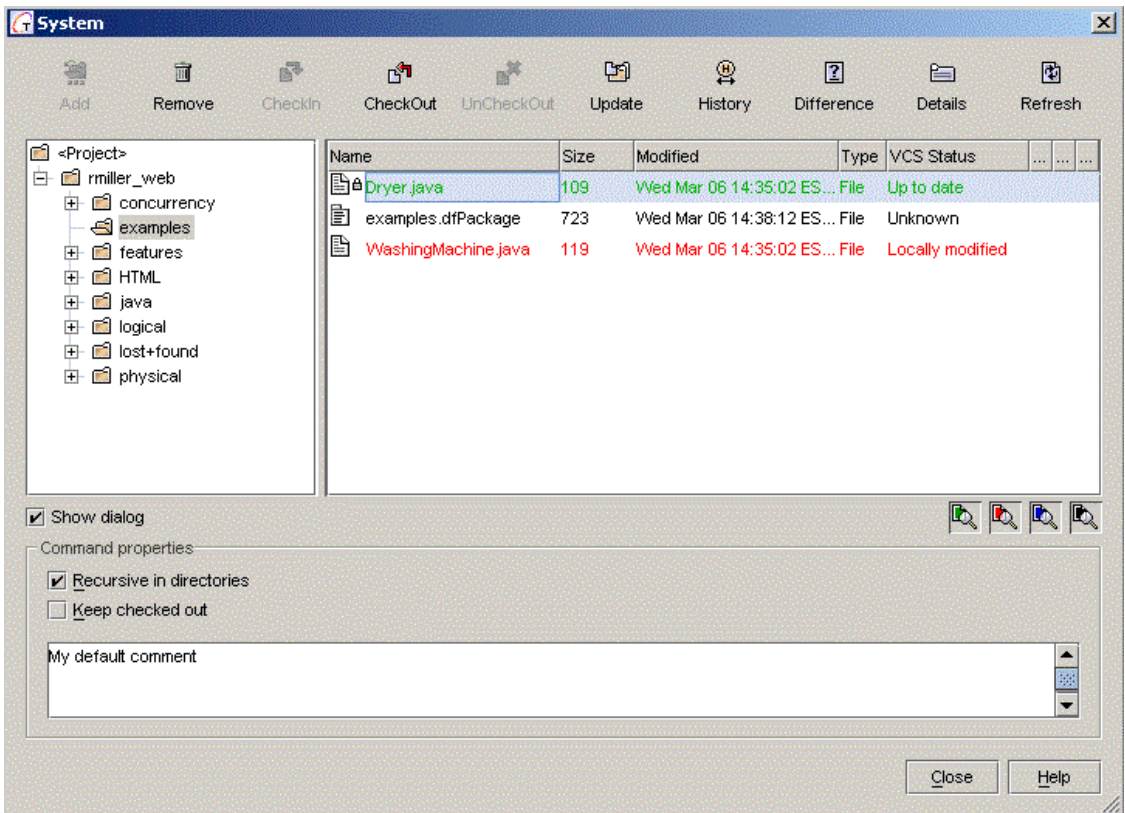
Checkout

When you wish to make changes to an element, you must check it out. Advanced options allow the checkout to be reserved (exclusive) or unreserved. Reserved checkouts lock others from being able to make changes to the element until it is checked in. Reserved checkouts are the default in ClearCase and the default in Together ControlCenter.

System

The system dialog brings up the status of the entire project. (See Figure 86.) Bringing up this view of the system takes some time since you are querying the status of the entire project. However, the results are cached for subsequent system checks. From the system panel, you can view the history of an element, which brings up the ClearCase history view. You can also perform many of the basic operations (add, checkout, etc.) from this view.

Figure 86 The system pane



Advanced Version Control

Advanced version control features are typically used less frequently than their basic counterparts. As a result, they are available only from the system dialog in Together ControlCenter 5.5. None the less, they are equally important to the interactions between Together ControlCenter and ClearCase.

History

The history browser option displays a ClearCase History Browser. From the history browser, you can view changes or the version tree. The version tree provides the ability to merge changes from different branches or streams.

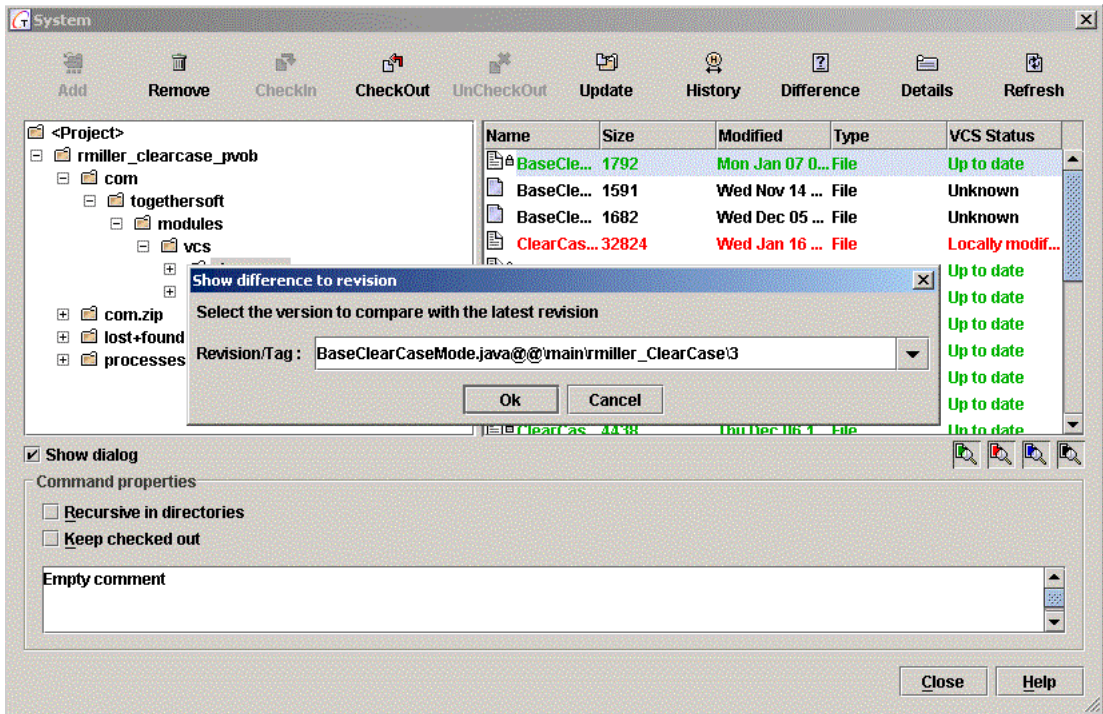
Remove

Remove acts much the same as it does in other version control systems. The element is removed from the directory version.

Difference

The difference option displays the ClearCase Difference Browser. A dialog prompts the user for a version to compare to. The current version of the file is compared with the version chosen, as shown in Figure 87.

Figure 87 Show difference to revision dialog



Details

Details of the currently viewed version of the file are displayed. Information such as the last modified date, status, and version id are included.

Refresh

The Refresh option recalculates the status of the files in the status window. If the status of a file is not correct, refresh will clear the cache of files statuses and rebuild the cache.

Reporting Results and Errors

Operations in Together ControlCenter may result in actions being performed in ClearCase. When these actions are performed, ClearCase usually returns messages indicating what action was taken and the results of the action. This information is

reported on the message pane on the Version Control tab. Any status messages or errors which occur during an action will also be reported.

The majority of the information displayed in this window comes directly from ClearCase. Occasionally, there will be a Together ControlCenter message pertaining to an action that could not be taken.

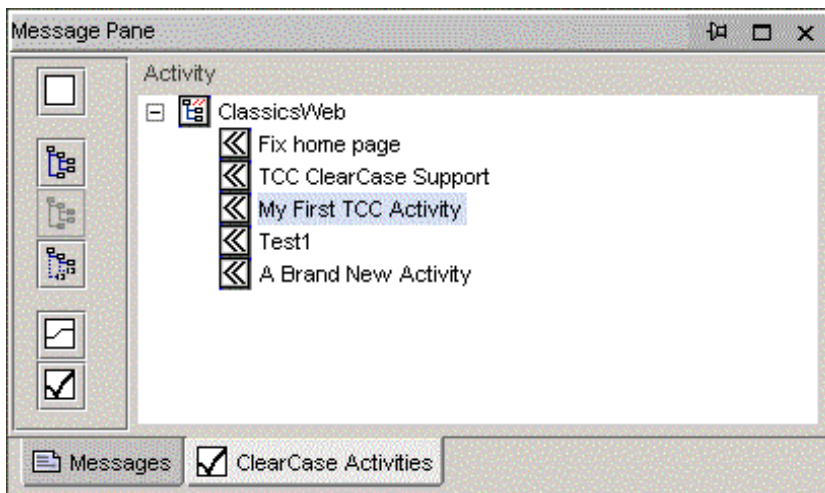
Unified Change Management

Together supports the creation and management of activities according to the principles of Unified Change Management. Under Together ControlCenter, you can:

- display all of the activities for a given project
- display activities which have no change sets
- refresh the list of activities
- create a new activity
- rebase a view
- deliver an activity

Activity management occurs in a Message pane tab entitled *ClearCase Activities*. (See Figure 88.)

Figure 88 Activity management in Together ControlCenter



The activity panel contains a toolbar of six buttons on the left hand side and a tree of activities to the right. The root of the tree is the project. Underneath the project is its activities. The current activity is highlighted in the tree.

You can change the current activity by selecting another in the tree. If the configuration option dictates (see “Setting up the interface” on page 319), the program asks if the

current activity should be changed. Otherwise, the newly selected activity will be highlighted and set as the current.

Check outs and checkins are made against activities in UCM. This allows changes to elements to be tracked. When an element is checked in against an activity, it is added to the changeset for that activity. Activities that contain changesets display with a box containing << (as shown in Figure 88). Activities with no changes checked in against them display with an empty box.

The first button on the toolbar creates new activities. This button is disabled if configuration dictates that new activities cannot be created in ControlCenter (see “Setting up the interface” on page 319). A dialog prompts for all of the information necessary to create a new activity.

The second and third buttons toggle the view of activities in the tree. If the second button is active, all of the activities are in the view. If the third button is active, only the activities with no changesets are visible. This view offers an idea of the tasks that need to be completed.

The forth button refreshes the activity tree. This refresh may be necessary if activities were created via ClearQuest and must be imported into Together. The refresh button updates the tree with the latest activities and their status.

The fifth button brings up the graphical interface for rebasing a view. Since this interface is asynchronous, you may return to ControlCenter during the rebase and compile and test against the new baseline. You can complete or cancel the rebase depending on the results of your tests.

The sixth button is for delivering changes to the integration stream. Deliver also utilizes a ClearCase graphical interface. ClearCase performs the necessary merge operations involved with such a delivery. Be sure to complete delivery (press the Complete button in the ClearCase window) so that all of the changes to the integration stream are posted.

PART
5

GENERATING DOCUMENTATION

- Chapter 17, “Generating Documentation for Together Projects”
- Chapter 18, “Designing Custom Documentation Templates”
- Chapter 19, “Documentation Template Controls”
- Chapter 20, “Creating Multi-Frame HTML Documentation”
- Chapter 21, “Documentation Generator and Template Designer Reference”

CHAPTER 17

GENERATING DOCUMENTATION FOR TOGETHER PROJECTS

Together ControlCenter can create external documentation for Together projects. This chapter describes how to use Together's documentation generator to create project reports from within Together and from the command line.

The contents of this chapter are:

- “Generating documentation for an open project” on page 327.
- “Automating documentation generation” on page 330.

Generating documentation for an open project

You can start the documentation generator from an open Together project. Simply select one of the following three commands from Together's main menu:

- 1. Project | Documentation | Generate HTML:** generates HTML documentation using a documentation template that Together provides.
- 2. File | Print Documentation:** generates PDF documentation or prints directly to the printer using a documentation template that Together provides.
- 3. Project | Documentation | Generate Using Template:** generates HTML, RTF, or text documentation using a custom documentation template or a template that ships with Together.

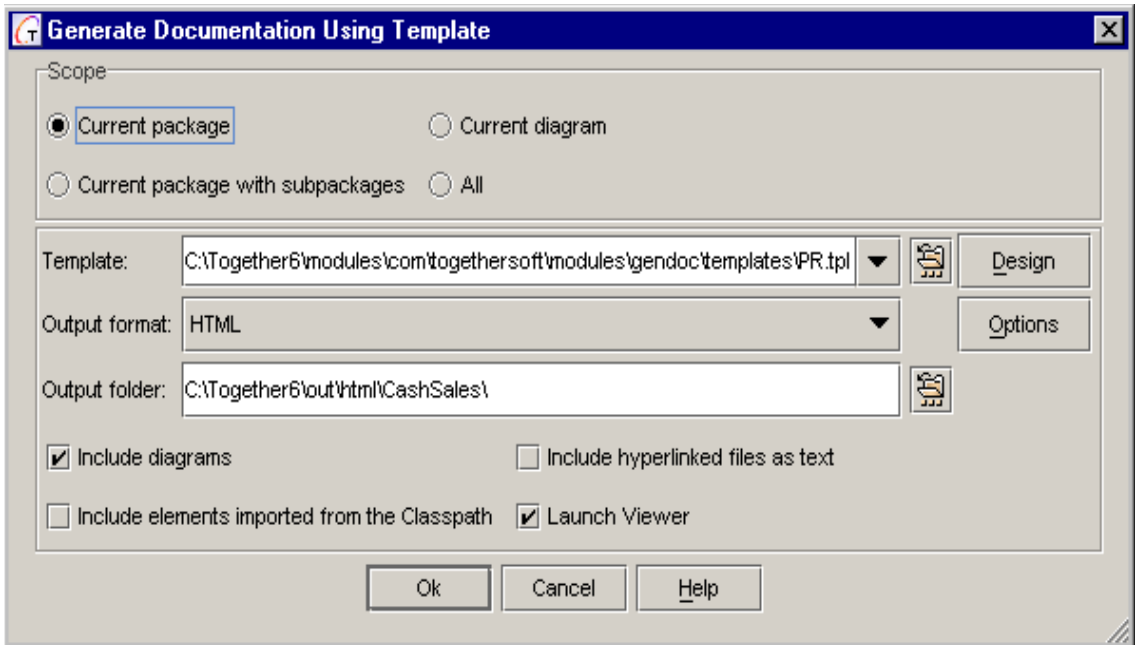
Figure 89 shows the Document Generation Using Template dialog box resulting from the last command.

Documentation generation parameters

The dialog box for generating documentation from a template enables you to fill in the parameters that determine the resulting report. The **Scope** section at the top of the dialog box has radio buttons to indicate what parts of the project should be parsed and included in the generated output.

- **Current package:** Generated output includes only the current package selected in the Model tab of the Explorer.
- **Current package with subpackages:** Generated output includes the current package selected in the Model tab of the Explorer and any subpackages under it.
- **Current diagram:** Generated output for the current diagram that is in focus in the Designer pane.
- **All:** Generated output covers the entire project.

Figure 89 Document Generation dialog box



The lower section of the dialog box has options to specify the type of template on which the documentation is to be generated, output format and destination, and optional actions.

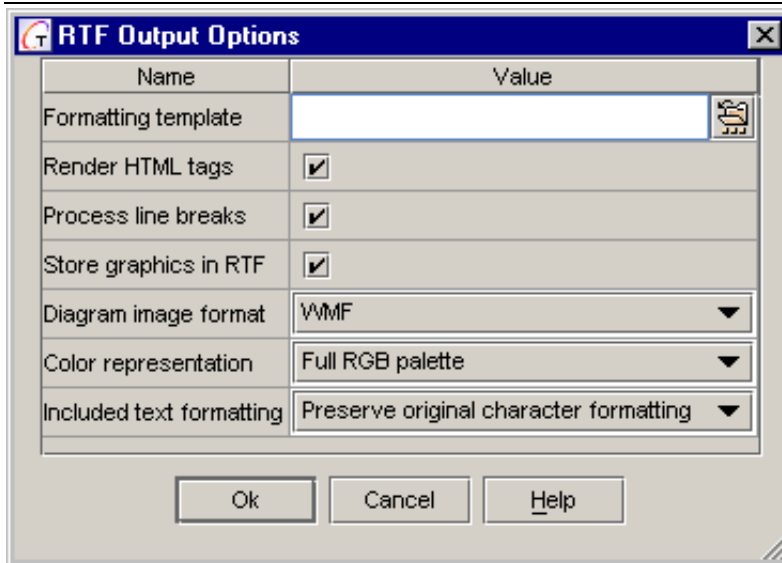
- **Template:** Specify the template for the documentation generator to use. You can select from the history dropdown list or press the file chooser button to select one of the available templates. By default, the file chooser navigates to the templates folder of the gendoc module.

- **Design:** Click to create your own template or modify an existing one. (Extensive information on how to design custom templates starts on Chapter 18, “Designing Custom Documentation Templates.”)
- **Output format:** Select HTML, RTF, or PDF from the dropdown list.
- **Output folder:** Enter the location for the generated output, or select from the file chooser. The default location is a subfolder of \$TGH\$/out.
- **Include elements imported from the classpath:** Check to include these elements in the generated output.
- **Launch viewer:** Check to view the documentation in the appropriate application as soon as the documentation is generated. This starts the application if necessary.

RTF documentation options

With **RTF** as the output format, you can click the **Options** button to refine the parameters for generating the documentation. Figure 90 shows the resulting RTF Output Options dialog box.

Figure 90 RTF Output format options



- **Formatting template:** The fully-qualified name of a formatting template (such as a Microsoft Word *.dot file), or pick a template with the file chooser button.
- **Render HTML tags:** Translate HTML tags into appropriately formatted text in the printed documentation. The following tags are now supported:
`<b>`, `<i>`, `<u>`, `<h1>`, ..., `<h6>`, `<code>`, `<tt>`, `<em>`, `<font color>`, `<pre>`,
`<p>`, `<br>`, `<ol>`, `<ul>`, `<li>`

NOTE: Descriptions for model elements may contain HTML tags. This setting is useful for documentation templates that process descriptions.

- **Process line breaks:** Preserves line breaks even if the Render HTML tags is checked on.
- **Store graphics in RTF:** Embeds all the graphics in a single RTF document.

NOTE: RTF supports PNG rather than GIF image format. Storing GIF images may cause problems with some RTF viewers. Since Microsoft Word 2000 handles GIF images by translating them to PNG images, you can save the generated documentation in Word in order to make that translation. Be aware, however, that you may have to make minor changes to the file (such as remove a space) in order to format the document correctly.

- **Diagram image format:** WMF or GIF format.
- **Color representation:** RGB color or 16-bit color.
- **Included text formatting:** Preserve the original formatting or apply the one from the Formatting template (specified in the first option).

Automating documentation generation

You can update project documentation as part of periodic automated build process by having the process script call Together's documentation generator through the command line interface. Together ships with three launchers:

- `$TGH$/bin/Together.exe`
- `$TGH$/bin/TogetherCon.exe`
- `$TGH$/bin/win32/umldoc.exe` (for Windows users only).

Together.exe and TogetherCon.exe produces RTF and HTML documentation; umldoc.exe produces HTML documentation only. Both TogetherCon.exe and umldoc.exe bypass the Together window, while Together.exe starts the shell and takes you through the entire documentation generation procedure.

Launching the documentation generator from the command line

Use the following syntax to launch the documentation generator from the command line:

```
<starter> -script:<command> <project>
```

where:

- `<starter>` is
 - Win32
 - `$TGH$/bin/Together.bat` or `$TGH$/bin/TogetherCon.exe` or `$TGH$/bin/Together.exe -con`
 - Other OS
 - `$TGH$/bin/Together.sh`
- `<command>` is `com.togethersoftware.modules.gendoc.GenerateDocumentation`

- <project> is the fully qualified project name, for example: \$TGH\$/samples/java/CashSales/CashSales.tpr

NOTE: This does not presently run fully in console mode; the Generate Documentation Using Template dialog box displays for setting the generation options.

Generating HTML and RTF documentation from the command line

Use this command to launch Together's documentation generation module from the command line:

```
<launcher> [module] [options] <project>
```

where:

- <launcher> is Together.exe, TogetherCon.exe, or umldoc.exe.
- [module] is a command for the specific utility:
 - script=com.togethersoft.modules.genhtml.GenerateHTML for generating HTML documentation;
 - script=com.togethersoft.modules.gendoc.GenerateDocumentation for generating RTF documentation;
- [options] are those for GenerateDocumentation or for the command-line GenerateHTML module launcher (\$TGH\$/bin/win32/umldoc.exe). They are listed in Table 40 and Table 41.
- <project> is the path to the project.

NOTE: This parameter is not required for umldoc.exe since it executes the GenerateHTML module only.

Documentation generation options

Automatic HTML and RTF documentation generation have command-line options.

RTF options

Each RTF documentation option is either a "switch option" or a "parameter option." Switch options have the form:

-option_name

Parameter options are followed by parameter values:

-option_name parameter_value

There are regular RTF options and custom options. The regular options are listed in Table 40.

Table 40 Regular RTF options

Option	Description
-template <path>	Template file. if omitted, the default “ProjectReport” template is used: ..modules\gendoc\templates\ProjectReport.tpl
-metamodel <path>	MetaModel file; if omitted, the default meta-model file is used: ..modules\gendoc\templates\MetaModel.mm
-format <RTF HTML TXT>	Output format; RTF is the default.
-styletemplate <path>	Style template file (depends on the output format such as *.dot file for RTF output)
-d <directory>	Output destination directory
-f <path>	Output file path, compatible with TXT output format only. This option redirects all output to the specified file.
-diagrams	Include diagram charts.
-recurse	Create output for packages specified in [packagenames] and their subpackages.

Custom options are those not recognized as regular options. There is no fixed list of custom options, and custom options are not processed directly by the Documentation Generator. Each custom option must have a parameter.

Custom options are stored with templates. You can access the name of any custom option within a template using this function call:

```
getDGOption(String optionName)
```

This feature allows you to pass parameters to a template from command line and adjust the template behavior dynamically.

HTML documentation options

The options in Table 41 are valid for the `uml.exe` command.

Table 41 Options for `umldoc.exe`

Option	Description
-overview <file>	Read overview documentation from HTML file.
-public	Show only public classes and members.
-protected	Show protected and public classes and members (default).
-package	Show package/protected/public classes and members.
-private	Show all classes and members.
-help	Display command line options.
-sourcepath <pathlist>	Specify where to find source files.
-classpath <pathlist>	Specify where to find user class files.

Table 41 Options for umldoc.exe (continued)

-d <directory>	Destination directory for output files. Note: Not required for Together .exe, since this initiates the Together documentation generation dialog.
-use	Create class and package usage pages.
-version	Include @version paragraphs.
-author	Include @author paragraphs.
-splitindex	Split index into one file per letter.
-windowtitle <text>	Browser window title for the documentation.
-doctitle <html-code>	Include title for the package index (first) page.
-header <html-code>	Include header text for each page.
-footer <html-code>	Include footer text for each page.
-bottom <html-code>	Include bottom text for each page.
-nodeprecated	Do not include @deprecated information
-nodeprecatedlist	Do not generate deprecated list.
-notree	Do not generate class hierarchy.
-noindex	Do not generate index.
-nohelp	Do not generate help link.
-nonavbar	Do not generate navigation bar.
-stylesheetfile <path>	Cascading stylesheet file to apply to generated documentation.
-togethersettings	Use settings from config/GenerateHTML.config file.
-recurse	Create output for packages specified in [package names] and their subpackages.
-javadoc	Create the same output as javadoc .exe produces.
-browser	Launch HTML browser.
-diagrams	Create diagrams' pictures.
-navtree	Generate navigation tree.
-nopackagedependencies	Do not show package dependencies in all class diagrams.

CHAPTER 18

DESIGNING CUSTOM DOCUMENTATION TEMPLATES

The *Documentation Generator*, or *DocGen*, is the Together module that produces project reports. DocGen creates project reports by applying *documentation templates* to Together projects. The templates contain commands to the DocGen engine; the projects provide the source of project-specific data. Documentation templates are *.tpl text files with formatting instructions and tags for the commands.

The *Documentation Template Designer* is a tool in Together for creating custom documentation templates. This chapter shows how to design documentation templates. The remainder of this part of the User Guide cover the details of documentation template creation.

The contents of this chapter are:

- “Template Organization” on page 334.
- “Metamodel types” on page 336.
- “Creating new templates” on page 338.
- “Designing the template body” on page 340.
- “Accessing model elements through iterator sections” on page 342.
- “Reusing templates and stock sections” on page 345.

Template Organization

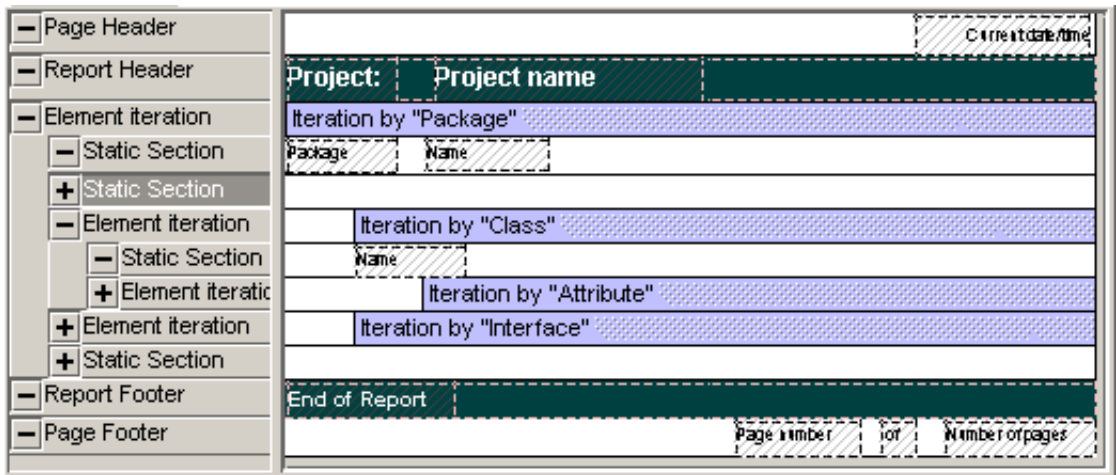
To open the Designer, select **Project|Documentation|Design Templates** from Together’s main menu. Alternatively, click the **Design** button when generating documentation from a template.

The Documentation Template Designer divides templates into five major zones:

1. Page Header
2. Report Header
3. Body
4. Report Footer
5. Page Footer

The zones are horizontal bands that go across two panes. The scope pane, which is on the left, reveals the template structure. The details pane on the right shows the contents of the zones, which include commands to the DocGen engine. (See Figure 91.)

Figure 91 Documentation Template Designer



Headers and footers are at the top and the bottom of the Designer window. The report header and the report footer apply only once per document. Page headers and footers apply once per page for RTF and PDF documentation; they are ignored for HTML documentation.

The body zone of a template contains the commands that produce the body of the generated report. DocGen builds a report into horizontal regions. Each region in the report corresponds to a section in the template that determines the data for that region and how that data should appear.

Body sections and the DocGen engine

The body of a documentation template is organized into a hierarchy of *sections*. The body of the template in Figure 91 starts with the first “Element iteration” section and ends with the last “Static Section.”

Some sections in the body are nested inside others. Some sections have siblings. Thus the sections form a tree whose root is the entire body zone. Sections that are not nested within any others are children of the root. The root of the tree in Figure 91 contains exactly one child (the first “Element iteration” section). The scope pane reveals the tree structure, indenting each section according to its level in the tree.

Every section in the body has a *section scope*. Scopes are based on *metatypes* that correspond to the different types of model elements. The section scope of the body zone corresponds to the *root object metatype*.

NOTE: Model elements can be thought of as directories, files, any visible on-screen widgets, as well as their properties that show up in the property inspector. This includes diagrams, actors, packages, classes, attributes, swimlanes, and so on.

The model itself is considered to be a special metatype, which is the default root metatype when you create a new template.

The DocGen engine uses a dynamic *current model element* to go through a template and access specific project information. The type of the current element is the metatype for the section that the engine is currently processing. The value of the current element changes according to when the processing for the section takes place.

DocGen produces the body of a report starting from the root element, going in a “depth-first” fashion. In other words, DocGen starts its processing with the first root section, visiting it along with any of its nested subsections before continuing to the next root section. This pattern is recursive: visit the subtree rooted at a section before going to the next sibling section. For each sibling of a section, DocGen begins its processing with the same current element.

Body section types

There are six different types of body zone sections.

1. *Static sections* contain the commands to DocGen for getting project data. (You can find detailed information on commands in Chapter 19, “Documentation Template Controls.”)
2. *Element iterators* are for looping through model elements of a particular type, applying commands in its nested sections.
3. *Element property iterators* are for looping through the properties of model elements.
4. *Folders* group other sections.
5. *Calls to stock sections* reuse sections from the template’s internal collection of stock elements.
6. *Calls to template sections* use other templates.

Static sections and iterations sections provide most of the functionality required by most templates. Folders, calls to stock sections, and calls to templates are conveniences for organizing and reusing templates. Detailed information on creating body sections is covered later in “Creating new templates” on page 338.

Metamodel types

Documentation metamodels define the types that determine the section scopes. The metamodels are described in two model definition files, `MetaModel.mm` and

GUIBuilderModel.mdf. These are text files in the Together installation under the folder modules\com\togethersoft\modules\gendoc\templates.

The model definition files show how shape types correspond to model elements, the types of elements another element can contain, and the properties of each element type.

The beginning of each model definition file lists the properties that DocGen knows. These include *RWI* properties, which are defined in the read-write interface of Together's open API, DocGen properties, and others.

The remainder of each model definition file contains the metatype definitions. The major fields in the definitions are as follows:

- name: metatype name
- extends: parent metatype
- rwi_entity: the name of the related element in the RWI of Together's open API
- full_name: the name displayed in the Documentation Template Designer
- properties: a list of properties available for this type (DocGen properties, RWI properties, and others). Every metatype except MODEL has a \$shapeType property.
- excluded_properties: items listed among the properties that are not available for this type (used for inheritance)
- metatype_filter: filtering conditions that distinguish among elements with the same shape type
- contained_metatypes: child metatypes that can be directly obtained from any element of this metatype

The name field for each type is always present. The existence of the other fields varies with the type. DocGen uses shape types, metatype filter expressions, RWI entity types, and collections of child metatypes to determine the relationships among the model elements.

Below is the code from MetaModel.mm that describes the ELEMENT, DIAGRAM, and NODE metatypes.

```
#-----
```

```
<metatype>
  name=ELEMENT
  rwi_entity=element
  full_name="Element (generic)"
  properties = { $shapeType; $name; $doc; %annotation;
                 $hyperlink; url }
  excluded_properties = { $shapeType; $name }
  contained_metatypes = { HYPERLINK_LINK }
</metatype>
```

```
#-----
```

```
<metatype>
  name=DIAGRAM
  extends=ELEMENT
  rwi_entity=diagram
  full_name="Diagram (generic)"
```

```

    properties = { %package; stereotype; alias }
    contained_metatypes = { DIAGRAM_REFERENCE; DIAGRAM; NODE; LINK }
</metatype>

```

```

#-----

```

```

<metatype>
    name=NODE
    extends=ELEMENT
    rwi_entity=node
    full_name="Node (generic)"
    properties = { %package }
    contained_metatypes = { NODE; MEMBER; LINK }
</metatype>

```

```

#-----

```

An element iterator or folder can contain nested element iterators whose type is listed among its contained metatypes, the contained metatypes of its parent, or indirectly through the contained metatypes of one of its contained metatypes. For example, an element iterator with DIAGRAM scope can contain nested element iterators with the following scopes:

- hyperlink (inherited from ELEMENT)
- diagram reference, diagram, node, link
- member (indirectly through the contained metatype, NODE)

Element properties are inherited. An element iterator can contain nested property iterators whose type is inherited from its ancestor or listed directly among its properties. For example, an element iterator with DIAGRAM scope can contain nested property iterators for the following type scopes:

- shapetype, name, documentation, annotation, hyperlink, url (inherited from ELEMENT)
- package, stereotype, alias

Creating new templates

To create a new documentation template, select **File | New | New Template** from the main menu of the Documentation Template Designer. The New Template dialog box displays two model options:

- **RWI meta-model**: for reports based on the project model
- **GUI Builder**: for reports based on the GUI builder model

and two kinds of templates:

- **Document Template**: for RTF, PDF, or non-frame HTML documentation
- **Frameset Template**: for HTML frame-based documentation

All of the topics discussed in this chapter and the next (Chapter 19, “Documentation Template Controls”) are appropriate for document templates. Frameset templates for

multi-frame HTML are covered in Chapter 20, “Creating Multi-Frame HTML Documentation.”

The main menu of the Documentation Template Designer has four commands: File, Object, Stock, and Help.

File provides the usual file operations (creating new templates, opening or re-opening existing templates, saving changes, exiting). You should go here to access properties for template settings.

Object is identical to the right-click menu of the currently selected template “object” -- both display the available object operations. Template objects can be the bands that make up a template as well as the visible items in the bands. Different kinds of objects have different right-click menus, although all menus list **Properties** or **Area Properties**. The list of items on right-click menus for static sections, headers, and footers varies between the scope pane and the details pane. (There is no Object or right-click menu for headers and footers in the scope pane.)

Stock is for creating a new stock section or editing an existing one.

Setting template properties

You can access options for a template in the Documentation Template Designer by selecting **File | Properties** from the Designer’s main menu. Figure 92 shows how the **Template Properties** dialog box displays the template settings on four tabbed pages: General, Page Settings, Formatting Styles, and Template Parameters.

- **General** properties are for defining the report title, picking the metatype of the root element, and toggling headers and footers. At the top of the page is a field for describing the template.

The **Report Title Expression** is used for HTML documentation. DocGen evaluates this expression to put the title on the browser window.

DocGen starts processing a template using a root element from the model of the type given by the **Root Object Metatype**. If this template is called from a different template, then the calling template supplies the actual root. Otherwise, the root element is the model, the current diagram, or the current package depending on how the documentation generation is started. Since the current element metatype must be consistent with the metatype of the current section, the Root Object Metatype limits where the template can be called.

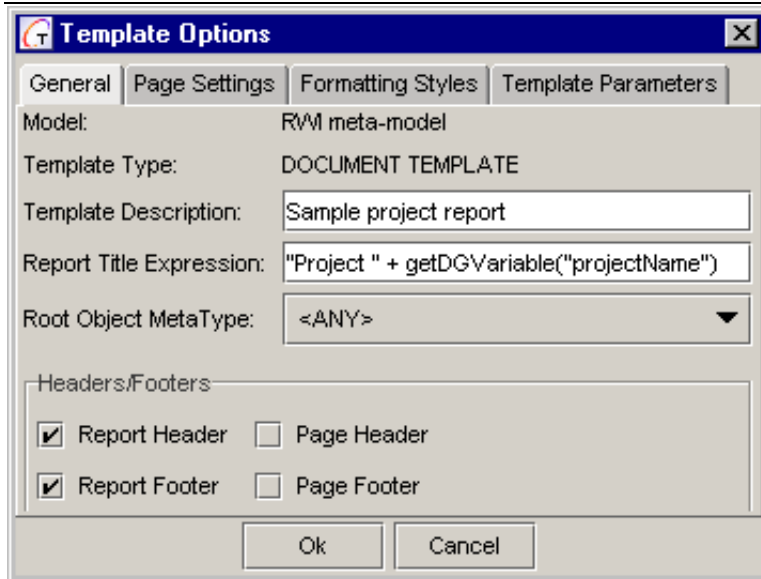
TIP: The default root metatype, <ANY>, is generic and consistent with all metatypes. You should use this default if you do not want to limit where the template can be used.

- **Page Settings** enables you to specify page size, margins, and landscape or portrait orientation.
- **Formatting Styles** are named text formatting styles in this template. When you create a new style by clicking the **New** button, a Style dialog window displays options for the style’s properties such font, size, color, border, and so on. (“Text format settings” on page 355 has more details on text formatting.)

On the Style dialog window, you can set the style name, apply the style to paragraphs or characters, and set this as the default style. The Levels list is for picking the outline level for RTF documents.

- **Template Parameters** are formal parameters used for calling this template from another template. There are three fields for each formal parameter: the parameter's name, the description, and a default value (a string). The name is required. The description and default values are optional.

Figure 92 Template Properties dialog box



Designing the template body

The body of a newly created template consists of a generic element iterator and a static section nested within. It provides a minimal base for constructing the tree of sections. Every new section must be a sibling or a child of an existing section.

Creating, arranging, and resizing sections

To create a new section, right-click on an existing section and select **Sibling Section** or **Nested Section**, then select the type of new section from the resulting menu. If the existing section is not static, you can right click anywhere in the section. For static sections, you must right click on the portion of the section that lies in the scope pane.

You cannot change the level of a section and you cannot move a section to a different parent. However, you can change the position of a section among its siblings. Right click the section and select **Move Up** or **Move Down** to reorder the section one position higher or lower.

To collapse or expand a section, click the “+” or “-” in the section’s upper left corner of the scope pane.

To resize a section, position the cursor over its lower edge in the details pane. After the cursor becomes a double-ended arrow, click and drag the lower edge to its desired position.

Setting section properties

After creating a section, you can change its properties by selecting **Properties** (or **Area Properties**) from its right-click menu or from **Object** on the main menu. The resulting dialog box is organized into tabbed pages. The tabs vary according to the type of section. However, every section has a tab for **Output Styles** and one for **Other** options.

Output Styles

The Output Styles tab is for specifying whether the documentation is to be in paragraph, text, or table format.

- **Paragraph Flow:** In this default format, documentation for each element constitutes a single paragraph.
- **Delimited Text Flow:** Delimiters separate the documentation for different elements. You must fill in the delimiter, and you can fill in an output style (a named style defined in the template properties) and font characteristics. There is a check box for printing headers and footers. There is another check box for suppressing formatting options.
- **Table:** Documentation for different elements is written to a table, with one row per element. Within each row, the different pieces of documentation are in different table cells. You can set border styles and cell paddings. There are two check boxes for RTF documentation: print a separate table header on each page and allowing breaking a table over successive pages.

Other options

The Other tab has options for setting the left indent and for specifying enabling conditions. Indents are relative to the indentation for the containing sections rather than the physical paper border.

An *enabling condition* can turn on or off different sections or controls within sections. Checking the **Disabled** box skips the entire section.

Enabling Conditions

Enabling conditions are boolean expressions for turning section processing on or off. If the condition is true, DocGen will process the section; if false, the DocGen will skip the section.

Enabling conditions typically have subexpressions that are properties of the metatype of a section or calls to special *DG functions*. (See Chapter 21, “Documentation Generator and Template Designer Reference” for a list of DG functions and variables.) The results may be joined together with logical operators (==, !=, |, &&) under the usual precedence rules. Here are three examples:

- `hasPropertyValue("$name", "problem_domain")`
- `getContainingNode() -> hasProperty("$interface")`
- `getParam("ShowImages") == "graphics" && getParam("Headers") != "show"`

Accessing model elements through iterator sections

Models are trees composed of elements such as diagrams, packages, classes, operations, and properties. When DocGen reaches an iterator section, it traverses the subtree of the model rooted at the current element. It applies the iterator to the children of the current element that match the iterator's metatype, recursing inside those children if instructed.

There are two kinds of iterators, element iterators and element property iterators. All iterators must have other sections nested within. Iterators may have optional headers and footers.

Element iterators

Element iterators provide a way of looping through elements of a model. Each element iterator has its own metatype, which must be consistent with the metatype of the iterator's parent's. For example, a section with activity diagram metatype may contain a swimlane iterator but it may not contain class or actor iterators.

When DocGen enters an element iteration section, it calculates a new current element according to the current element of the parent section and the metatype of the iterator. DocGen loops through an element iteration section using each possible new element as the current element for that iteration. The properties of an element iterator affect the way a new current element is calculated and how it changes during iterations.

If DocGen enters an iterator and finds no elements corresponding to the iterator's metatype, it will produce no documentation except for an optional header or footer.

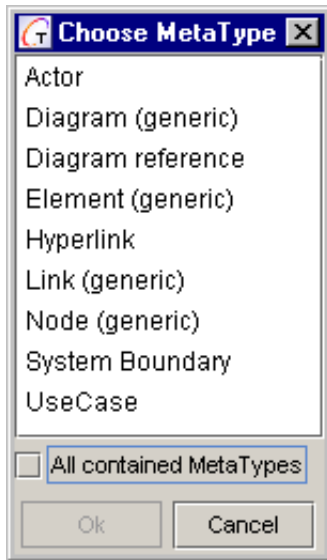
When you create a new element iterator, you must select its metatype from a list whose choices depend on the metatype of the parent section. If you check **All contained Metatypes**, the list expands to include the available metatypes of the items on the list as well. Figure 93 shows the available child metatypes for UseCase Diagram iterator.

TIP: If you want an iterator to be able to access an entire model, choose *Package* or *Diagram (generic)* as the metatype.

After you create an element iterator section, you can change its properties: choose **Properties** from its right-click menu to bring up the Element Iterator dialog box.

NOTE: You can change an iterator's metatype after you create it, but you should keep the new metatype consistent with its parent and any of its children.

Figure 93 Child metatypes for a Use Case diagram iterator



Element iterator scopes

Scope Options are for determining which elements of the model this iterator will document. Each iterator works over the subtree of the model that is rooted at the current element (the element that starts the iteration).

The top of the Scope Options tab has three buttons for limiting the part subtree that the iterator will apply to.

- **Default:** There are no special limitations. The elements considered depend solely on the element starting the iteration and the metatype of the iterator.
- **Customized:** The elements of the subtree under consideration are further limited by two expressions for DocGen to process. One expression describes which element to begin with; the other describes how to get the subsequent element from the current one. (“Formula and text controls” on page 353 has more information on constructing expressions.)
- **Programmed:** This limitation of the subtree under consideration works in conjunction with Together’s open API. This is particularly useful for custom elements. When you choose Programmed, you must specify the name of the method that can access the elements and the class to which it belongs.

With the Default option, you can choose how DocGen should recurse the subtree rooted at the current element to search for elements. There are five different search strategies to find which elements of the subtree to document. You can reject (**no**), allow (**yes**), or partially restrict (**expr**) recursive and other kinds of searching. If you choose **expr**, you must enter an expression to define the restriction.

- **Recurse subpackages:** Search inside packages.
- **Recurse Subnodes:** Search inside elements.
- **Include Parent Element:** Visit the parent element (the current element, from which the iteration was initiated). Normally, an iterator goes through the contents of an

element, not the element itself. This option is for documenting the containing element as well.

- **Visit Diagrams:** Search diagrams as well as packages.
- **Include Shortcuts:** Search shortcuts contained in elements.

The field at the bottom of the Scope Options page is for filter expressions. A filter expression restricts the search to elements that satisfy the filter properties. For example, `!hasPropertyValue("$name", "<default>")` restricts the search to non-`<default>` diagrams or packages.

NOTE: Filters and an enabling condition work at different levels of granularity. A filter works per iteration while an enabling condition works at a global level, turning on or off the entire iterator section.

Sorting

You can specify the order in which the elements are to be searched and thus documented on the Sorting tab of the Element Iterator dialog box. There are three options:

- **None:** the default choice.
- **Element Property:** sort the elements searched according to a property of the section's metatype. You must select the property from a list of properties.
- **Expression:** sort the elements searched according to a string-valued expression. You must enter an expression that DocGen can evaluate. (See “Formula and text controls” on page 353 for information on constructing expressions.)
- The elements are always documented in ascending order.

Element property iterators

Element iterators traverse model elements. Element property iterators traverse element properties instead of elements. For instance, since operation is a class property, you can use a property iterator to loop through all of the operations of a class.

Properties belong to elements. Thus a property iterator can reside only inside an element iterator or folder. (The reverse is not true; elements have properties but properties do not have elements). A property iterator must contain at least one static section, folder section, or call to a stock section or template. It may not contain other property iterators or element iterators.

You can select the properties of an element property iterator when you create it. The Element Properties dialog box has three tabs: Iteration Scope, Output Style, and Other. **Iteration Scope** lists the property to iterate over. There are three sets of properties choices:

- **Set of Properties:** These are properties that belong to the metatype of the parent element iterator. (See “Metamodel types” on page 336.)
- **All User-Defined Properties:** These are properties that are not described in the metamodel. The dialog box has two checkboxes:
 - **Exclude already iterated properties** omits properties that were already iterated for the current element.

- **Iterate only unknown properties** includes only those properties that were not included in the metamodel.
- **Instances of a Single Property:** These are properties that can have multiple values, for example, `@see` or `@author`.

A property iterator can iterate over multiple properties. You must use Ctrl+click to select multiple properties.

Folder sections

Folder sections group other sections together. A folder has at least one nested section, and it may have a header or footer. In that sense, folders are similar to element iterators, except that DocGen executes folders only once.

Folders inherit their metatypes from their parents. The sections nested within a folder must be consistent with its metatype. Folders provide a way to put section-level properties on their contents. This includes enabling conditions for toggling its processing on and off.

Both folders and iterators can have headers and footers. If the sections in a folder do not result in output, then the folder's headers and footers are ignored. Headers and footers of iterators are not ignored, even when no other output for the iterator is produced.

Reusing templates and stock sections

Static sections and iterators are critical for templates. Folders are conveniences for organizing template sections. Calls to stock sections and external templates are essential for reusing existing sections.

Calls to stock sections

Stock sections are reusable folders or iterators that reside in the template's collection of stock sections. They are not shared among different templates.

When DocGen processes a call to a stock section, it is the same as if the called stock section were simply embedded at the position of the call.

Stock sections are especially convenient for frequently used constructs. You can insert a call a stock section from any section whose metatype is consistent with the metatype of the stock section. Stock sections may contain calls to other stock sections as well recursive calls to themselves.

Creating calls to stock sections

When you create a call to a stock section, the resulting dialog box displays a choice of appropriate sections. You can right-click the section to bring up its **Properties** options. The resulting dialog box has two tabbed pages: **Call to**, and **Other**. **Call to** lists the possible stock sections and highlighting the name of the stock section that is actually called. **Other** has a field for the enabling condition and a field for the template parameter.

Creating and editing stock sections

You can create a stock section from an existing folder or element iterator by right-clicking on the section and selecting **Copy Into Stock**. You can also create a new stock section without using an existing section by selecting **Stock | New | Folder** (for a folder root) or **Stock | New | Element Iterator** (for an iterator root) from the main menu of the Document Template Designer. In the resulting dialog box you must enter the new stock section's name. If the root element for the stock section is an iterator, you must also select a metatype.

You can edit a stock section by selecting **Stock | Edit** from the main menu. The resulting dialog box displays the list of stock sections. Click to select, or Ctrl+click to select multiple sections. There are four options:

- **Edit** the selected stock sections, each in its own tab of the Documentation Template Designer.
- **Delete** selected stock sections from the list of stock sections. (All calls to these stock sections are replaced by calls to “<none>.”)
- **Copy** the stock sections into the clipboard.
- **Paste** the contents of the clipboard to create a new stock section. Paste adds copies to the list of stock sections, automatically generating their names (for example, Class, Class1, and so on).

To edit or display a single stock section from the template, select **Show Stock Section** from the right-click menu of a call to the stock section.

When you add a new stock section or edit an existing one, the Document Template Designer window displays a new tab for the stock section. Right-clicking the tab enables you to close or rename the stock section.

Calls to template sections

With a call to a template, DocGen can produce documentation using a different template without terminating the current one.

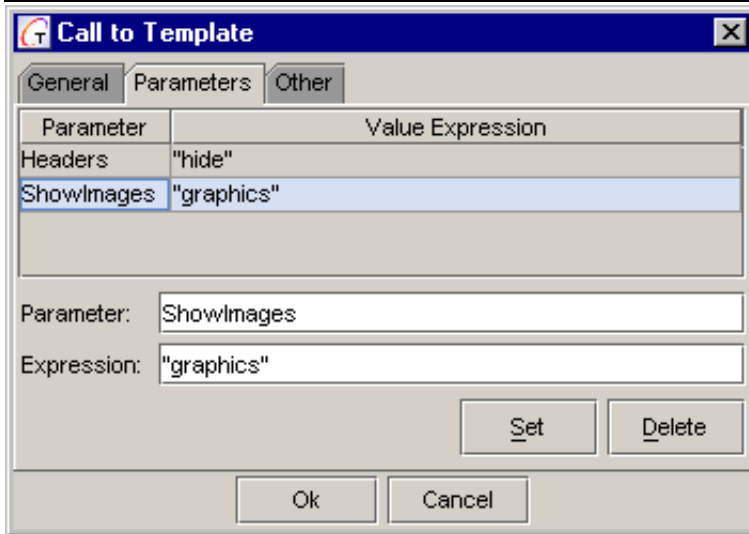
After creating a call to template section, right-click the new section and select **Properties**. This displays the Call to Template dialog box, where you specify the name of the called template. Clicking on the button to the right of the file name text field opens a dialog box for selecting the actual called template.

The Call to Template dialog box gives a choice of where the output for the called template goes:

- **Separate file**. This is important for generating multi-frame HTML documentation consisting of separate HTML documents that are extensively linked together.
- **Common stream**. The called template behaves like a stock section.

When a template is called, the current element of the calling template becomes the root element of the called template. A calling template can pass additional information to the called template through template parameters. Figure 94 shows the tab on the Call to Template dialog box for specifying the parameters and assigning actual values to them. The formal parameters are options of the called template. (See “Setting template properties.”)

Figure 94 Actual parameters for a Call to Template Section



A called template can access actual parameter values through the function `getParam`, which has the following signature:

```
String getParam(String parameterName)
```

Parameters are especially useful for enabling conditions. Here is an example of an enabling condition based on Figure 94:

```
getParam("ShowImages") == "graphics"
```

For more information, see “Enabling Conditions” on page 341 and “Formula and text controls” on page 353.

Calls to templates make it possible to construct a library for generating documentation for particular model elements (class, actor, use case, and so on). By calling pre-designed library templates, you can quickly construct templates for more general reports (such as the built-in template `ClassReport.tpl`) intended for both multi-frame HTML output and RTF printable documentation.

CHAPTER 19

DOCUMENTATION TEMPLATE CONTROLS

The documentation generator in Together ControlCenter uses Together projects and templates to produce project reports. This chapter discusses *controls*, which are the items in documentation templates that determine the contents of reports. Controls are the commands to the DocGen engine for placing data into reports.

This chapter assumes that you understand the concepts discussed in the previous chapter, “Designing Custom Documentation Templates.” The contents of the chapter are:

- “Using static sections, headers, and footers” on page 348
- “Creating controls and setting control properties” on page 350
- “Managing the display of output” on page 354
- “Hyperlinking controls to element documentation” on page 356

Using static sections, headers, and footers

Documentation templates consists of headers, footers, and body sections. The body of a documentation template is organized as a tree of sections. Static sections, calls to stock sections, and calls to templates are the leaves on the tree -- they can have siblings but they cannot have other sections nested within. (See “Body sections and the DocGen engine” on page 335.)

Of the six kinds of body sections, only static sections contain controls for producing actual output. Headers and footers can also contain controls. Folders and iterators, which cannot directly contain controls, must have at least one static section nested somewhere within.

Creating and deleting static sections, headers, and footers

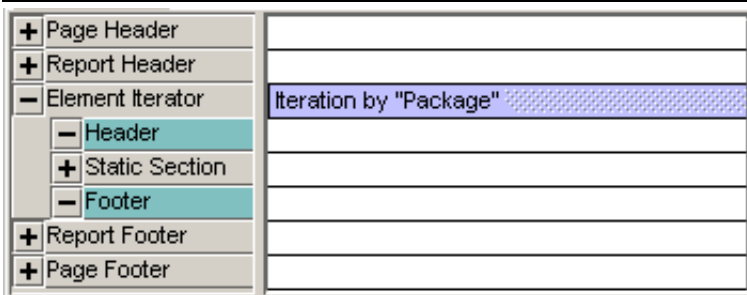
You can create a static section in the usual way: right click an existing section and choose **Insert Nested Section | Static Section** if the existing section is a folder or iterator or **Insert Sibling Section | Static Section** for any existing section. (Alternatively, select the existing section in the details pane and make your choice from **Object** on the main menu.)

The metatype for a static section is the metatype of its parent. (“Body sections and the DocGen engine” on page 335 has an introduction to metatypes.)

Report and page headers and footers lie outside the body of the template. You can turn on or off report and page headers and footers in the template options (use **File | Options** from the Documentation Template Designer main menu).

Iterators and folders can contain headers and footers too. To add a header or footer to a folder or iterator, right click it and select **Add Header** or **Add Footer**. Figure 95 shows the aqua coloring of body section headers and footers in the scope pane.

Figure 95 Section headers and footer coloring



Each header, footer, and static section has two different right-click menus, one for the scope pane (on the left) and the other for the details pane (on the right). The right-click menus on the details pane are for setting area style and format characteristics and for inserting controls.

The right-click menus on the scope pane vary among headers, footers, and static sections. Items on those menus include the following:

- **Delete**: for headers, footers, and static sections with siblings only. Delete for static sections without siblings is disabled.
- **Insert Sibling Section**: for headers and static sections only. The new section goes immediately below this header or section.
- **Move Up, Move Down, Copy**: for static sections only. **Copy** creates a copy of the section in the clipboard. You can paste from the clipboard when you insert a new section.
- **Properties**: for static sections only. Static sections have only one property, which is an enabling condition. (See “Enabling Conditions” on page 341.)

Setting area properties

Area Properties is the top item on right-click menus in the details pane for headers, footers, and static sections. Selecting Area Properties displays a dialog box with three tabbed pages:

- **Settings:** contains check boxes for page settings (such as “start a new page with this section”) and for suppressing formatting.
- **Hypertext Target:** for inserting anchors and bookmarks. (See “Hyperlinking controls to element documentation” for more details.)
- **Other:** identical to the Other tabbed page for all other kinds of sections. (See “Setting section properties” on page 341.)

Creating controls and setting control properties

There are six different kinds of controls. To create a new control in a static section, header, or footer, select **Insert Control** from its right-click menu in the details pane followed by:

- **Label:** for static text
- **Image:** for graphics (project diagrams or other images)
- **Panel:** for a container for other commands
- **Data Control:** for project-specific data
- **Formula:** for project-specific data (necessary when the data cannot be accessed with simple data controls)
- **Include Text:** for including text from external files

When you insert a new control, the Documentation Template Designer displays a dialog box for setting the control’s properties. The template shows each control as a shaded rectangle in the details pane.

You can change the properties of a control after it is created right-clicking the control and choosing **Properties** or by selecting the control and choosing **Object | Properties** from the main menu. The result is a dialog box organized into tabbed pages for the properties settings.

Labels, images, and panels

The simplest kinds of controls are labels, panels, and images.

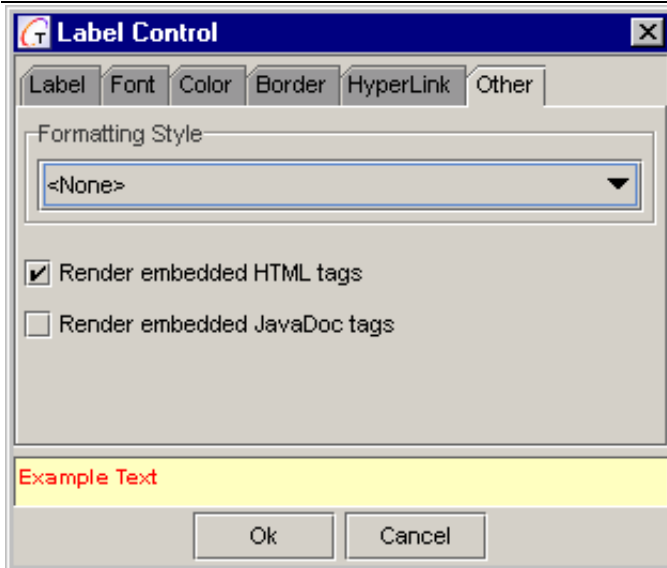
Labels

A label control generates static text that is independent of its containing section -- the text does not depend on the metatype of the section or where the section belongs in the template. Placing identical labels in a header and a static section results in the same output as long as the header and static section are not skipped.

Label properties include the label's text, style (font, color, and border), and if and how to hyperlink the output. (The section "Text format settings" on page 355 has more details on styles.)

Figure 96 shows the Other tabbed page of the Label Control dialog box. There is a menu for choosing a named style from the template options. The checkbox for **Render embedded HTML tags** determines whether to interpret any HTML tags in the label as HTML tags or simply show them as text.

Figure 96 Label properties



Images

An image can be external to the project or it can be a project diagram. The Image tabbed page on the Image Control dialog box has a **Type** menu for selecting which. There are two choices:

- **Static:** for a file (gif, jpeg, or suitable image file type). With this choice, you should specify the name of the file in the File text field at the top of the page. You can fill the name in directly or click the command button to the right of the text field to bring up a file selection dialog box.
- **Diagram:** the image is a diagram from the project. If the containing section metatype is a diagram, DocGen will produce an image of the diagram of the current element when it processes the section. If the containing section metatype is not a diagram, choosing Diagram results in no output.

Images can have borders. They can also be hyperlinked in the same way as labels.

Panels

A panel is simply a container for other controls. Panels are convenient for grouping controls together to provide a uniform style and precise alignment. You can set the panel's background color, border, and style.

To insert a new control into a panel, use the panel's right-click menu or click the panel and go to **Object** on the main menu. Select **Insert Control** followed by the control type. If a copy of a control is in the clipboard, you can choose **Paste** from the menu to insert it into the panel. You cannot drag a control into a panel from outside the panel.

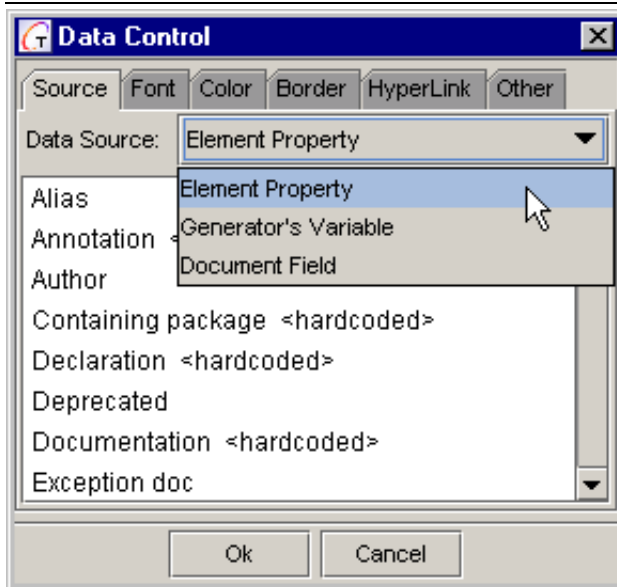
Data controls

Data controls provide the major mechanism of placing data from a project into a report. When DocGen processes a data control, it obtains the actual data from the current model element. (See “Body sections and the DocGen engine” on page 335 for a review of current model element.)

Data Control Sources

The source of data for a data control can be DocGen variables, the generated report, or (most frequently) the project. When you create a data control, you must specify its data source. Figure 97 shows picking the data source from the drop-down menu in the Data Control dialog box.

Figure 97 Data Control dialog box



The menu gives three choices:

- **Element Property:** a property of the current element. The Data Control dialog box displays a list of every property belonging to the metatype of the current model element. Metatype properties are defined in the metamodel files. (See “Metamodel types” on page 336.)

If the root object metatype (for the entire template) is MODEL or <ANY>, the list of Element Properties for report and page headers and footers is empty.

- **Generator's Variable:** a variable used by DocGen. You can use this in report headers or footers to insert the project name or the date and time the report is created. If the data control is inside a property iterator, the list of Generator's Variables includes the property's characteristics (value, name, and so on).
- **Document Field:** a field of the report such as page number or bookmark. You can select Document Field to insert page numbers and number of pages into page headers or footers. The Document Field list is empty for report headers and footers.

The Data Control dialog box has tabbed pages for formatting, hyperlinking, and substituting default text for missing data.

Displaying custom properties

Data controls can generate documentation for custom properties if you make the appropriate modifications, as shown in these two steps.

1. Add the property
2. Edit the file `MetaModel.mm`: by putting the property in two places:
 - in the list of properties at the beginning of the file.
 - in the set of properties belonging the appropriate metatype for that property.

When you create a data control in a section whose metatype is that of the new property, the Data Control dialog box lists the new property among the Element Property sources.

The next section on Formula Controls discusses an alternative way of displaying custom properties.

Formula and text controls

Formulas provide a way to place data into a report that DocGen calculates when it processes the control. You must enter the formula that DocGen evaluates to calculate that output. Both formula controls and text controls rely on such formulas.

Formula controls

A *formula* is an expression that DocGen can evaluate to a string. The expression can be a combination of string literals, *DG variables*, and *RWI functions* from the read-write interface of Together's Open API.

DG variables are special variables that are available to DocGen at runtime when it is producing a report. DG variables include items such as current element, the date and time, and template parameters. A complete list of DG variables and RWI functions is in Chapter 21, "Documentation Generator and Template Designer Reference."

When you insert a formula control, you must enter the formula in the Formula Control dialog box. Formulas use C++ style syntax, with double quotes for string literals, + for string concatenation, and -> for calls to functions via pointers. The following example from a section with class metatype puts *Package* followed by the name of the containing package into the report.

```
"Package " + getContainingPackage() -> getProperty("$name")
```

The next example is a formula control inside a section with a generic class metatype. It puts *Interface* in the report if the current element is an interface and *Class* if it is not.

```
if (hasProperty("$interface"), "Interface", "Class")
```

The remaining properties for a formula control are similar to those for a label control.

To display custom properties in the generated report, you can follow the instructions outlined in the previous section on data controls. Alternatively, you can use a formula control to specify the property. This does not require editing the metamodel file. The function call `getProperty(property_name)` gives access to any property, whether it is included in the metamodel or not.

Include text controls

Include text controls are for copying text from other files into a template. When you insert an include text control, you must enter an expression for the location of the text file. The expression can be hard coded as a string literal, or it can use a formula as described above.

Include text controls have formatting properties identical to those for formula and label controls.

Managing the display of output

The positioning and properties of controls effect their associated display in the generated reports.

Moving and resizing controls

The Documentation Template Designer displays a newly created control as a rectangle positioned at the insertion point. You can move the control to change where its output appears in the generated documentation. You can also modify the size of the rectangle to determine the approximate size of the region for the output. (Precise positioning and sizing is not possible.) Increasing the rectangle size is especially important for a label for which the default size is not large enough to allow its entire text to be displayed in the report.

To move a control within a section, select the control then drag it to the desired position after the mouse pointer changes to a cross of double-ended arrows. To move the edge of a control and thus change the size of its rectangle, select the control and move the pointer to the side. When the pointer changes to a double-ended arrow, you can drag the edge of the rectangle until the rectangle is the desired size.

TIP: You can move multiple controls at the same time with a multiple selection. To select multiple items from lists or controls from sections, hold down the control key as you select them.

You cannot move a control from one section to another by dragging it. Instead, you must **Cut** from the control's right-click menu and **Paste** from the target section's right-click menu.

Text format settings

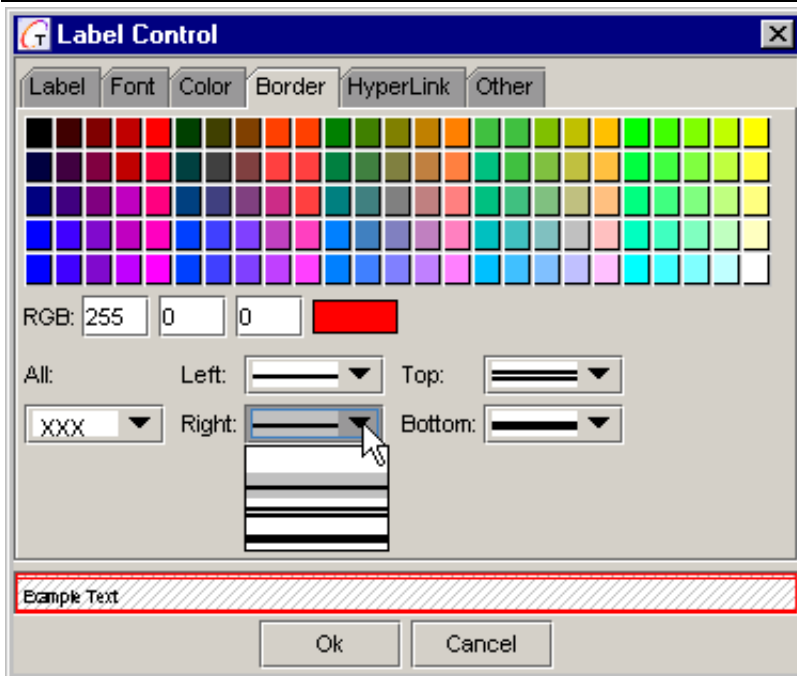
Properties for controls include tabbed pages for formatting the presentation of text: **Font**, **Color**, and **Border**. Each page has a preview box at the bottom for evaluating the format settings.

On the Font page, you can select the font family, size, style, and alignment within the rectangular area of the generated output.

On the Color page, you can select the text and background colors by clicking the color chips or filling in the RGB values. Check the transparent checkbox to the right of the Fill color to make the background transparent. A hatched background in the preview indicates a transparent background.

On the Border page, you can specify whether you want a border, its style, and its color. The border styles are on drop-down lists, as shown in Figure 98.

Figure 98 Setting border properties



Aligning controls

You can move a single control within a section to determine the positioning of the associated output and its ordering within the output for the entire section. You can also move multiple controls by first selecting them with Ctrl+ click.

If you select two or more controls in a section, you can use the right-click menu to uniformly align them within the section. The alignment options are **Left Side**, **Right**

Side, **Top Side**, and **Bottom Side**. You should take care when applying multiple alignments since it is possible for controls to overlap.

You can also use the right-click menu to apply uniform sizes to the controls' enclosing rectangles: **Make same width**, **Make same height**, and **Make same size**.

NOTE: There is no simple undo for changes in alignment or size. When you change alignment or resizing, you must manually readjust the controls to return them to the former status.

Hyperlinking controls to element documentation

A *hypertext link* connects a *link reference* (starting point or source) to a *link destination* (*target*). The link reference is a text or image in the HTML document. The link destination is a file (usually an HTML document or an anchor in an HTML document).

Document templates support both references and targets. Link references are properties of controls. Link targets are properties of static sections, headers, and footers.

Making a target from a static section, footer, or header

Any generated output that contains an anchor or bookmark can be a link target. Documentation templates have facilities for inserting anchors at the “main documentation” of model elements.

You can insert anchors for static sections, headers, and footers. Start with **Area Properties** on the section's right-click menu and go to the **Hyperlink** tabbed page. The page displays three items:

- **Expression for Target Bookmark Selector:** a text field.
- **Start of the current element's specific documentation:** a checkbox to identify the output of this section as the “main documentation” for the current element. When DocGen processes the section, it inserts a hypertext anchor or bookmark into the output, automatically generating its name. DocGen recognizes this section as the *element's main documentation*.
- **Expression for Documentation Subject Selector:** a text field also used for documentation for the current element.

Linking a control to a target

When you define the location of a model element's main documentation, you can specify hyperlink references to it for any control that is not a panel. The control can be in the same template as the main documentation or it can be in a different template.

To hyperlink a control to a target, select **Properties** on the control's right-click menu. Then go to the **Hyperlink** tab. Figure 99 shows the Hyperlink tab.

There are three choices of hyperlink target types:

- **File Link:** You must fill in the path to the file. If the hyperlink target is a bookmark in the file, you must supply that as well.
- **URL Link:** You must supply the URL.

- **Link to Element's specific Doc:** You must identify the element whose documentation is to be the target in the text field for **Expression for RWI-Element**.

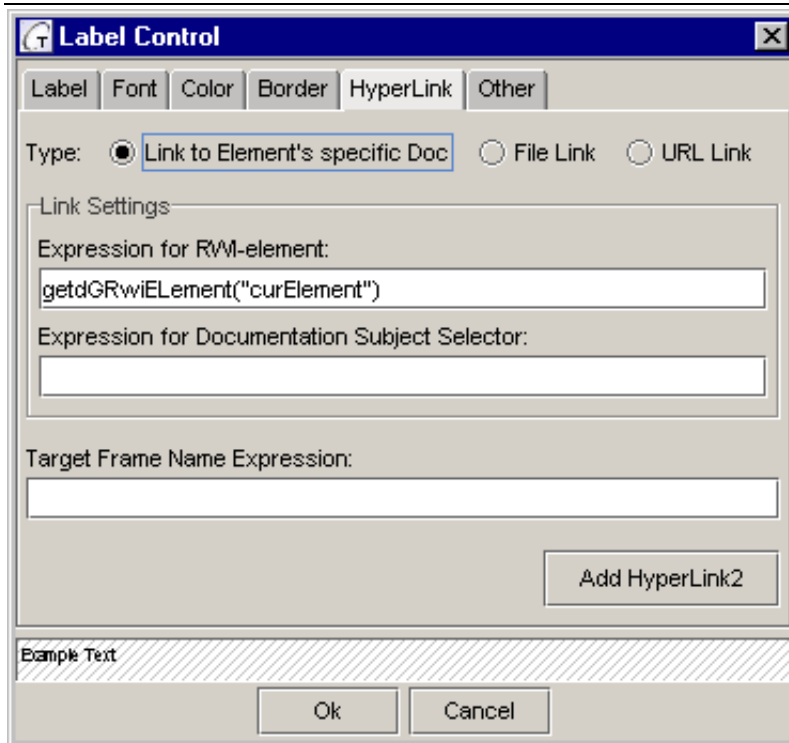
DocGen evaluates the **Expression for RWI-Element** to determine which element's documentation is the hyperlink target. For hyperlinking to the current model element, you can use the following expression:

```
getDGRwiElement("curElement")
```

This is the same expression as in Figure 99. Expressions can be more complicated, such as:

```
findElement(getDGRwiProperty("curPropertyInstance")->
    getSubproperty("$referencedElement"))
```

Figure 99 Hyperlinking a label to an element's main documentation



“Creating hypertext links (advanced)” on page 363 has substantial additional information on documentation hyperlinks.

CHAPTER 20

CREATING MULTI-FRAME HTML DOCUMENTATION

Together ControlCenter can generate simple HTML, RTF, PDF, and multi-frame HTML documentation. This chapter discusses how to use the Documentation Template Designer to create the templates for multi-frame HTML documentation. It assumes that you understand the material covered in the two previous chapters, Chapter 18, “Designing Custom Documentation Templates” and Chapter 19, “Documentation Template Controls.”

The contents of this chapter are:

- “Multi-frame HTML document template basics” on page 358.
- “Defining the frameset structure” on page 359.
- “Designing the frameset template body” on page 361.
- “Creating hypertext links (advanced)” on page 363.

Multi-frame HTML document template basics

Multi-frame HTML documentation divides project reports in frames to give multiple views within the same browser window. Multi-frame HTML documentation consists of two kinds of HTML files:

- A collection of HTML files to define the content for each frame
- A *frameset* file to specify the layout of frames

The frameset file is the starting point for viewing the documentation. It has hyperlinks to the HTML files for its frames.

Frameset document template organization

A frameset template consists of two major parts. One describes the frameset file. The other, which is the body of the frameset template, contains calls to the templates that provide the contents of the frames.

Every frameset template is designed for a particular type of model element (a metatype) for which it can produce specific documentation. For example, you can design a frameset template for a class, an actor, or for an entire model. This is the same as for ordinary document templates. (See “Metamodel types” on page 336.)

Creating frameset templates

To create a new frameset documentation template, choose **File | New | New Template** from the main menu of the Documentation Template Designer. In the resulting dialog box, select the model type (**GUI-Builder Model** or **RWI meta-model**) and select **Frameset Template** for the **Template Type**.

Once you create a frameset template, you cannot convert it into a document template or vice versa.

NOTE: By convention, the names of frameset template end in “Frames.” For example, “ProjectReportFrames.tpl” and “ClassReportFrames.tpl,” which are part of the Together installation.

Defining the frameset structure

The *frameset structure* of a multi-frame HTML document describes how the frames are organized within the browser window. You can set the structure in a frameset template through the template properties. Select **File | Properties** from the main menu of the Documentation Template Designer. In the resulting **Template Properties** dialog box, go to the **FrameSet Structure** tabbed page as shown in Figure 100.

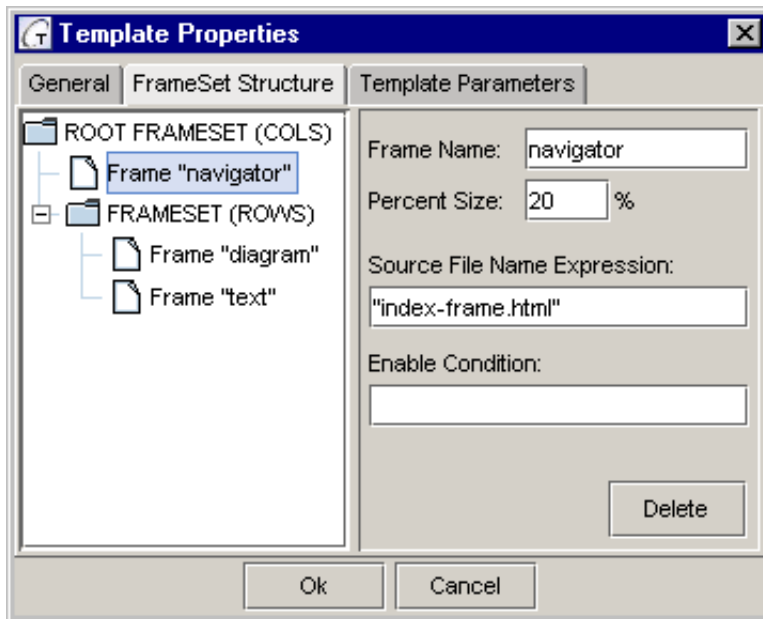
The FrameSet Structure tab shows the frameset as tree on its left panel. The tree has two kinds of nodes:

- **Frame:** a frame to display a single HTML document. This corresponds to the `<frame>` tag in HTML. Frame nodes are leaves in the structure tree. Figure 100 shows three Frames: navigator, diagram, and text.
- **FRAMESET:** a container for additional Frames or FRAMESETs. This corresponds to the `<frameset>` tag in HTML. In the tree, a FRAMESET is the parent to the child Frames and FRAMESETs that it contains. The tree in Figure 100 has two FRAMESETs and three Frames.

In the tree, nodes with children have folder icons. Nodes without children have page icons. The root-level node is always a FRAMESET.

The right pane of the FrameSet Structure tab shows the properties of the currently selected item in the tree.

Figure 100 Frameset structure



Specifying FRAMESET properties

The topmost property of the FRAMESET is its **Layout**, with radio buttons for selecting **Columns** or **Rows**. A FRAMESET with a row layout divides its window (HTML frame) into rows, with one frame per row for each of its children. A FRAMESET with a column layout divides its window into columns, with one frame per column for each child.

The frameset structure of Figure 100 has a root frameset organized into columns. The navigator frame is the left column. The right column is an additional frameset, which is divided into two rows, one for diagram and one for text.

You can assign a **Percent Size** to each FRAMESET child to determine the percentage of the FRAMESET's total space to be allocated to the child. For example, the width of the navigator Frame of Figure 100 is 20% of the root window. The total of the sizes of a FRAMESET's children should be 100%. Otherwise, the browser will decide the sizes for the children when it displays the documentation.

A FRAMESET has an optional enable condition, which determines if the FRAMESET is to be skipped or included when DocGen generates the frameset file. This condition is identical to that for body sections. (See "Enabling Conditions" on page 341 for more details.)

Specifying Frame properties

The DocGen engine translates the **Frame Name** into the name parameter (the name of the frame as a target) of the corresponding `<frame>` tag. You can use that name in a

hyperlink to load the referenced document into the frame window. The tree in the left pane of the Frameset Structure tab shows the Frame names.

The DocGen engine evaluates the **Source File Name Expression** expression to determine the name of the HTML file that will be initially loaded into the frame. The section “Assigning a target frame to a link reference” on page 364 has additional details of this feature.

Enable Condition and Percent Size for a Frame are identical to those for a FRAMESET.

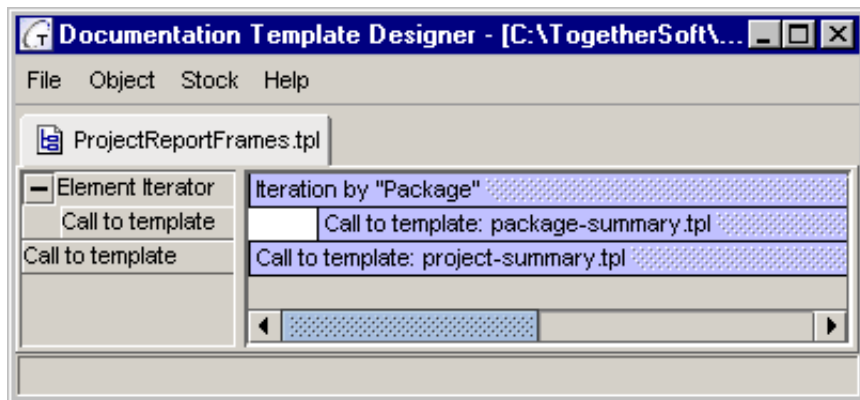
Designing the frameset template body

The body of a frameset template determines the documents that DocGen will create (or use) to load into the frames of the resulting project report.

Frameset template body organization

The body of a frameset template is similar to the body of an ordinary document template. Figure 101 shows the “ProjectReportFrames” template, which ships with your Together installation.

Figure 101 Frameset documentation template



A frameset template body can contain any number of iteration sections (element iterators and property iterators), folder sections, and stock section calls. However, static sections and headers and footers for folder sections and iterators are prohibited. Calls to template sections replace static sections to produce the actual output.

NOTE: The body of a frameset template produces no separate output file.

Frameset template processing

When DocGen processes a frameset template, it produces the frameset HTML file as well as the separate HTML files for the frame content.

DocGen begins processing a frameset template at its body. When it encounters a call to a template section, the DocGen engine suspends the current template execution, loads the called template, and processes it to produce a separate HTML document. The root element for the called template is the current model element of the calling template. After it completes processing the called template, DocGen resumes executing the calling template.

After DocGen completes processing the body of the frameset template, it produces the special HTML frameset file. This file corresponds to the frameset structure specified in the template properties. The name of the frameset file matches the name of the frameset template. It is the starting point of the generated documentation.

Call to template section properties

The section properties of a call to template determine how the output for a template call can be used. With multi-frame HTML documentation, call to template sections typically generate separate files that can be loaded into a frame of the resulting HTML project documentation.

NOTE: You can set the name of the file to be loaded into a frame through the Frameset Structure tab of the template properties. You can set the name of the output file generated from a call to template section through the section properties.

To access the properties of a call to template section, select **Properties** from the section's right-click menu.

The General tab of the Call to Template properties dialog box has the output settings. If the output generated from the template is to be loaded into a frame, you should select **Separate File** from the radio buttons at the top of the page.

Figure 102 shows the properties of the second call to template section from the "ProjectReportFrames" template of Figure 101.

Naming the generated document

Output File Name Expression is the name of the document. This expression is for the document name only; it should not include the file path.

If a particular call of a template is to be iterated many times to produce multiple documents, you should derive the output document name from the properties of the current model element. You can use the expression `getProperty("$name")` to get the name of the current model element.

If the Output File Name Expression is blank, DocGen names the generated document according to the name of the called template.

Naming the output directory

Output Directory Expression is the path to the destination directory of the generated document. This path is always relative. You should define it according to the following conventions:

1. If the calling template is a frameset template, the path is relative to the destination directory for the entire documentation.

2. If the calling template is a document template, then the path is relative to the location of the document that is generated by the calling template.
3. The right slash character "/" is the name-separator for the path.

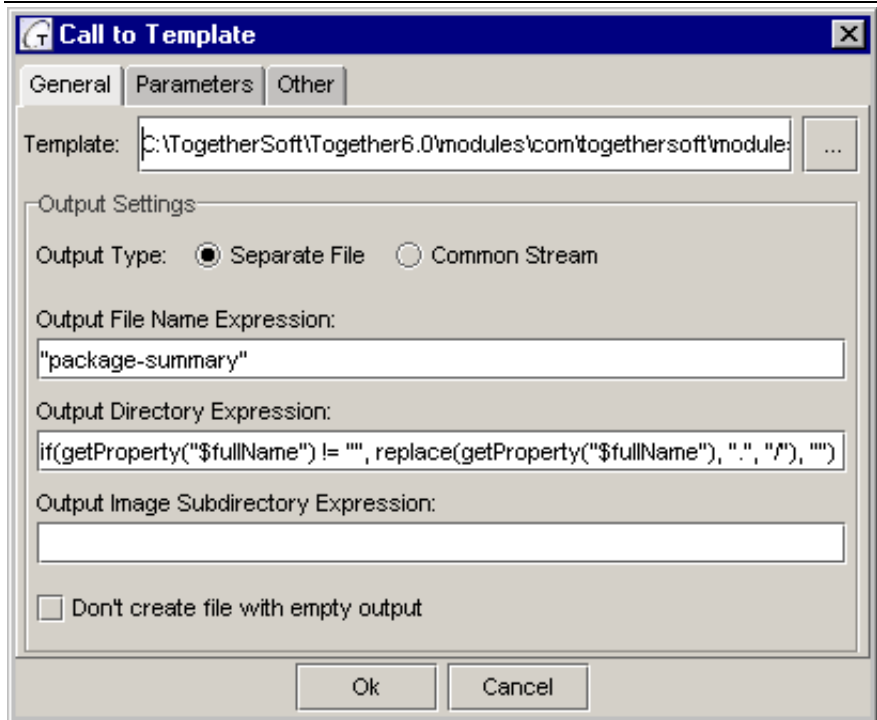
If the expression contains directories that do not yet exist, DocGen will create them when it processes the template.

The template in Figure 102 sets up a directory structure for the documentation to mirror the package structure of the model by having this expression as part of the path.

```
replace(getProperty("$fullName"), ".", "/")
```

The **Output Image Subdirectory Expression** is similar to the Output Directory Expression, except that it is meant for image files rather than HTML files.

Figure 102 Hyperlink for a call to template section



Creating hypertext links (advanced)

Multi-frame HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image. The link destination is a file or an anchor in a file.

“Hyperlinking controls to element documentation” on page 356 discusses hypertext links in the context of ordinary document templates. This section expands that discussion.

Recall how the Documentation Template Designer supports references and targets:

- Link references are properties of controls. To access those properties, select **Properties** from the control's right-click menu. Use the HyperLink tabbed page of the resulting dialog box.
- Link targets are files, URLs, or properties of static sections, headers, and footers. To access those properties, select **Area Properties** from the section's right-click menu. Use the Hypertext Target tabbed page of the resulting dialog box.

Assigning a target frame to a link reference

By default, browsers load a link target in the same frame as the page containing the link reference. A target parameter in the HTML hyperlink tag changes that default behavior to load the target file into a named frame.

The Documentation Template Designer allows you to set the target frame to a named frame. To change that target for a control that is a link reference, go to the HyperLink page of the control's properties. Use the text field of the **Target Frame Name Expression** to enter an expression for the name of a frame window defined in the frameset structure.

Targeting an element's "specific" documentation

"Hyperlinking controls to element documentation" on page 356 explains how to simplify creating hypertext links using the notion of an element's "main" documentation. The advantage of this is that DocGen makes all necessary calculations and markups for the hyper-references.

It is occasionally necessary to provide link references to several different documents (or locations in HTML files) created with the same model element. For example, along with the main documentation file created for a package, there could be a different HTML document that simply lists all classes in the package. If this listing document were in a separate "navigation" pane, it would serve as an index for the package. Clicking the package on a diagram (or in some more general text) could load that listing document in the navigation frame.

The Documentation Template Designer enables you to target different documentation locations generated by the same model element. Follow the same steps as for targeting the location of an element's main documentation:

1. Invoke the **Area Properties** for the template area where the location starts.
2. Choose **Hypertext Target** tab of the Area Properties.
3. Check **Start of the current element's specific documentation**.
4. Fill in an expression for **Expression for Documentation Subject Selector**.

After identifying specific documentation in this manner, you can define link references to that documentation in the same way as for an element's main documentation. To create such a link reference for a control:

1. Invoke the **Properties** for the control.
2. Choose **HyperLink** tab of the Properties.
3. Select **Link to Element's specific Doc** from the radio buttons at the top.

4. Fill in the **Expression for RWI-Element** text field to determine which element's documentation is the link target. (See "Formula controls" on page 353 for a discussion of RWI elements.)
5. Fill **Expression for Documentation Subject Selector** to match the expression for template area described above.

Let us resume our previous example of creating an index for a navigation page. To make link references to the package's index document, we could mark the first area of the template for this document as the Start of the current element's specific documentation, using "packageIndex" as the Subject Selector. We would then enter "packageIndex" as the Subject Selector for any link references to this document.

NOTE: The main documentation of an element is not merely an element's specific documentation with empty subject selector. Only the element's main documentation can be referenced from JavaDoc link references (those with the @link JavaDoc tag) embedded in the text returned by some RWI-properties. You cannot specify subject selector for these references!

Image mapping diagram elements

You can put an image control in a static section to include an image of a diagram in the generated document. DocGen will create an image only if the current model element represents a model diagram when it processes the section.

Image controls are similar to other controls in that their properties include a Hyperlink tab. If the image is not a diagram, the Hyperlink tab enables you to create an ordinary link reference. However, when the image control is for a diagram in the model, the same definition in the HyperLink tab creates link references for all model elements depicted on the diagram -- the resulting image will be "image mapped." To create the image map, you must enter all expressions in the link reference definition relative to the RWI-element returned by the call `getDGRwiElement("diagramMapElement")`.

DocGen generates the image map by iterating through elements of the diagram, substituting the variable `diagramMapElement` with every diagram element, calculating a link reference for it, and inserting the link reference into the image map.

Creating compound link references

Link references in multi-frame documentation may have multiple targets. Clicking on such a reference could simultaneously load two different documents in two different frames. For example, suppose a diagram element represents a package. Clicking on this element could load the image of the package diagram in one frame and the main (textual) documentation for the package in another.

To program such dual targets, you can define two different link references for a single control. Go to the HyperLink tab of control's properties dialog box click the **Add HyperLink2** button at the bottom. Then you can specify the second link reference definition.

JavaDoc link references

JavaDoc References (or *JDocRefs*) are the expressions associated with JavaDoc tags such as `{@link}` and `@see`. You can use them to create link references inside documentation text (`{@link}`) as well as with some other documenting tags. DocGen can convert JDocRefs into real hypertext links.

Each JDocRef should conform to the rules described in JavaDoc documentation provided with any JDK. There are three types of JavaDoc references.

1. *element reference* refers to an element of the model (method, class, package, etc.). The general form of an element reference is: `package.class#member label`, where `package.class#member` is the referenced model element and `label` is optional text to be displayed with the link. (If `label` is omitted, the name of the referenced element is displayed). DocGen can convert each element reference into a hyperlink to the main documentation of the element.
2. *url reference* represents a link to a relative or absolute URL. The general form of a url reference is: `<a href="URL#value">label</a>`
3. *text reference* has the form "string" (a text string in double-quotes). A text reference is simply information that does not represent a hyperlink.

Converting JDocRefs into hyperlinks

A JDocRef appears in one of two forms:

- inside `{@link}` tags embedded in documentation text. The JDocRef is the value of the `$doc` property and other JavaDoc element's properties.
- as the value of some JavaDoc element's properties such as `see`.

The Documentation Template Designer provides conversions for both cases. You need to specify the conversion in the properties of the control.

Converting `{@link}` tags

Only a text control (label control, data control, or formula control) can generate documentation text. To convert `{@link}` tags to hyperlinks, access the control's properties from its right-click menu. Then go to the **Other** tab and check **Render embedded JavaDoc tags**.

Converting the value of an element's property

Converting a value of an element's property to a hyperlink is more complicated than converting an `{@link}` tag. The conversions require using one of these documentation generation functions:

- `getJDocRefType()`
- `getJDocRefDisplayName()`
- `getJDocRefElement()`
- `getJDocRefURL()`

The remainder of this section is an example to illustrate how to display the values of the `see` properties with its associated hyperlinks. The template body for accessing those values is shown in Figure 103.

Figure 103 Iterator to display the see property values

Property Iterator	Property Iterator: See Also
☐ Header	See:
☐ Static Sec	<code>getJRefDisplayName (getdGVariable ("curPropertyInstance"))</code>
☐ Static Sec	<code>getJRefDisplayName (getdGVariable ("curPropertyInstance"))</code>
☐ Static Sec	<code>getJRefDisplayName (getdGVariable ("curPropertyInstance"))</code>

The example uses a property iterator to go through the instances of see since see can potentially have several values. The property iterator contains three static sections that correspond to the three types of JavaDoc references. Each static section has its own enable condition, which activates it for the appropriate JRef.

The following enable condition activates the first static section for element type JRefs only:

```
getJRefType(getDGVariable("curPropertyInstance")) == "element"
```

Each static section area contains a formula control and nothing else. All formula controls use the same expression:

```
getJRefDisplayName(getDGVariable("curPropertyInstance"))
```

The value of the expression is the text that will be displayed in the documentation.

Figure 104 shows the HyperLink tab in the properties dialog of the formula control for the first static section. Note the value for Expression for RWI-element.

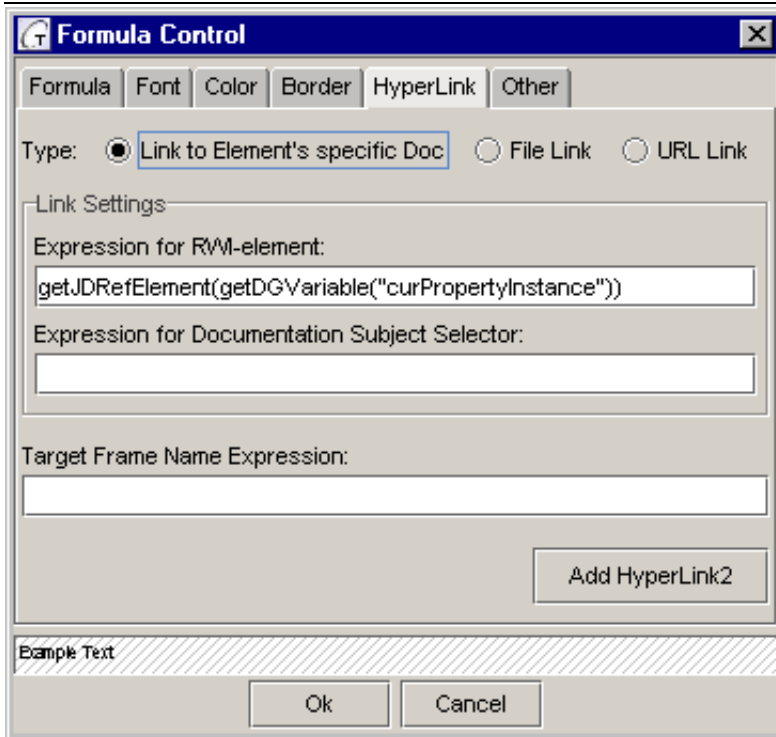
The function `getJRefElement()` returns an RWI-element referenced by JRef. The target of the link is the element's main documentation.

The second static section is activated when JRef's type is url. It is similar to the first one. Its formula control hyperlink is a URL link and its Expression for URL is:

```
getJRefURL(getDGVariable("curPropertyInstance"))
```

The third static section is activated for JRefs of text type. It differs from the previous two in that its formula control has no hyperlink definition. Actually, it can be combined with the first static sections. Since such a JRef does not refer to any element, the function `getJRefElement()` always returns null, producing no hyperlinks.

Figure 104 HyperLink tab for a Formula Control



CHAPTER 21

DOCUMENTATION GENERATOR AND TEMPLATE DESIGNER REFERENCE

When using the Documentation Template Designer to develop custom documentation templates for the Documentation Generator module, you can reference the internal variables and functions to specify formula expressions and provide flow of control among sections. This chapter lists the variables and functions available for those expressions.

The contents of this chapter are:

- “Documentation generator variables” on page 369.
- “Documentation generator functions” on page 372.

Documentation generator variables

When Together’s Documentation Generator executes a template and generates a project report, it uses internal information that includes such things as the project name and the current date/time as well as temporary processing data such as the current model element. DG variables enable you to insert this information into the report. Each variable represents some kind of internal Documentation Generator information available at any particular moment during the generation process.

Not all DG variables are accessible at any instant. Most of them appear only in special areas or inside special sections. Some of them rely on the metamodel types, which are discussed in “Metamodel types” on page 336.

There are three different types of DG variables:

- String
- RwiElement
- RwiProperty

You can access DG variables by using appropriate DG functions in formula expressions:

- getDGVariable
- getDGRwiElement
- getDGRwiProperty

Table 42 lists the DG variables. Table 43 lists the DG functions.

Table 42 Documentation generator variables

Name and type	Description
curItemNo: String	Current iteration item number (starting at 1). Availability: inside property and element iterators Access via: getDGVariable
curPropertyName: String	Name of the current property. Availability: inside property iterators Access via: getDGVariable
curPropertyFullName: String	Full name of the current property as specified in MetaModel file. (See “Metamodel types” on page 336.) Availability: inside property iterators Access via: getDGVariable
curPropertyType: String	Type of the element property as specified in the MetaModel file. (See “Metamodel types” on page 336.) Availability: inside property iterators Access via: getDGVariable
curPropertyValue: String	Value of the current property. Availability: inside property iterators Access via: getDGVariable
curPropertyInstance: RwiProperty	RwiProperty object of the current property instance. This variable is useful when you need a subproperty of the current property instance. Example: Suppose the current model element is a <i>class</i> and you need to list information about all <i>interfaces</i> implemented by this class. You must create a section that iterates by instances of the IMPLEMENTS property of the current <i>class</i> element. Within this iteration section, you can use curPropertyInstance to access the subproperty REFERENCED_ELEMENT, which gives access to information about the implementing class. If you need the <i>full names</i> of the implemented interfaces, you can use the expression: <pre>findElement(getDGRwiProperty("curPropertyInstance")-> getSubproperty("\$referencedElement"))-> getProperty("\$fullName")</pre> Availability: inside property iterators while iterating by instances of the specified property Access via: getDGRwiProperty

Table 42 Documentation generator variables (continued)

Name and type	Description
curPropertyInstance: String	Value of the current property instance. Availability: inside property iterators while iterating by instances of the specified property Access via: getDGVariable
curElement: RwiElement	Current model element. Availability: inside element iterators Access via: getDGRwiProperty
prevElement: RwiElement	Previous element in the current iteration scope. Possible value: null if this is the beginning of the scope Availability: inside element iterators Access via: getDGRwiProperty
diagramMapElement: RwiElement	Diagram element; used to create images maps for diagrams. Availability: inside image controls Access via: getDGRwiElement
projectName: String	Project name. Availability: inside report / page header / footers Access via: getDGVariable
nowDateTime: String	Current date/time. Availability: inside report / page header / footers Access via: getDGVariable
outputFormat: String	Output format of the generated documentation. Use this variable to control the behavior of your templates based on the output format for the generator. Possible Values: "RTF", "HTML", "TXT" Availability: Anywhere Access via: getDGVariable
reportScope: String	Shows the report scope. Possible values: "all_model" - scope is the whole model "current_package" - scope is the current package only "current_package_recursive" - scope is the current package and its subpackages "current_diagram" - scope is the current diagram only Availability: Anywhere Access via: getDGVariable
stockParam : String	Parameter of the stock section call. Availability: inside stock sections Access via: getDGVariable

Documentation generator functions

Table 43 lists the major functions that can be used in formula expressions and enabling conditions.

Table 43 Documentation generator functions

Function name	Signature and Description
getDGVariable	String getDGVariable(String variableName) Parameter: name of a variable Returns: value of variableName or the empty string if there is no such variable defined where the function is called.
getDGRwiElement	RwiElement getDGRwiElement(String variableName) Parameter: name of a variable Returns: RwiElement value of the variable of the specified name; null if the variable is not defined where the function is called
getDGRwiProperty	RwiElement getDGRwiProperty(String variableName) Parameter: name of a variable Returns: RwiProperty type DG variable of the variable of the specified name; null if the variable is not defined where the function is called
getDGOption	String getDGOption(String optionName) Parameter: name of a option Returns: value of the option with the specified name; the empty string if no such option is defined where the function is called Features: The option can be specified for an object of Report Generator (descendant of the class <code>..gendoc.docgenerator.Generic.GnrReportGenerator</code>, for example: class <code>..gendoc.docgenerator.txt.TXTReportGenerator</code>) using the method <code>addReportOption (String optionName,String optionValue)</code> Default values for some options can be defined in the template file. These definitions persists even when the Document Template Designer subsequently modifies the template. However, the method <code>addReportOption</code> overwrites the options values. Example: default values for the options <code>inclSubpackages</code>, <code>inclDoc</code>, <code>DTLAdapter</code> <pre>DEFAULT_OPTIONS={inclSubpackages='yes';inclDoc='yes'; DTLAdapter='com.togethersoft.modules.doorslink.DTLAdapter'}</pre>
getPar	String getParam(String paramName) Parameter: parameter name Returns: value of specified template parameter. Warning: The requested parameter should be declared in the Template Parameters tab of Template Properties. If the parameter is not declared, calling this function will cause an error message and stop the generator. Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
invokeForName	String invokeForName(String className, String methodName) String invokeForName(String className, String methodName, String param1) String invokeForName(String className, String methodName, String param1, String param2) Parameters: className: fully-qualified name of the user-provided class. This class should not be abstract. The Documentation Generator creates an instance of className and calls the method methodName with this instance. An instance object is created for each entry of invokeForName call within each particular expression of template where this function is used. However, the instance is created only during the first call from such an entry and will be used for the next calls, unless the actual className parameter is changed. methodName: name of the method in the class className. The method should have the following signature: String methodName (..gendoc.api.GenDocContext) The last parameter is an instance of ..gendoc.api.GenDocContext, which provides the following methods: • RwiReference getRwiReference returns RwiReference if the current DG iteration element is an RWI-reference within a diagram. Otherwise the method returns null. • RwiElement getRwiElement returns an RwiElement. If the current DG iteration element is an RWI-reference, then returned element is rwiReference.getElement(). Otherwise, returned element is the current DG iteration RWI-element. • String getParameter1() returns the value of the first optional parameter passed to invokeForName function, or null if the parameter is omitted. • String getParameter2() returns the value of the second optional parameter passed to invokeForName function, or null if the parameter is omitted. Returns: value calculated by the method named methodName of the user-provided class.
getContainingDiagram	RwiDiagram getContainingDiagram() Returns: RWI-diagram containing the primary reference to the current element Example:prwiElement -> getContainingDiagram()

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
isDiagram	<code>boolean isDiagram()</code> Returns: true if the current RWI-element is a diagram.; false otherwise. (You can call this function to test any RWI-element of an expression.) Example calls: <pre> rwiElement->isDiagram() getDGRWIElement("diagramMapElement")->isDiagram() </pre> This function is useful in designing Multi-Frame documentation when you need some special behavior when clicking hyperlinks. For example, you may want clicking a hyperlink to a diagram to reload one frame with a document describing the diagram and another frame with the graphic chart of this diagram. Alternatively, clicking hyperlink to any other model element would load only the document frame. See also , “Creating compound link references” on page 365.
isImported	<code>boolean isImported()</code> Returns: true if the current element in the diagram is a shortcut; false if it is not a shortcut
getSubproperty	<code>String getSubproperty(RwiProperty rwiProperty, String subpropertyName)</code> Parameters: rwiProperty: the element property subpropertyName: the name of its subproperty Returns: the value of subpropertyName contained in rwiProperty. Example call: rwiProperty->getSubproperty(subpropertyName)

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
hasSubproperty	String hasSubproperty(RwiProperty rwiProperty, String subpropertyName) Parameters: rwiProperty: the element property subpropertyName: the name of its subproperty Returns: true if rwiProperty has the specified subproperty; false otherwise. Example call: rwiProperty->hasSubproperty(subpropertyName)String getJdRefType(String jdref) Parameter: JavaDoc reference Returns: type of jdref as follows: "element" if jdref is a model element (if jdref has the form package.class#member label) "url" if jdref is a URL (if jdref has the form label) "text" if jdref has the form "string" Since: Together 5
getJdRefType	String getJdRefType(String jdref) Parameter: JavaDoc reference Returns: the JavaDoc reference type as: "element" if jdref references a model element (if it has the form: package.class#member label, where package.class#member represents some model element) "url" if jdref references a url (if it has the form label) "text" if jdref has the form "string" Since: Together 5
getJdRefDisplayName	String getJdRefDisplayName(String jdref) Parameter: JavaDoc reference Returns: text to be displayed in place of the specified JavaDoc Reference: if jdref is an "element" reference (if it has the form: package.class#member label, where package.class#member represents some model element) the returned text is label. If the label is omitted, returns the name of the referenced element. if jdref is a "url" reference (if it has the form label) the returned text is label. if jdref has the form "string", the returned text is string. Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
getJDRefElement	RwiElement getJDRefElement(String jdref) Parameter: JavaDoc reference Returns: If the specified JavaDoc Reference is an "element" reference (if it has the form <code>package.class#member label</code>, where <code>package.class#member</code> represents some model element) and referenced element exists in the model, the function returns this element; otherwise returns null. Since: Together 5
getJDRefURL	String getJDRefURL(String jdref) Returns: If the specified JavaDoc Reference is a "url" reference (if it has the form <code>label</code>) the function returns the text <code>URL#value</code>; otherwise, returns an empty string Since: Together 5
getCodeElement	Object getCodeElement(RwiElement rwiElement) Returns: An object. Details: The function passes the call to <code>rwiElement.getCodeElement()</code> method declared in <code>com.togethersoft.openapi.rwi.RwiElement</code> interface. Usage: This function is used in the template expressions in conjunction with one of the following functions: <code>findMember()</code>, <code>findNode()</code>, <code>findLink()</code>, <code>findPackage()</code>. Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
getCodeElements	Enumeration getCodeElements(RwiElement rwiElement) Parameter: an RWI-element. Returns: code elements associated with the parameter. Details: This function passes the call to the <code>rwiElement.getCodeElements()</code> method declared in <code>com.togethersoftware.openapi.rwi.RwiElement</code> interface. Usage: This function is used in the template expressions with one of the following: <code>findDocumentedMember()</code>, <code>findDocumentedNode()</code>, <code>findDocumentedLink()</code>, <code>findDocumentedPackage()</code>. These functions are helpful for creating hyperlinks from elements on a diagram. Example: When <code>Recognize Java Bean/C++ properties</code> is on, each <code>JavaBean/C++</code> property is represented by a single element on a class diagram. Each property actually consists of its attribute plus its setter and getter methods. When you generate the class documentation, those 3 elements will be documented (or at least documentation for accessor methods if private members are skipped). The corresponding element on the diagram associates with an RWI-element that you can obtain via the variable <code>diagramMapElement</code> (see also: “Image mapping diagram elements” on page 365). But this RWI-element is actually a kind of a proxy. It is not identical to any of those 3 elements making up the <code>JavaBean/C++</code> property. Thus, you cannot directly use the RWI-element representing a <code>JavaBean/C++</code> property on the diagram to establish a hyperlink to anything contained in the generated documentation. The solution is to use this expression: <pre>findDocumentedMember(getCodeElements(getDGRwIElement("diagramMapElement")))</pre> The function <code>findDocumentedMember()</code> returns one of the RWI-elements associated with the <code>JavaBean/C++</code> property and which is definitely presented in the generated documentation. Since a diagram contains ordinary elements as well as properties, the expression for diagram hyperlinks connecting RWI-elements may be more complicated: <pre>if(getDGRwIElement("diagramMapElement")-> hasPropertyValue("\$shapeType","BeanProperty"), findDocumentedMember(getCodeElements(getDGRwIElement("diagramMapElement"))), getDGRwIElement("diagramMapElement"))</pre> Since: Together 5
findElement	RwiElement findElement(String uniqueName) Parameter: string with the unique name of an RWI-element to be found Returns: an element of the given name. Details: This function passes the call to the <code>RwiModel.findElement()</code> method to do the search.

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
findMember	<code>RwiElement findMember(Object codeElement)</code> Parameter: code element of interest. Returns: the member for codeElement. Details: This function passes the call to the method <code>com.togethersoft.openapi.rwi.RwiModel.findMember()</code>. This function should be used together with the function <code>getCodeElement()</code>. Since: Together 5
findNode	<code>RwiElement findNode(Object codeElement)</code> Parameter: code element of interest. Returns: the node for codeElement. Details: This function passes the call to the method <code>com.togethersoft.openapi.rwi.RwiModel.findNode()</code>. This function should be used together with function <code>getCodeElement()</code>. Since: Together 5
findLink	<code>RwiElement findLink(Object codeElement)</code> Parameter: code element of interest. Returns: the link for codeElement. Details: This function passes the call to the method <code>com.togethersoft.openapi.rwi.RwiModel.findLink()</code>. This function should be used together with function <code>getCodeElement()</code>. Since: Together 5
findPackage	<code>RwiElement findPackage(Object codeElement)</code> Parameter: code element of interest. Returns: the package for codeElement. Details: This function passes the call to the method <code>com.togethersoft.openapi.rwi.RwiModel.findPackage()</code>. This function should be used together with the function <code>getCodeElement()</code>. Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
findDocumentedMember	<pre>RwiElement findDocumentedMember(Enumeration codeElements) RwiElement findDocumentedMember(Enumeration codeElements, String subjectSelector)</pre> Parameters: Code elements and subject selectors to search. Returns: found RwiElement that satisfies these conditions: • it is associated with the passed codeElements • it is an RwiMember • it will be presented in all generated documents by its Main Documentation or, if subjectSelector is specified, by its “specific” documentation associated with the passed subjectSelector. Returns null if the requested element doesn't exist in the model . Details: This function should be used together with the function getCodeElements(). It calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findMember()</code>. Since: Together 5
findDocumentedNode	<pre>RwiElement findDocumentedNode(Enumeration codeElements) RwiElement findDocumentedNode(Enumeration codeElements, String subjectSelector)</pre> Parameters: See findDocumentedMember. Returns: See findDocumentedMember. Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findNode()</code>. Since: Together 5
findDocumentedLink	<pre>RwiElement findDocumentedLink(Enumeration codeElements) RwiElement findDocumentedLink(Enumeration codeElements, String subjectSelector)</pre> Parameters: See findDocumentedMember. Returns: See findDocumentedMember. Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findLink()</code>. Since: Together 5
findDocumentedPackage	<pre>RwiElement findDocumentedPackage(Enumeration codeElements) RwiElement findDocumentedPackage(Enumeration codeElements, String subjectSelector)</pre> Parameters: See findDocumentedMember. Returns: See findDocumentedMember. Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findPackage()</code>. Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
findDocBySubjectSelector	String findDocBySubjectSelector(String subjectSelectorList) Parameter: list of subject selectors separated with semicolons. Returns: the path of the first generated document that contains an area marked with one of the specified subject selectors from the list. The path is relative to the documentation's root directory. Subdirectories are delimited with a slash, "/". If no document is found, the function returns an empty string. Details: The function takes the first passed subject selector from the list and checks if there are any generated documents that contain areas marked with this subject selector. If such documents exist, the function returns the one that has been generated first. Otherwise, it iterates to the next subject selector from the list and repeats examination. If all subject selectors are passed and no is document found, the function returns an empty string. Note: A blank subject selector is allowed; it will refer to the Main Documentation of an element. Example: findDocBySubjectSelector("package-summary;summary") returns the first generated document for one of the subject selectors: "package-summary", "summary" Warning: This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition. (See “Defining the frameset structure” on page 359.) Since: Together 5
findDocByTemplate	String findDocByTemplate(String templateList) Parameter: list of template names (without file name extensions) separated with semicolons. Returns: the path to the first generated document produced by one of the specified templates. The path is relative to the documentation's root directory. Subdirectories are delimited with a slash, "/". If no document is found, the function returns an empty string. Details: The function takes the first passed template name and checks if there are documents generated by this template. If such documents exist, it returns the one which has been generated first. Otherwise, it iterates to the next template from the passed list and repeats examination. When all templates are passed and no document has been found, the function returns an empty string. Example: findDocByTemplate("all-classes;all-diagrams") returns the first document produced by one of the templates: "all-classes.tpl" and "all-diagrams.tpl" Warning: This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition. (See “Defining the frameset structure” on page 359.) Since: Together 5

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
checkStockSectionOutput	<code>boolean checkStockSectionOutput(String stockSectionName, RwiElement rwiElement)</code> Parameters: name of a stock section and an RWI element passed to the stock section as the current model element. <code>stockSectionName</code> - name of the Stock Section to be tested. <code>rwiElement</code> - RWI-element passed to the Stock Section as the current model element. Returns: true if the stock section with the given will produce a non-empty output when it is invoked from a stock section call with <code>rwiElement</code> passed to it as the current model element; false otherwise. No actual output is produced from a call to this function. Warning: If no Stock Section with the specified name is found in the template, the function call issues an error message and stop the generator. Example: <pre>checkStockSectionOutput("Included Diagram List", getDGRwiElement("curElement"))</pre> Since: Together 5
getPropertyExt	<code>String getPropertyExt(String propertyName)</code> Parameter: name of a property Returns: value of the property or empty string if the element has no such property. This function gets any element property available in the Document Generator for the metatype to which this element belongs. It includes the properties provided by RWI and the properties calculated only by the Document Generator . (Names of such properties start with %. See “Metamodel types” on page 336.) Example: <pre>rwiElement->getPropertyExt(propertyName)</pre> <code>rwiElement</code> is the element whose property is to be checked. See also: <code>getProperty()</code>
	Utility String Functions
substring	<code>String substring(String str, int beginIndex)</code> <code>String substring(String str, int beginIndex, int endIndex)</code> Parameters: identical to those in the standard Java <code>String.substring()</code> methods. Returns: a new string that is a substring of <code>str</code>.

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
replace	String replace(String str, String oldStr, String newStr) Parameters: three strings. Returns: a new string produced by replacing all occurrences of oldStr in str with newStr. This operation is case sensitive. Example: replace("aBC-BC-bc", "BC", "XY") returns "aXY-XY-bc" Usage: This function is especially helpful for a Call to Template section in which the location of the document generated by the called template must be derived from some properties of the current model element (for example, from the full name of the package where the current element belongs). In this case you can write in the field "Output Directory Expression" an expression such as this: replace(getContainingPackage()->getProperty("\$fullName"), ".", "/") See also: "Creating hypertext links (advanced)" on page 363. Since: Together 5
duplicate	String duplicate(String str, int num) Parameters: a string and a non-negative number. Returns: a new string resulting from duplicating str num times. If num is 0, returns an empty string. Example: duplicate("aBC", 3) returns "aBCaBCaBC" Since: Together 5
length	int length(String str) Parameter: a string. Returns: the number of characters in the string (string length).
str	String str(Numeric N) Parameter: a number Returns: a string representation of the number.
val	Numeric val(String str) Parameter: a string. Returns: the number represented by the string. If conversion is impossible, the function returns 0.
	The following functions, commonly provided in Together formula queries, are also very useful in DG expressions.
getProperty	String getProperty(String rwiPropertyName) Parameter: the name of an RWI-property. Returns: the value of the specified RWI property of the (current) element, or the empty string if the element has no such property. Example call: rwiElement->getProperty(rwiPropertyName) .rwiElement is the element whose property is returned. See also: getPropertyExt()

Table 43 Documentation generator functions (continued)

Function name	Signature and Description
hasProperty	<code>boolean hasProperty(String rwiPropertyName)</code> Parameter: the name of an RWI-property. Returns: true if the (current) element has the specified property; false otherwise. Example call: <pre>rwiElement->hasProperty(rwiPropertyName)</pre> rwiElement is the element whose property is to be checked.
hasPropertyValue	<code>boolean hasPropertyValue (String rwiPropertyName, String value)</code> Parameters: an RWI-property and a possible RWI property value. Returns: true if the (current) element has specified property with the required value; false otherwise. Example call: <pre>rwiElement->hasPropertyValue (rwiPropertyName, value)</pre> rwiElement is the element whose property is to be checked.
if	<code>type if(boolean condition, type value1, type value2)</code> Parameters: any Boolean condition and two values of the same type. The type can be any data type allowed in queries. Returns: If the Boolean condition is true, the function returns value1. If the condition is false, the function returns value2.
getContainingNode	<code>RwiNode getContainingNode()</code> Returns: the RwiNode element that contains the current element. Can be called for RWI member or node current element. Possible call: <code>rwiElement->getContainingNode()</code> Example: The following expression calculates visibility modifier for the class/interface member: <pre>if (hasProperty("\$private"), "private", if (hasProperty("\$protected"), "protected", if (hasProperty("\$public") && !getContainingNode()-> hasProperty("\$interface"),"public", "")))</pre> In this case, the public modifier is printed only when the containing node is not an interface, since all interface members are implicitly public.

p

PART

6

CONTEMPORARY SOFTWARE PRACTICES

- Chapter 22, “Refactoring”
- Chapter 23, “Using the Testing Framework”
- Chapter 24, “Templates and Patterns”
- Chapter 25, “Audits and Metrics”

CHAPTER 22

REFACTORING

Refactoring means rewriting existing source code with the intent of improving its design rather than changing its external behavior. The focus of refactoring is on the structure of the source code, changing the design to make the code easier to understand, maintain, and modify. The primary resource book on refactoring is *Refactoring -- Improving the Design of Existing Code* by Martin Fowler (Addison -Wesley, 1999).

Together provides extensive support for refactoring Java projects. Some of the refactoring operations can be applied to other languages as well. This chapter discusses all available refactoring operations. Chapter topics are:

- “Enabling Refactoring” on page 385
- “Showing code structure” on page 386
- “Moving classes, interfaces, attributes, and operations” on page 388
- “Renaming” on page 390
- “Encapsulating attributes” on page 391
- “Extracting interfaces, superclasses, and operations” on page 392
- “Summary of refactoring commands” on page 396

Enabling Refactoring

Refactoring is an activatable feature in Together. You can activate or deactivate such features from Together’s main menu.

To activate the Refactoring feature:

1. Select **Tools | Activate/Deactivate Features** from the main menu.
2. Go to the **Together Features** tab in the resulting dialog box.
3. Check **Refactoring**.
4. Click **Ok** to activate the checked features and close the dialog box.

When Refactoring is an active feature, you can refactor code in several ways:

- Select **Tools | Refactor** from Together's main menu.
- Select **Refactor** from the right-click menu of the currently selected item in the Explorer or the Designer.
- Select **Refactor** from the Editor's right-click menu. The position of the cursor determines the element to be refactored. (For all commands except *Extract Operation*, the cursor must be in the name of the element to be refactored.)

The *Refactor* command has a submenu that lists the individual refactoring commands. The items on the list vary according to the kind of the selected element. The Tools | Refactor menu contains the full list of refactorings. However, only those refactoring that are appropriate for the currently selected element are enabled.

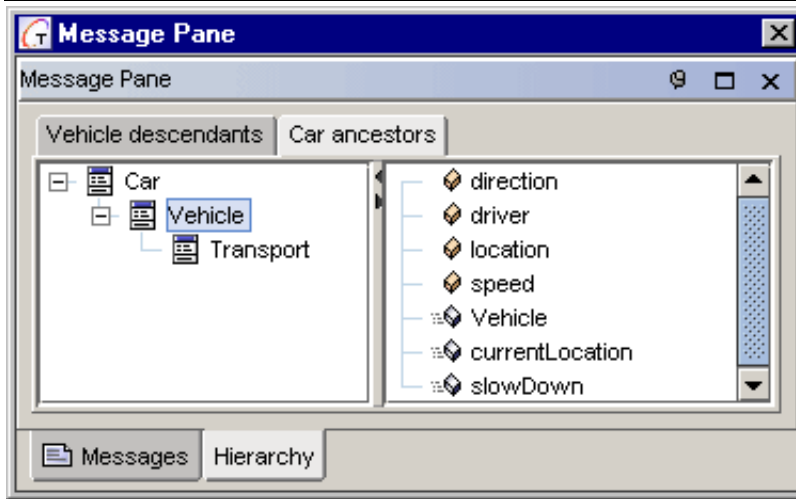
Showing code structure

Together has refactoring commands that enable you to view the part of the class hierarchy that relates to a single class or operation. These “show” commands are available in all languages. They are not refactorings as such, since they do not change the code. However, they can help you investigate the code to determine which refactorings are appropriate.

There are four refactoring commands that show some of the class hierarchy.

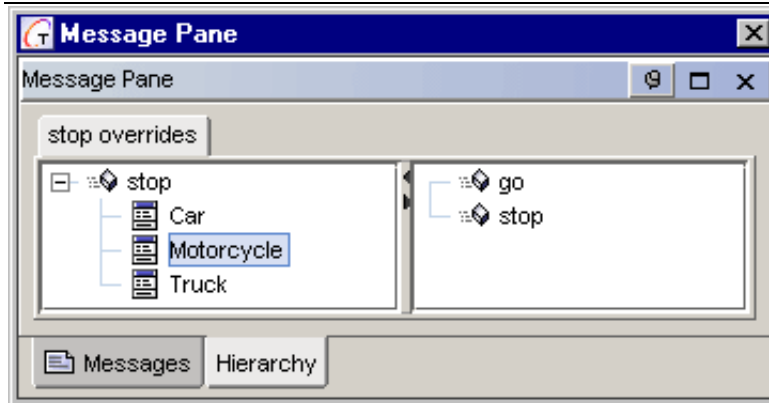
- *Show Ancestors*: (classes and interfaces) shows all of the ancestors of the selected class or interface. Figure 105 is a snapshot of Show Ancestors results.
- *Show Descendants*: (classes and interfaces) shows all of the descendants of the selected class or interface.
- *Show Implementing Classes*: (interfaces only) shows the classes that implement the selected interface. Shows also all derived interfaces and the classes that implement them.
- *Show Overrides*: (operations only) lists all subclasses of the given class that override the selected operation. Subclasses that do not override this operation are grayed.

Figure 105 The result of *Show Ancestors* of Car. The members of the selected class in the left frame (Vehicle) are shown in the right frame.



Together displays the results of a show command in a Hierarchy tab in its Message pane. The tab splits into two frames. The left frame displays the tree structure of the relevant classes, with the selected class or operation at the top. The right frame displays the members of the class on the left that you select. You can move the bar dividing the two frames to resize them.

Figure 106 The result of *Show Overrides* of the operation named stop.



TIP: To invoke a Show command from the Editor, place your cursor on the name of the element (class, interface, or operation) and right click. This brings up a right-click menu that has Refactor | Show <command>.

Moving classes, interfaces, attributes, and operations

You can move classes and interfaces to different packages. You can also move attributes and operations up or down in the class hierarchy.

Moving classes and interfaces

There are two refactoring commands for moving classes and interfaces.

- *Move Class*: moves a class into a different package.
- *Move Interface*: same as *Move Class*, only the move is applied to an interface rather than a class.

To move a class:

1. Apply the move command to the class by selecting the class on the diagram and then doing one of the following:
 - Choose **Refactor | Move Class** from the right-click menu of the class, or
 - Choose **Tools Refactor | Move Class** from the main menu, or
 - Move the cursor into the name of the class in the Editor, then choose **Tools Refactor | Move Class** from the Editor right-click menu.
2. In the resulting dialog box, select the target package. Then click **Next** to review the code or **Finish** to complete the move.
3. If you clicked Next on the previous step, review the code and click **Finish**.

If the selected package already contains a class (or interface) of the same name, the **Finish** button at the bottom of the display is disabled. Clicking the **Next** button shows the error messages. You can navigate back to select a different package by clicking the **Previous** button. Clicking the **Cancel** button cancels the attempted move.

With both *Move Class* and *Move Interface*, Together makes the appropriate changes to all of the project code that uses the class or interface to reflect the new package.

Moving attributes and operations in the class hierarchy

There are four refactoring commands for moving attributes and operations.

- *Push Down Operation*: copies an operation from a superclass to a subclass, deleting the original and optionally changing its visibility. If there are no subclasses, Together displays a warning message.
- *Push Down Attribute*: same as *Push Down Operation*, except the move is applied to an attribute rather than an operation.
- *Pull Up Operation*: copies an operation from a subclass to a superclass, deleting the original and possibly changing its visibility.
- *Pull Up Attribute*: same as *Pull Up Operation*, except the move is applied to an attribute rather than an operation.

TIP: You can move multiple attributes or operations by selecting several at once in the Designer or Explorer and invoking the right-click menu of the entire

selection. If the selection contains both operations and attributes, choose Refactor | Pull Up Member or Refactor | Push Down Member.

Moving attributes and operations is more complicated than moving classes among packages because class members often contain references to each other. Together issues a warning when a *Pull Up Operation* or *Push Down Operation* has the potential for corrupting the syntax if the operation being moved references other class members. You can elect to move the referenced class members as well, or you can ignore them and correct the resulting code manually.

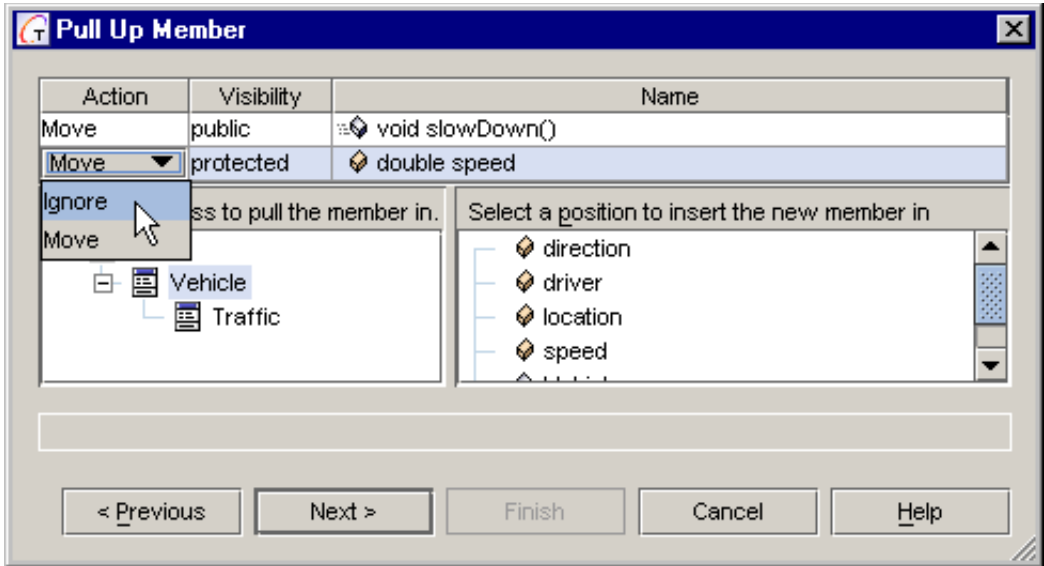
When you pull up or push down members, Together displays a dialog box for specifying additional information that it must know to make the move. The dialog box takes you through a series of pages:

1. Warning page (optional): For moves that could generate syntax errors (such as pushing down an attribute that is referred to elsewhere in a superclass).
2. Class page: For selecting the target classes that are to receive the members to be moved. Together includes referenced class members in the list of members to move. You can change any of the following options:
 - Action for each member listed (Ignore or Move).
 - Visibility for the moved member (private, public, protected, packageLocal).
 - The target classes to receive the moved members (on the left frame).
 - Where among the target classes' members to make the insertion.

Figure 107 shows the display to Pull Up an operation named `slowDown`. Since the operation uses the attribute named `speed`, Together gives the option of moving `speed` as well.

3. Code page: Shows changes resulting from the move. The left frame has the tree of classes whose code will change with the move. When you select a class on the left, the right frame displays the original class code and the changes in the code resulting from the move.
4. Error/warning page (optional): Shows a list of the kinds of potential problems resulting from a move. Together will not move any members that cause errors. Click Cancel to leave the code in its original form or click Previous to change the move options.

Figure 107 *Pull Up Member* dialog box displaying target classes. The operation will be moved but the attribute will be ignored.



Renaming

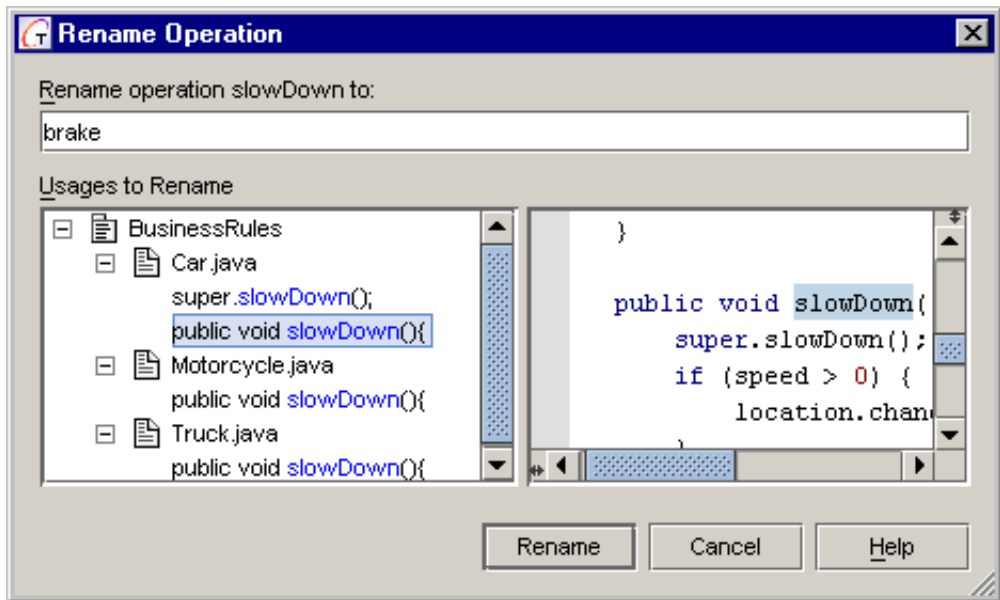
You can rename any code element: class, interface, operation, attribute, parameter, or local variable. Together propagates the name changes to dependant code in the project files.

To rename a local variable or parameter, put your cursor on the variable name in the Editor and right click. Choose Refactor | Rename Variable from the resulting right-click menu. To rename a property, use the right-click menu of the property in the Designer pane or the Explorer pane, or use Tools | Refactor | Rename Property. For any other code element, you can use the Editor, the Designer, the Explorer, or the Tools menu.

Figure 108 shows the dialog box that results from renaming an operation. At the top is a text field for entering the new operation name. The left frame shows all the tree of usages of the old operation name. Selecting a usage brings up the corresponding code in the right frame. Clicking the Rename button closes the dialog box and makes the name change.

NOTE: You can rename packages simply by changing their names in the Designer or Explorer. Together makes the appropriate changes in this case as well. This action does not require Refactoring to be activated.

Figure 108 Renaming an operation



Encapsulating attributes

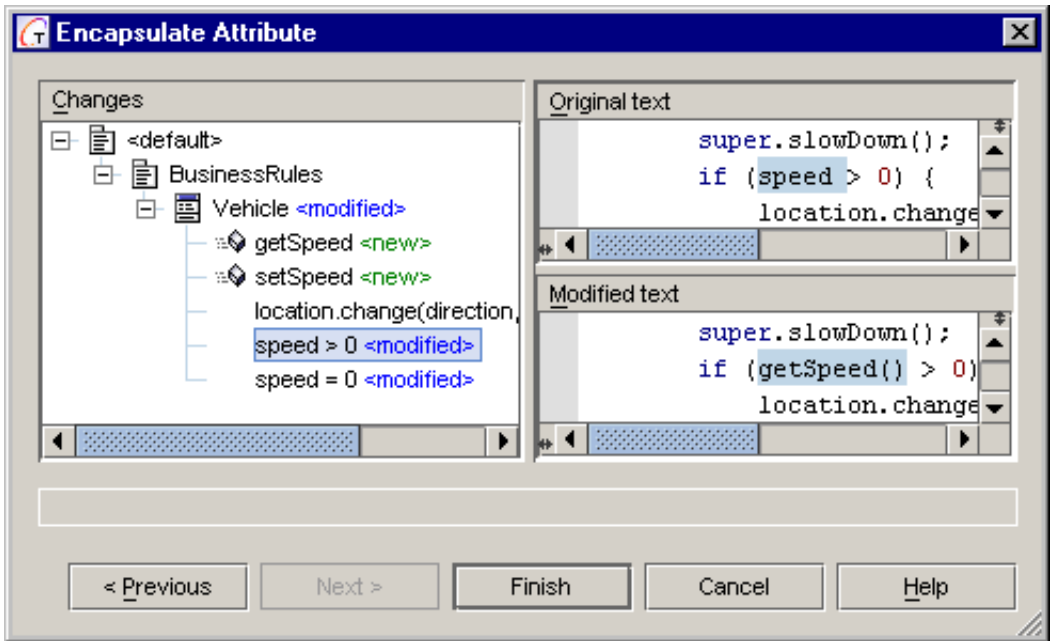
Encapsulating an attribute means hiding it by making it private and providing setters and getters where needed. You can encapsulate an attribute (or multiple attributes) through its right-click menu by selecting Refactor | Encapsulate Attribute. The resulting dialog box has multiple displays. With the first display, you can specify:

- Visibilities: Dropdown list to select the visibilities of the attribute and its getter and setter methods.
- Self Encapsulate: Check box for replacing all references to the attribute by calls to its getter and setter methods (even within the class itself).
- Add Comments: Check box for placing new comments into the modified code:
 - JavaDoc comments for the getter and setter
 - Comments for replacing direct assignments to the attribute by calls to the setter.

After selecting the modifications, click the Next button to preview the results in the code. Click Finish to complete the refactoring.

Figure 109 shows the results of encapsulating an attribute named speed. Self Encapsulating was checked on. The changed lines of code are listed on the left frame. Selecting a line in the left frame displays the resulting text modifications on the right frame.

Figure 109 Code changes with encapsulating an attribute



Extracting interfaces, superclasses, and operations

Together's Refactoring feature enables you to create new interfaces, superclasses, and methods by extracting the information from existing code. You can do extraction in the Designer, the Explorer, the Tools menu, and in the Editor.

Extracting interfaces and superclasses

Extracting a superclass means creating an ancestor class from several operations of a given class or from several different classes. If any selected class already has a superclass, the new superclass is placed in the inheritance chain between the selected class and its existing superclass.

Extracting an interface means creating a new interface from one or more selected classes. Each selected class would implement the interface.

To extract from multiple classes (or methods), you must select them in the Designer or Explorer and use the Tools menu or the right-click menu of the selection. To extract from a single class, you can use the Editor as well, placing the cursor in the class name and then invoking the Editor's right-click menu. In any case, choose Refactor | Extract Interface or Refactor | Extract Superclass from the menu.

TIP: You cannot extract an interface or superclass from multiple classes unless they have either a static member or an operation with the same signature in common.

The dialog boxes for extracting interfaces or superclasses takes you through a sequence of two pages as follows:

1. Initial page: For specifying the details of the new interface or superclass:

- *Name:* Enter the name in the textfield at the top.
- *Package location:* Pick from the display of the project directory.
- *Select Members:* Determine the action and visibility for the list of effected members. The membership of the list varies according to the kind of element to be refactored. If the element is a class, all the class methods are on the list. If the element is a method (attribute), only the method (attribute) is on the list. Clicking the *Action* field for a member displays a dropdown lists with choices of *Ignore*, *Move*, *Copy*, and *Abstract*. Clicking the *Visibility* field for a member displays a dropdown list with choices of *public*, *private*, and *protected*.

NOTE: Together does not allow you to continue to the next page in the *Extract Interface* or *Extract Superclass* dialog if you fail to enter a name or select a package for the new superclass or interface.

2. Review page: For viewing the relevant code. This page has two frames. The frame on the left lists the code elements changed. The frame on the right splits into an Original text frame and a Modified text frame. Clicking on a code element in the left frame highlights the corresponding code in the frame on the right.

Extracting operations

You can extract interfaces and superclasses through the Designer, Explorer, or the Editor. Extracting an operation, however, requires the Editor. To extract an operation, select the code fragment that you want to extract, place your cursor in the selection, and choose Refactoring | Extract Operation from the Editor's right-click menu. Figure 110 shows the resulting dialog box.

In the dialog box, you can specify information about the new operation, including:

- Name
- Visibility (public, private, protected, packageLocal)
- Whether the operation is static, final, or synchronized
- A header comment
- Parameter names and comments. (You cannot change the parameter types.)

The bottom of the dialog box has a frame for previewing the new method.

The Extract Operation command applies only to semantically complete pieces of code. The results of applying the command are as follows:

- Parameters and local variables in the selection become the parameters of the newly created method.
- If a local variable is declared before the selection but modified therein, then the newly created operation uses it as a return value. When the selection is replaced with

the created method, the return value of the method is assigned to this variable, except when this variable is not referred to after the selection.

- Any exceptions thrown in the selection become part of the new method signature.

Figure 110 Extract Operation

Extract Operation

Name and modifiers

Name:

Visibility: Other: ☐ static ☐ final ☐ synchronized

Comment:

Operation parameters

type	name	comment
int	k	local variable
int	alpha	first original parameter
int	beta	

Operation body preview

```
* Generated by Together on Jan 21, 2002  
* @param k local variable  
* @param alpha first original parameter  
*/  
public int aNewOperation(int k, int alpha, int beta) {  
    k += alpha + beta;  
    synchronized(this) {
```

Ok Cancel Help

Restrictions on the selected code to extract from include the following:

- The code cannot contain a return statement of the original method. If you attempt to include a return statement, Together displays an error message.
- The code cannot modify more than a single local variable. Attempts to violate this restriction also results in an error message.

TIP: If the selection of code to extract from is repeated in several locations, it is your responsibility to replace these fragments in the other places with appropriate method calls.

Figure 110 shows the *Extract Operation* dialog box resulting from refactoring this code:

```
int someOperation(int a, int b ) {  
    int k = 4;  
    k += a + b;  
    synchronized(this) {  
        if( attr < 0 ) {  
            attr = k;  
        }  
    }  
    return k;  
}
```

The resulting code is as follows:

```
int someOperation(int a, int b ) {  
    int k = 4;  
    k = aNewOperation(k, a, b);  
    return k;  
}  
  
/**  
 * Generated by Together on Jan 21, 2002  
 * @ param k local variable  
 * @param alpha first original parameter  
 */  
public int aNewOperation(int k, int alpha, int beta){  
    k += alpha + beta;  
    synchronized(this) {  
        if (attr < 0) {  
            attr = k;  
        }  
    }  
    return k;  
}
```

Summary of refactoring commands

Table 44 lists all of the refactoring commands. You can invoke most commands through the Designer, Explorer, and Editor right-click menus as well as the Tools menu. However, you cannot use the Editor to apply commands to multiple objects..

Table 44 Refactoring commands

Command	Applies To	Editor	Designer/ Explorer	Comments
Encapsulate Attribute	One or more attributes	Yes	Yes	
Extract Interface	One class	Yes	Yes	All class members are listed. Default action is 'Ignore.'
	Several classes	No	Yes	Only members with equal signatures are listed. Default action is 'Abstract.'
	Several members of a single class	No	Yes	Selected members are listed. Default action is 'Abstract.'
Extract Operation	Selection in editor	Yes	No	
Extract SuperClass	One class or interface	Yes	Yes	All class members are listed. Default action is 'Ignore.'
	Several classes or interfaces	No	Yes	Only members with equal signatures are listed. Default action is 'Move.'
	Several members of a single class or interface	No	Yes	Selected members are listed. Default action is 'Move.'
Move Class	One or more classes	Yes	Yes	
Pull Up Member	One or more members of a single class	Yes	Yes	
Push Down Member	One or more members of a single class	Yes	Yes	
Rename Attribute	One attribute	Yes	Yes	
Rename Class	One class	Yes	Yes	
Rename Interface	One interface	Yes	Yes	
Rename Operation	One operation	Yes	Yes	
Rename Property	One property	No	Yes	
Rename Variable	One parameter or local variable	Yes	Yes	
Show Ancestors	One or more classes or interfaces	Yes	Yes	Available for all languages.
Show Descendants	One or more classes or interfaces	Yes	Yes	Available for all languages.
Show Implementing	One or more classes or interfaces	Yes	Yes	Available for all languages.
Show Overrides	One operation	Yes	Yes	Available for Java.

USING THE TESTING FRAMEWORK

This chapter provides instructions for creating and running tests using Together's testing framework. The testing framework enables you to develop both unit tests for source code and visual tests for user interface components.

This chapter includes the following topics:

- “Overview of the testing framework” on page 397
- “Before you begin” on page 398
- “Creating a test project” on page 400
- “Configuring options” on page 402
- “Developing unit tests for source code” on page 405
- “Working with unit tests” on page 409
- “Developing visual tests for a user interface” on page 410
- “Working with visual tests” on page 415
- “Setting-up the bootstrap loader for a test server” on page 418

Overview of the testing framework

Together features a robust testing framework that supports unit testing of source code and visual testing of a user interface. The testing framework allows you to collect details of the tests you plan to run in a structured “test tree.” From the test tree, you can record user interactions and data input of applications, and then replay the actions at a later date.

For unit testing, Together supports JUnit. JUnit is a testing framework for building and executing unit tests in Java. With JUnit, you can create unit tests for incremental testing, and then setup the testing framework to execute continuous builds each time you compile or deploy your application. At the time of this release, JUnit is open source software. For more information specific to JUnit, visit the JUnit website at <http://www.junit.org/>.

For visual testing, Together provides a visual recorder that records and plays scripts. Visual scripts can include tests to analyze Swing and AWT components.

Testing often includes a variety of activities such as manual testing, automated testing of source code, and running visual scripts. A *test plan* is a script that describes such actions, and controls their sequence. To see how Together represents a test plan, see Figure 111, “Tests tab showing a test plan” on page 400.

A test plan defines *test assets* and *test results* as follows.

- **Test assets** include:
 - **TestStep:** The minimal step of the testing process; for example, a JUnit test case or visual script.
 - **TestSuite:** A script that combines test steps (or other test suites) into a single entity, each constituting a separate testing activity. A test suite executes test steps and test suites at the same time. Test suites contain logic. In addition, you can use test steps (or test suites) more than once within a test suite.
 - **TestData:** Any data required by a TestStep. Test data is required by certain test categories, such as visual scripts. Examples include the user interface elements map for a visual script and the visual script itself.
 - **TestCollection:** A directory containing other test assets, as well as other test collections. Unlike a test suite, a test collection does not contain logic and can not be executed.
- **Test result** is produced by the test runner and describes the outcome of a particular test step, or all the test steps of a test suite.

NOTE: All test assets in the test plan exist as separate resources or files. Access each test asset by a URL or unique identifier (such as a reference ID).

Additional terms used to describe the testing framework include:

- **TestRunner:** An abstraction that identifies the object responsible for executing a TestStep or TestSuite.
- **TestGenerator:** Similar to TestRunner, an abstraction that identifies the object responsible for populating TestData for a TestStep.
- **TestResult:** A resource that contains the results of executing a test, including outcome and details. An example is a URL to the resource with error message or exception stack trace.

Before you begin

Before you can use the testing framework, you need to activate the testing framework module. In addition, it may be helpful to review the sample test project provided for CashSales.

NOTE: If you plan to use a separate test server, you need to follow the instructions in “Setting-up the bootstrap loader for a test server” on page 418 in addition to the tasks explained in this section.

Activating the testing framework module

To use the instructions in this chapter, active the testing framework module. To activate the module, go to **Tools | Activate/Deactivate Features** and then check the box under Activated for Testing Framework.

Accessing the sample test project *CashSales*

Together includes a sample Java application called CashSales. The CashSales example also includes a sample test project. The CashSales test project is used to illustrate the instructions provided in this chapter.

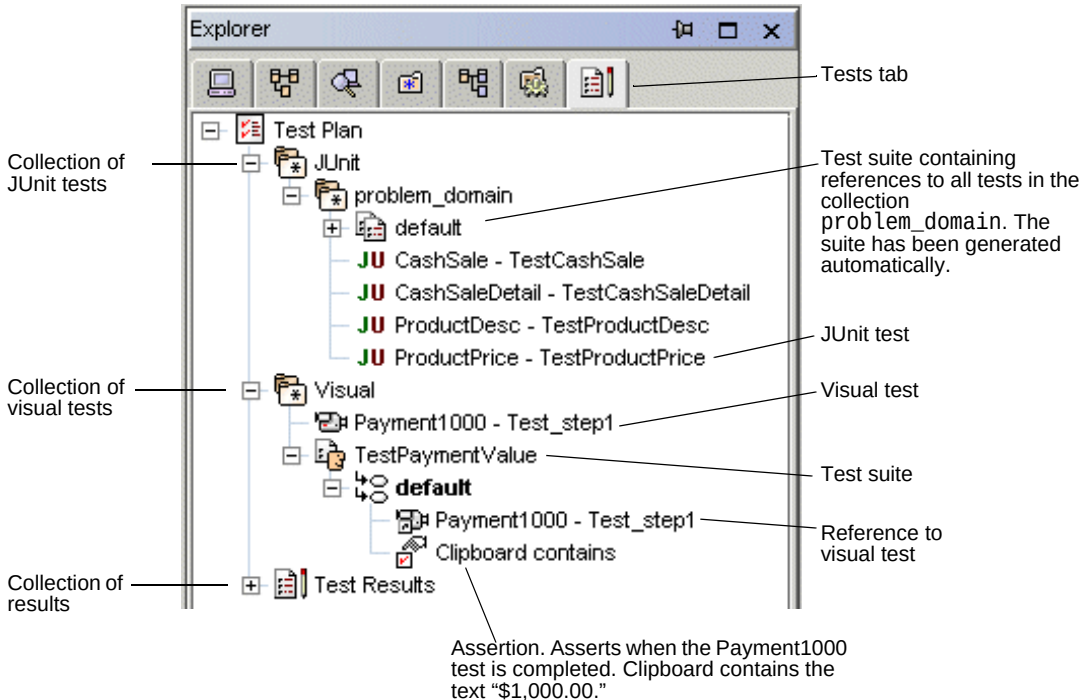
To access the sample test project for CashSales, follow these steps:

1. From the Directory tab of the Explorer pane, select **Samples | java | CashSales.Test | TestCases**.
2. Select **CashSalesTests.tpr**. As this project opens, the Tests tab appears in the Explorer pane.

IMPORTANT: If the Tests tab does not appear, make sure that you are running the testing module. See “Activating the testing framework module” on page 399 for more information.

3. Select the Tests tab to view the test plan as shown in Figure 111.

Figure 111 Tests tab showing a test plan



Creating a test project

The first step to developing unit tests or visual scripts in Together is to create a test project. This test project is specific to the project for which you plan to create tests.

As an example, the instructions in this section use the CashSales sample project. See “Accessing the sample test project CashSales” on page 399 for more information.

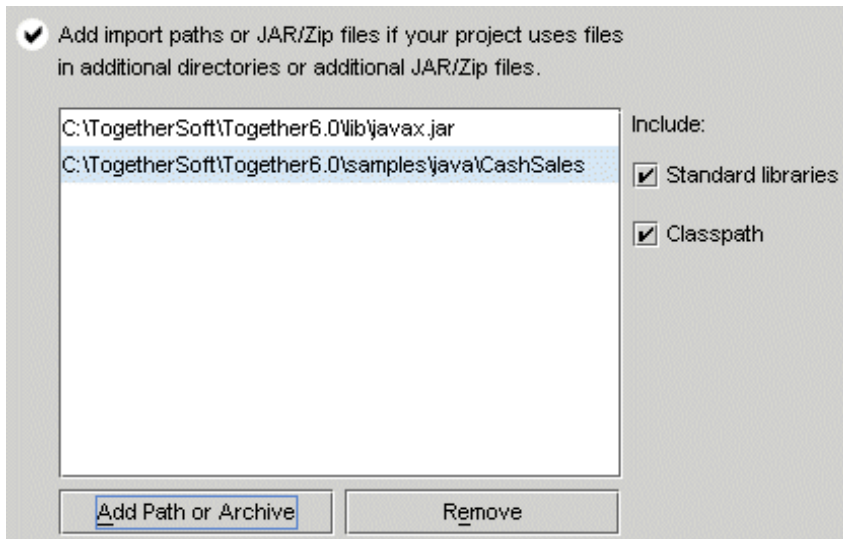
To create a test project, follow these steps:

1. Create two directories for the project that you plan to test:
 - A directory that will contain test code, diagram files, and the project file such as:
TGH\samples\java\CashSales.Test\TestCases
 - A directory that will contain the test description and result files generated by the testing framework. To prevent automatic parsing of the directory contents, this directory must not be a parent or child of the directory containing the test code and project file such as:
TGH\samples\java\CashSales.Test\TestPlan
2. Create a new project using **File | New Project Expert**. (For additional information, see “Using the New Project Expert” on page 96.)

TIP: Locate the project in the TestCase subdirectory below the project root directory. For example, the location for CashSales is TGH\samples\java\CashSales.Test\TestCases.

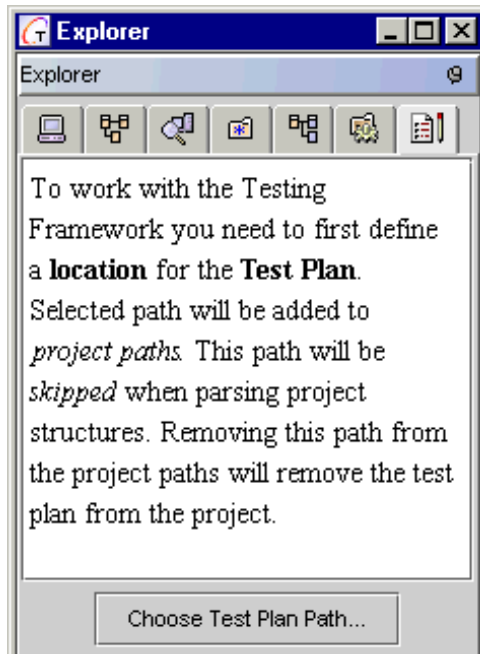
3. Add path(s) to the code being tested when the New Project Expert displays the screen shown in Figure 112.

Figure 112 Specifying paths of code to be tested in New Project Expert



4. Continue providing the information requested by the New Project Expert, and then click **Finish** to create a new project.
5. After Together generates the new project, click on the **Tests** tab of the Explorer pane. The message shown in Figure 113 appears.

Figure 113 Tests tab for a new project



6. Click Choose Test Plan Path and select the project path for the test plan directory.

After completing these steps, you can proceed to “Developing unit tests for source code” on page 405 or “Developing visual tests for a user interface” on page 410.

Configuring options

Together allows you to set options for the testing framework at the default and project levels. You can configure options for visual steps, test suites, JUnit steps, the unit test builder, as well as the testing framework itself.

To access the configuration options, follow these steps:

1. From the Tools menu, select **Options | Default Level (or Project Level)**
2. Expand **Testing Framework** to show:
 - Visual step
 - Test Suite
 - JUnit Step
 - Building Unit Tests

Testing framework options

The options that follow appear when you select **Testing Framework** in the Options dialog:

- **Testing support enabled:** Enables and disables testing support in the project **Explorer** pane. When checked, the **Tests** tab appears in the pane.
- **Confirm test server launch:** Enables and disables prompting for confirmation before the test server launches.
- **Confirm delete:** Enables and disables deletion confirmation prompts.
- **Call make before running tests:** Enables and disables automatic code builds before the running test executes.
- **Confirm before creating test plan:** Enables and disables prompting for confirmation before test plans are created.
- **Result-to-HTML stylesheet:** Defines a custom stylesheet to be applied to the test result output. Custom stylesheets can be used to generate a user-defined, custom HTML report of test results. The file specified must be an XSL stylesheet.

Options for visual steps

The options that follow appear when you select **Testing Framework | Visual step** from the Options dialog:

- **Default package:** Specifies the default package name for visual test steps. The default is `test.visual`.
- **Delay script by:** Specifies a delay time in milliseconds between each visual action.
- **Component timeout:** Specifies the time within which a screen component is expected to appear. After the specified number of seconds elapse, a component is considered to have not been found. The default is ten seconds.
- **Stop application:** Enables and disables automatic stopping of the tested application at the conclusion of the test.

Options for test suites

The following options appear when you select **Testing Framework | Test suite** from the Options dialog:

- **Default target name:** Specifies the default suite name. This name is used when creating a new suite and as the expected default target for existing suites.
- **Stop test server:** Enables and disables automatic termination of the test server after the completion of a test suite's execution. The default is to stop the test server after execution of a test suite is completed.
- **Recurse default suites:** Enables and disables the creation and referencing of default suites in child collections. When enabled, default suites are created recursively in children collections and referenced in the parent's default suite. The

default is to create and reference default suites in child collections. For more information, see “Creating and configuring collection suites” on page 407.

Options for JUnit steps

The following options appear when you select **Testing Framework | JUnit Step** from the Options dialog:

- **Stop test server:** Enables and disables automatic termination of the test server after the completion of a test suite’s execution. The default is to not stop the test server after execution of a test suite is completed.
- **Create test step for new test case:** Enables and disables automatic creation of a test step when a new test case is created. The default is to create a test step whenever a new test case is created.

Options for building unit tests

The testing framework includes a *unit test builder* that generates unit tests. You can configure the unit test builder to customize test case creation for supported unit test families (JUnit, JUnitX, Cactus, and HttpUnit).

Using filter options, advanced users can limit generation of test cases by package, class, and method. In addition, locations of additional required .jar files can also be specified. See “Setting-up filter options for the test builder” on page 405 for an example of how to use these options.

Support for JUnit

JUnit is a testing framework for building and executing unit tests in Java. For more information about JUnit and the testing framework implemented in Together, see “Overview of the testing framework” on page 397. To set JUnit options, choose **Testing Framework | Building Unit Tests | JUnit Family**.

Support for JUnitX

JUnitX extends the functionality of JUnit by enabling access to private and protected classes, methods and packages during testing. As such, you can configure testing framework options for package filters, class filters, and method filters used by private test cases.

To set JUnitX options, choose **Testing Framework | Building Unit Tests | JUnitX Family**. For more information about JUnitX, visit the JUnit website at <http://www.junit.org/index.htm>.

Support for Cactus

Cactus is a framework for unit testing servlets, EJBs, taglibs and other server-side Java code. For installation and integration instructions, refer to the *Getting Started* guide for Cactus available at <http://jakarta.apache.org/cactus/>.

At the time of this release, testing framework support of Cactus is limited to generating a test skeleton. To set Cactus options in Together, choose **Testing Framework | Building Unit Tests | Cactus Family**.

Support for HttpUnit

HttpUnit enables you to access web sites without a browser. When used with JUnit, HttpUnit provides for automated unit testing of web sites. For more information, visit the HttpUnit web site at <http://www.httpunit.org/>.

At the time of this release, testing framework support of HttpUnit is limited to generating a test skeleton. To set HttpUnit options in Together, choose **Testing Framework** | **Building Unit Tests** | **HttpUnit Family**.

Developing unit tests for source code

The information in this section assumes you have completed the instructions for “Creating a test project” on page 400. Creating unit tests involves sequentially completing the following tasks, as described in this section:

1. Setting-up filter options for the test builder
2. Generating stubs for test cases
3. Writing test cases
4. Creating a test suite

In addition to the tasks listed above, this section provides instructions for the following features:

- “Importing JUnit tests” on page 407
- “Using templates to create test cases” on page 408

Setting-up filter options for the test builder

The test builder generates stubs based on filter options specified for packages, classes, and methods. The default values for filter options exclude certain packages, classes, and methods. For example, the default filters exclude private, protected, and set/get methods from stub generation. To test these types of methods, change the filter options accordingly.

To set-up options for the test builder, follow these steps:

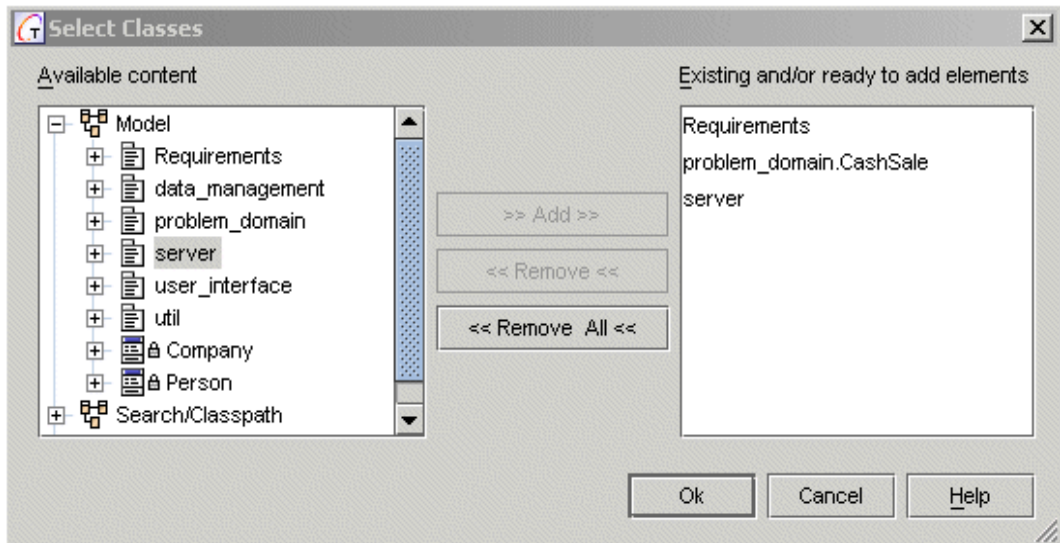
1. From the Tools menu, choose **Tools** | **Options** | **Project Level**.
2. From the tree-view of Project Options, expand **Testing Framework** | **Building Unit Tests** | **JUnit Family** | **Default TestCase**.
3. Select **Package Filters** and configure options as appropriate.
4. Select **Class Filters** and configure options as appropriate.
5. Select **Method Filters** and configure options as appropriate.
6. Click **Apply** and then **OK**.

Generating stubs for test cases

Before you can create test cases, you need to create stubs by following these steps:

1. Open your test project. (See “Creating a test project” on page 400 if you have not created a test project).
2. From the Explorer pane, select the **Tests** tab.
3. From the tree, right-click on **Test Plan** and select **New | JUnit Step | Default TestCase**.
4. From the tree-view for Select Classes, expand **Model** to view the accessible classes.
5. Select the classes that require stubs, adding them to the right hand column, as shown in Figure 114.
6. Click **OK**.

Figure 114 Select Classes dialog



Writing test cases

The default behavior of a generated test stub is for each method test to fail. This code must be replaced by code implementing the actual, intended method test.

These instructions assume that you have followed the steps for “Setting-up filter options for the test builder” on page 405 and “Generating stubs for test cases” on page 406.

To write test cases, edit the code by following these steps:

1. Open a test project.
2. From the Explorer pane, select the **Tests** tab.

3. From the tree-view, right-click on the name of a test step and select **Edit JUnit Test** or double-click on the name of the test step.
4. Using the Editor pane, replace each test method body with the desired test code.

For further documentation on JUnit, see <http://www.junit.org/> or <http://junit.sourceforge.net/>.

Creating a test suite

A test suite is used to organize individual unit test steps into a logical collection that can be run as a group. These instructions assume that you have followed the steps for “Writing test cases” on page 406.

To create a test suite, follow these steps:

1. Open a test project.
2. From the Explorer pane, select the **Tests** tab.
3. From the tree-view, right-click on **Test Plan** and select **New | Suite**. The new node Test Suite appears in the tree.
4. From the tree-view, expand **Test Suite** and right-click on the **default** target of the new suite and select **New | Step Reference**.
5. From the Select Test Steps dialog, select one or more unit test steps.
6. Click **OK**.

Importing JUnit tests

To import a JUnit test, follow these steps:

1. Open a test project.
2. From the Explorer pane, select the **Tests** tab.
3. Right-click on **Test Plan**  (or any test collection ) and choose **Import JUnit Test**.
4. Use the **Import JUnit Test** dialog to add elements from available content
5. Click **Ok**.

Creating and configuring collection suites

In Together, a *collection suite* is a specific type of test suite that enables you to run all the tests contained in a collection. You can configure collection suites to recursively include other collection suites from sub-collections, including all the tests specified for a collection.

To create a collection suite, follow these steps:

1. Open your test project.
2. From the Explorer pane, select the **Tests** tab.

3. Right-click on **Test Plan** and choose **Create Collection Suite**. Together adds the Test Plan Suite node under the test plan.

To configure a collection suite to recursively include other suites, follow these steps:

1. From the Tools menu, choose **Options | Default Level** (or **Project Level**).
2. From the Default Options dialog, select **Testing Framework | Test Suite**.
3. Check the *Recurse default suites* option.
4. Click **Apply** and then **OK**.

Using templates to create test cases

When Together creates unit tests, it uses the unit test builder plug-in. The configuration file of the unit test builder contains references to templates that it uses to generate JUnit tests. The configuration file is `unittestbuilderconfig.xml` located in `$TGH$\modules\com\togethersoftware\modules\testingframework\plugin / \testbuilder\config\`.

You can edit the templates in the configuration file, or add references to your own templates. For the format of `unittestbuilderconfig.xml`, refer to the DTD file `unittestbuilderconfig.dtd` located in the same directory.

Elements of the unit test builder configuration file

The configuration file uses the elements listed in Table 45.

Table 45 Elements used by `unittestbuilderconfig.xml`

This element...	...refers to this type of template
<code>testClass</code>	Class
<code>constructor</code>	Method
<code>method</code>	Method
<code>testMethod</code>	Method
<code>fields</code>	Attribute

The following is an excerpt of the configurations section of the file, showing the elements listed in Table 45:

```
<configurations>
  <!--
  testcase configuration
  -->
  <testCaseConfiguration name="default">
    <testClass templateName="JUnit/DefaultTestCase"/>
    <fields/>
    <constructors>
      <constructor templateName="JUnit/Default_Constructor"/>
    </constructors>
```



```

    <auxiliaryMethods>
        <method templateName="JUnit/Default_SetUpMethod"/>
        <method templateName="JUnit/Default_TearDownMethod"/>
        <!--
        <method templateName="JUnit/Default_TestCase_MainMethod"/>
        -->
    </auxiliaryMethods>
    <testMethods>
        <testMethod kind="template" templateName="JUnit \
        /Default_TestMethod"/>
    </testMethods>
</testCaseConfiguration>

.....
</configurations>

```

Using the XML editor

To edit the unit test builder configuration file, `unittestbuilderconfig.xml`, use Together's XML editor.

To open the configuration file in the XML editor, follow these steps:

1. Activate the XML editor:
 - From the Tools menu, choose **Activate/Deactivate Features**.
 - From the Together Features tab, check **XML Support**.
2. In the Explorer pane, click the **Directory** tab and choose:


```
$TGH$\modules\com\togethersoftware\modules\testingframework\plugin /
\testbuilder\config\unittestbuilderconfig.xml
```
3. The XML editor tab  appears in the Explorer pane. Click the tab to view the file. See Chapter 14, "Working with Code" for instructions on using the XML editor.

Working with unit tests

The instructions that follow apply to your unit tests. As an example, the CashSales test project is used in this section.

Running JUnit tests

As an example, this section uses the sample test *ProductPrice* which is one of the JUnit tests included in the CashSales test project.

To run a JUnit test, follow these steps:

1. From the Explorer pane, select the **Tests** tab.

2. From Test Plan, right-click on a JUnit test, marked by the JUnit **JU** icon. You may need to expand a test collection and test suite in order to locate a JUnit test. (If you are using the CashSales example, go to **JUnit | problem_domain** and select **ProductPrice - TestProductPrice**).
3. Select **Run Test** from the right-click menu.
4. The result of the test appears in the Message pane.

Running a collection of tests

JUnit allows you to group unit tests as a test suite. To run a collection of tests, use your cursor to select the test suite and then choose **Run Suite** from the right-click menu.

Generating a report in HTML

After running unit tests, you can generate a report showing the test results. The instructions that follow use the test results that Together generates after running the sample test for ProductPrice, as described in the previous section.

To generate a report, follow these steps:

1. From the Explorer pane, select the **Tests** tab.
2. From **Test Plan | Test Results**, use your cursor to select the **ProductPrice** test result.
3. Right-click on **ProductPrice** and then select **Generate HTML**. The report appears in the default HTML viewer.

Together also provides you with the option of applying your own stylesheet (.xsl file) to a generated report. By default, the stylesheet used is `xresult.xsl` located in `$TGH$/modules//com/togethersoft/modules/testingframework/plugin/xsl`.

To apply your own stylesheet, follow these steps:

1. From the Tools menu, select **Options | Default Level (or Project Level)**
2. Select Testing Framework from the tree-view, and then select a stylesheet for the *Result-to-HTML stylesheet* option.
3. Click **Apply** and then **Ok**.

Developing visual tests for a user interface

The instructions in this section assume that you have completed the instructions in “Creating a test project” on page 400.

Creating a test collection

A test collection is a directory that contains test assets and other test collections. Unlike a test suite, a test collection does not contain logic and can not be executed.

If you plan to have multiple tests for a project, (for example, unit tests and visual tests), you can create separate collections of tests. Figure 111, “Tests tab showing a test plan” on page 400 illustrates the organization of unit test collections and visual test collections.

To create a test collection, follow these steps:

1. Open a test project.
2. From the Explorer pane, select the **Tests** tab.
3. From the tree-view, right-click on **Test Plan** and select **New | Collection**. The new node Test Collection appears in the tree.

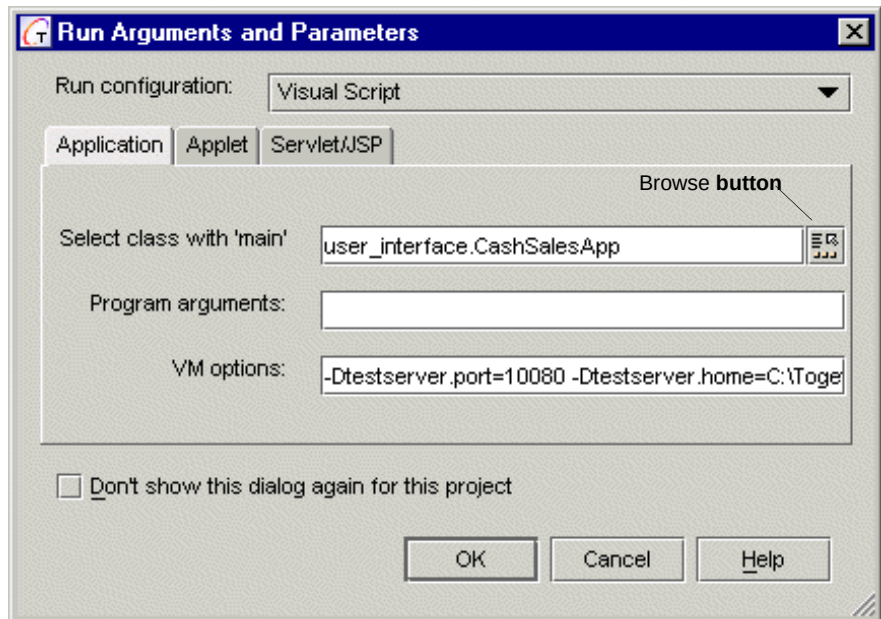
Recording a visual script

The testing framework includes a recorder (as shown in Figure 116) for creating visual scripts. This section provides instructions for using the recorder to create a visual script. To set options for recording a script, see “Setting options for recording a visual script” on page 413.

To record a visual script, follow these steps:

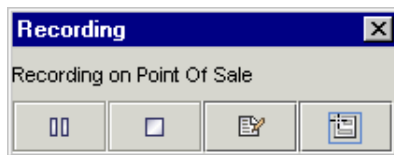
1. Open a test project. If you have not created a test project, see “Creating a test project” on page 400.
2. From the Explorer pane, select the **Tests** tab.
3. From the tree-view, right-click on **Test Plan** and select **New | Visual Step | Record**.
4. From the Run menu, choose **Run** to invoke the Run Arguments and Parameters dialog, and click on the browse button (as shown in Figure 115) to bring up the Select main class dialog.

Figure 115 Run Arguments and Parameters dialog



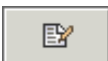
5. From the Select main class dialog, select the application or applet to run and then click **OK**.
6. From the Run Arguments and Parameters dialog, click **OK** to invoke the Recording window as shown in Figure 116.

Figure 116 Recording window



7. Use the Recording window to control the visual script session, including inserting an assertion or a component identification statement, as follows.

To insert an assertion, follow these steps:

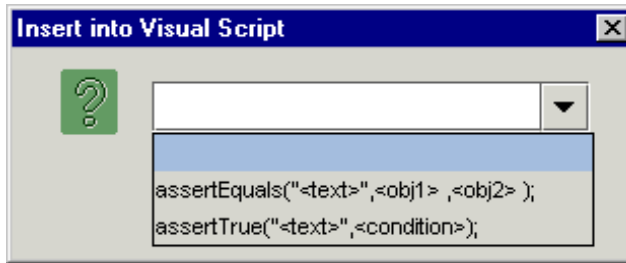
1. Click the assert button  on the Recording window.
2. From the Insert into Visual Script dialog shown in Figure 117, select a pre-defined assertion statement or enter a comment to be placed in the script.

If necessary, edit inserted statements and comments by using the instructions in "Editing a visual test" on page 414. For more information about assertions, see "Creating an assertion" on page 417.

NOTE: If you are using a clipboard assertion, you can set the contents of the clipboard when recording a visual script by using keyboard shortcuts (such as Ctrl+C depending on your operating system) to copy the assertion from the user interface to the clipboard.

3. Click **OK**.

Figure 117 Insert into Visual Script dialog



To insert a component identification statement in the script, follow these steps:

1. Click the component identification button  on the Recording window.

NOTE: A component identification statement returns the object reference for the selected user interface component.

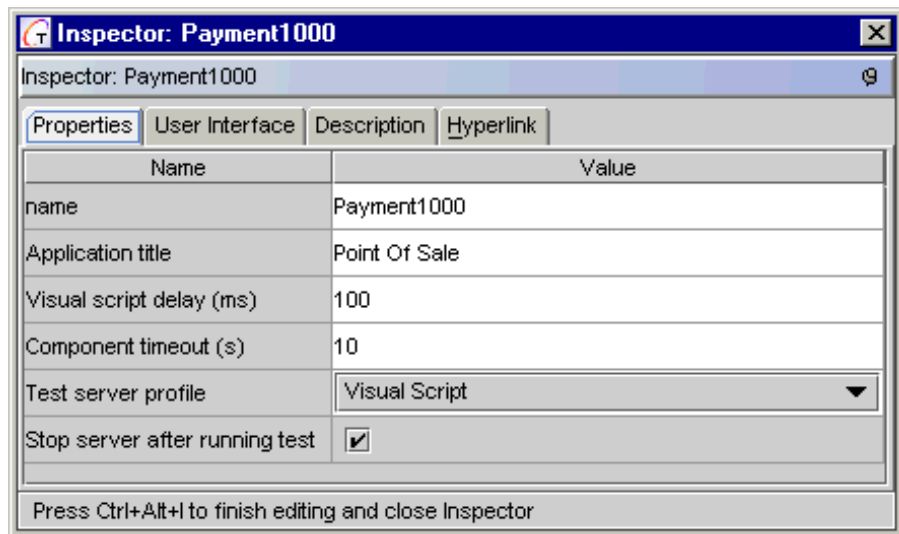
2. Use the cross-hair to select a component.
3. Click **OK**.
4. Press the stop button  on the Recording window.

Setting options for recording a visual script

To set options for recording a visual script, follow these steps:

1. From the Explorer pane, select the **Tests** tab.
2. Expand **Test Plan | Visual**.
3. Right-click on a visual test, as marked by the movie-camera  icon, and choose **Properties**. The inspector for the visual script appears, as shown in Figure 118.
4. In the inspector, enter a value (in milliseconds) for *Visual script delay*.
5. Enter a value (in seconds) for *Component timeout*
6. Set other options as required.

Figure 118 Inspector for a visual script



Editing a visual test

To edit a visual script in Together, you can edit source code from the Editor pane.

To edit a visual script, follow these steps:

1. From the Explorer pane, select the **Tests** tab.
2. Right-click on **Test Plan** and select **Edit Visual Test**.
3. From the Editor pane, modify the script.

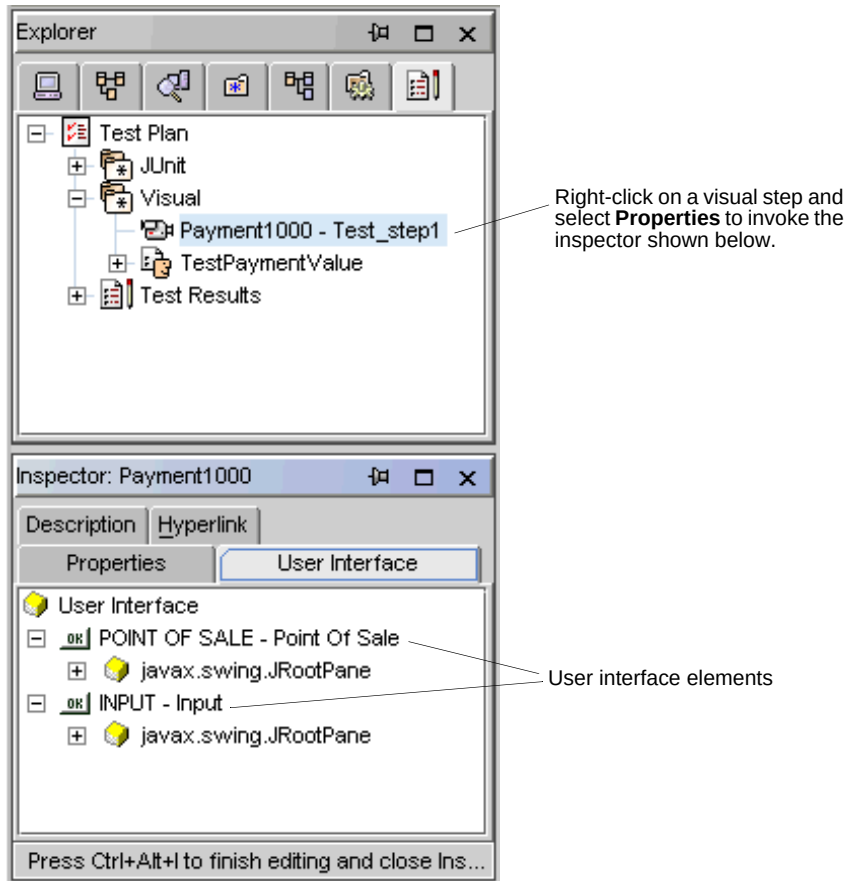
You can also change the user interface elements referenced by a visual test.

To change the user interface elements referenced by a visual step, follow these steps:

NOTE: The instructions that follow use the XML editor and require that you activate XML support. See “Using the XML editor” on page 409 if you need to activate the XML editor feature.

1. From the Explorer pane, right-click on a visual step and choose **Properties**. The inspector for the visual test appears.
2. From the inspector, choose the **User Interface** tab. As shown in Figure 119, the OK icon indicates the user interface elements considered “tagged” and referenced in the visual script; for example, “moveMouse (“tagged”,0,0)”. These user interface elements and their respective values are defined by the element interface_map of the visual test asset.
3. Right-click on the visual step in the Explorer pane and choose **Edit Test Asset**. Together opens the source code for the user interface elements in the XML editor.
4. Using the XML editor, you can change the values for <tag> elements or add your own <tag> elements under the <ui_element> element.

Figure 119 Interface tab showing user interface elements



Working with visual tests

The instructions that follow apply to your own visual tests. As an example, the CashSales test project is used in this section.

Running visual tests

As an example, this section uses the sample visual test *Payment1000* which is one of the tests included in the CashSales test project.

To run a visual test, follow these steps:

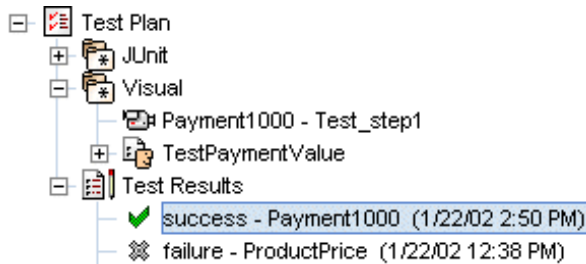
1. From the Explorer pane, select the **Tests** tab.
2. Expand **Test Plan | Visual**.

3. Right-click on a visual test, as marked by the movie-camera  icon and select **Play Script**. (If you are using the CashSales example, select **Payment1000 - Test_step1**). The Run Arguments and Parameters dialog appears.
4. Enter information for *Select class with 'main'*. (For the CashSales examples, enter **user_interface.CashSalesApp**).

The Payment1000 test executes. When finished, Test Results should indicate success status, as shown in Figure 120.

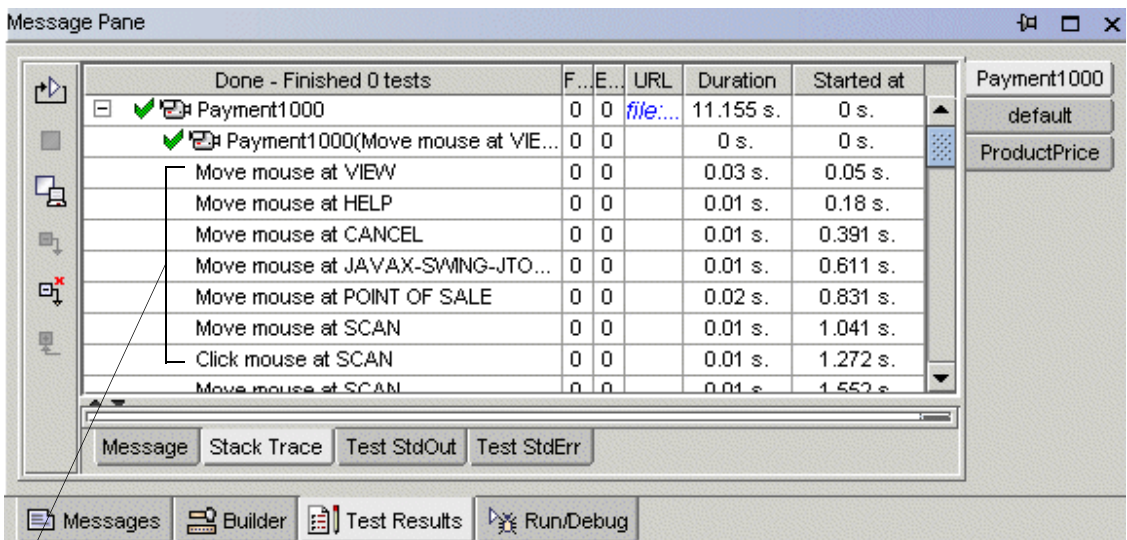
NOTE: When a visual script is played for the first time, the run configuration dialog, Run Arguments and Parameters appears.

Figure 120 Results for visual test indicating success



In addition, note that after the test completes, the Message Pane lists each step that comprises the visual test, and provides test results for each step, as shown in Figure 121.

Figure 121 Results for each step of the visual test, as shown in the Message pane



Steps for the visual test script.

Creating an assertion

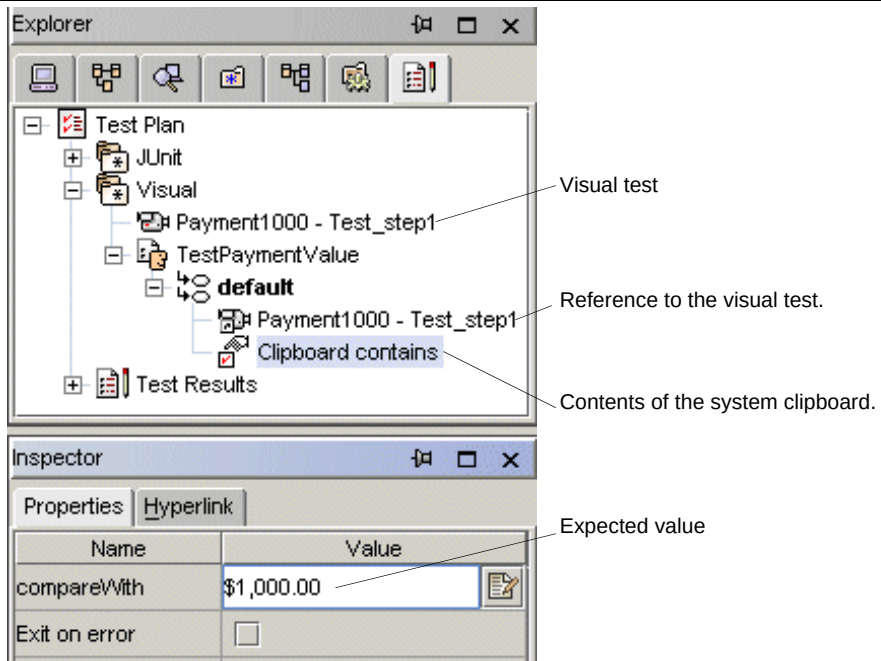
Together enables you to use four types of assertions:

- A file that does not exist
- The contents of a clipboard
- A title of an application that matches a given value
- A file that does exist

The visual script of the CashSales test project uses a clipboard assertion. Figure 122 shows that the visual test includes both the clipboard with the assertion, and the script that uses the assertion.

If you plan on using a clipboard assertion, note that at the time of this release, Together supports only text contents of a clipboard. For information on how to include an assertion while recording a visual script, see “Recording a visual script” on page 411.

Figure 122 CashSales test suite showing clipboard and visual script

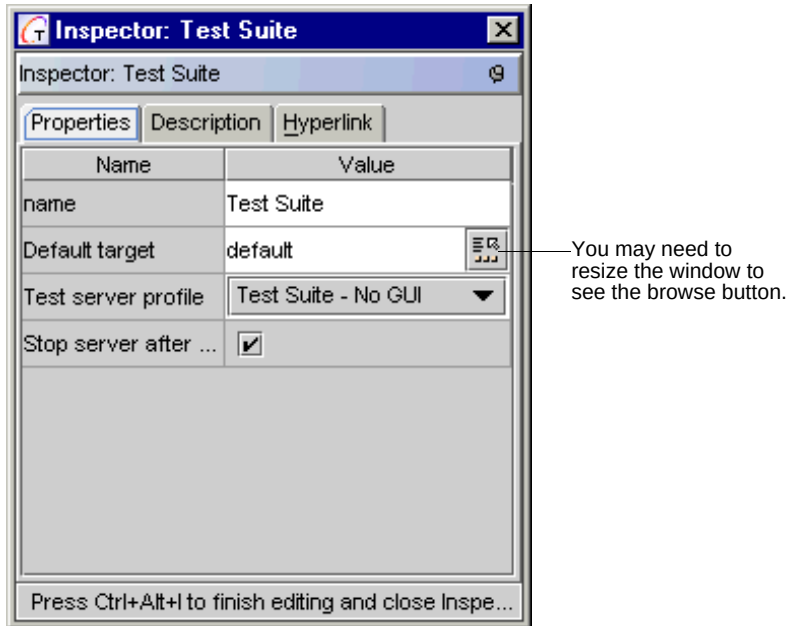


To create an assertion, follow these steps:

1. From the Tests tab of the Explorer Pane, right-click on **Test Plan** and choose **New | Suite** to create a new test suite.
2. Right-click on the new test suite and choose **Properties** to invoke the test suite inspector. Select a default target for the new test suite, as shown in Figure 123.
3. On the Tests tab, right-click on the default and choose **New | Step Reference**.

4. From the Select Test Steps dialog, select an existing visual test step. A node to the step reference is created under the suite.
5. On the Tests tab, right-click on the step reference (that is, the node created in the previous step) and choose **New | Assertion**.
6. From the Create Assertion dialog, select the type of assertion and set the associated properties.
7. Click **OK**.

Figure 123 Using the test suite inspector to set a default target



Setting-up the bootstrap loader for a test server

If you plan to use a test server, you need to follow the instructions in this section in order for the test server to use the testing framework features. These instructions also apply if you need to launch applications separately from Together.

The bootstrap loader is included in the `testingbootstrap.jar` file. You need to add this file to the JDK used with Together. In addition, you need to add the `accessibility.properties` file to the JDK. This file contains a reference to the bootstrap loader class so it can instantiate on JDK startup.

Your Together installation includes both the `testingbootstrap.jar` and `accessibility.properties` files. However, if you did not include the files during installation, you can add them at a later time using the instructions in this section.

The bootstrap loader is a class `$TGH%/modules/com/togethersoftware/modules/testingframework/lib/ext/TestServer.java` that checks two JDK properties:

`testserver.port` and `testserver.home`. You need to set properties or grant read access to system properties in order for the bootstrap loader to function.

If the properties `testserver.port` and `testserver.home` have been passed to the JDK, the bootstrap loader attempts to create the classpath for the test server and start it. An error during startup causes the Java process to terminate.

Source code of the bootstrap loader is in the `testingbootstrap.jar` file.

System Requirements

Following are the requirements for the bootstrap loader:

- Linux™, Sun Solaris™ (SPARC), Microsoft Windows® (95/98/NT/2000), Hewlett-Packard HP UX™, or Compaq Tru64™ UNIX.
- 32 Mb RAM or more
- Installation of JDK 1.3 or higher. You can obtain the JDK from the Sun web site: <http://java.sun.com/products/jdk/>
- TCP/IP protocol stack
- When installing on Windows 95, WinSock2 must be installed. You can download this from: <http://www.microsoft.com/windows95/downloads/default.asp>

Installing the bootstrap loader

To install the bootstrap loader, follow these steps:

1. Confirm that you have a copy of the JDK installed on your system.
2. If you have previously installed any version of testing extensions, remove the `%JAVA_HOME%/jre/lib/ext/testingbootstrap.jar` and `%JAVA_HOME%/jre/lib/accessibility.properties`.
3. Copy `testingbootstrap.jar` to `%JAVA_HOME%/jre/lib/ext`
4. Copy `accessibility.properties` to `%JAVA_HOME%/jre/lib`

To uninstall the bootstrap loader:

1. Delete `%JAVA_HOME%/jre/lib/accessibility.properties`
2. Delete `%JAVA_HOME%/jre/lib/ext/testingbootstrap.jar`

Setting run configuration options

Together provides a set of run configurations (Run | Run/Debug Configuration) that enable you to perform testing with the following options:

- `-Dtestserver.port=<port>` - Specify any port number above 1024 for the test server.
- `-Dtestserver.home=<directory>` - Specify a directory where the testing framework is installed. (The default is `%TGH%\modules\com\togethersoft\%TGH%\modules\testingframework`).

- `-Dtestserver.configuration=<string>` - String identifier of the test server configuration. When you run a test, Together will compare this string against the required configuration string stored in the test asset. If the strings do not match, the test will not execute.

Default values include:

- `JUnit` - junit test
- `Visual Script` - visual test
- `Test Suite - GUI` - test suite which can run visual tests
- `Test Suite - No GUI` - test suite which does not require GUI components to run
- `-Dtestserver.console=true` - Enables console output in the test server.
- `-Dtestserver.debug=true` - Use this option with the option 4 to print debugging information from the test server.
- `-Dtestserver.loglevel=1` - Enables logging-in from the test server.

Testing your application with Together

To test your application with Together, follow these steps:

1. Modify your application batch file to pass following properties to the Java machine:
 - `-Dtestserver.port` - Port on which the test server listens for connections
 - `-Dtestserver.home` - Directory where the testing framework is installed
 - `-Dtestserver.configuration` - String which identifies the configuration
2. In the Run/Debug Configuration dialog, choose the desired testing configuration.
3. Run your application.
4. Check that the test server successfully started by pointing your browser to `http://localhost:<testserver.port>/tf/`.

Running a stand-alone test server

Follow these steps to run a stand-alone test server:

1. Set the variable `JAVA_HOME` to point to your JDK installation
2. Set the variable `TGH` to point to your Together installation
3. Set the variable `TEST_SERVER_HOME` to point to `%TGH%\modules\com\togethersoft\modules\testingframework`
4. Set the variable `CONF_NAME` to any string specified in 4.1.3.1
5. Set the variable `TEST_SERVER_PORT` to valid port number
6. Execute the appropriate command:
 - **Windows:**

```
%JAVA_HOME%\bin\java -Dtestserver.port=%TEST_SERVER_PORT% -
Dtestserver.home=%TEST_SERVER_HOME% -
```

```
Dtestserver.configuration=%CONF_NAME% -classpath %CLASSPATH%
com.togethersoft.testing.TestServer
```

- **UNIX:**

```
$JAVA_HOME/bin/java -Dtestserver.port=$TEST_SERVER_PORT$ -
Dtestserver.home=$TEST_SERVER_HOME -
Dtestserver.configuration=$CONF_NAME -classpath $CLASSPATH
com.togethersoft.testing.TestServer
```

7. Check that test server started successfully by pointing your browser to `http://localhost:<testserver.port>/tf/`.

Troubleshooting

Table 46 lists conditions to check when troubleshooting your test server.

Table 46 Troubleshooting your test server

If this occurs...	...then check this condition
If the test server exits with an error message	• Testing extensions are present in your Java home • You have specified a dedicated port in the <code>-Dtestserver.port</code> option (only the test server should use the port) • All required classes (<code>xerces.jar</code>, <code>junt.jar</code>, <code>junitx.jar</code>, <code>ant.jar</code> etc.) exist in the locations below or on your classpath: • <code>%TGH%\lib\xerces.jar</code> • <code>%TGH%\lib\junit\junit.jar</code> • <code>%TGH%\lib\junitx\junitx.jar</code> • <code>%TGH%\bundled\tomcat\lib*.jar</code> • The TCP/IP network is present on your machine • You have installed WinSock2 if you are using Windows 95
An internal error message occurs when running tests results	All classes required for running test are present on classpath
An unexpected <code>ClassNotFoundException</code> is reported when running tests	All classes required for running tests are present either in your Together project classpath or the test server classpath
An unexpected <code>SecurityException</code> is thrown when running tests	• Your security policy file (located in <code>JAVA_HOME/jre/lib/security/</code>) allows you to start the server socket and create temporary files. • For a visual test, check that your security policy allows you to add an AWT Listener and create <code>java.awt.Robot</code>.
An <code>AWTException</code> is thrown upon starting an application with your virtual machine and the application exits	Your Together installation includes both the <code>accessibility.properties</code> and <code>testingbootstrap.jar</code> files as discussed in “Installing the bootstrap loader” on page 419.

Network security

Testing extensions start the HTTP server on the port specified in the `-Dtestserver.port` option. The HTTP server allows execution of arbitrary code passed through the network in the virtual machine, creating a potential security risk. To prevent unauthorized access, you can configure a firewall that prohibits external parties from accessing the port on your machine.

Known problems and limitations

At the time of this release, you cannot use other accessibility extensions with the JDK where testing extensions are installed. Instead, use a separate JDK for testing extensions.

TEMPLATES AND PATTERNS

This chapter provides instructions for the following:

- “Code templates” on page 423
- “Custom code templates” on page 427
- “Patterns” on page 431
- “Using templates and patterns” on page 435

Code templates

Code templates are used to generate the initial source code and default property values for new modeling elements you create in Together. A simple example is the default template for a class. When you create a new Java class, its default name is "Class1," and the default code generated for it is:

```
public class Class1 {
}
```

By modifying the appropriate template, you can change the name to "New1" (or whatever you want). More importantly, you can also modify the default code, adding default attributes or operations. For example, you could change the default code for a Java class so that a default addNew() operation is always generated when you create new classes in diagrams:

```
public class Class1 {
    public void addNew() {
    }
}
```

This may seem trivial at first glance, but consider the implications for things like Enterprise JavaBeans (EJBs). When you use the one-click EJB feature, you get default source code for an entity or session EJB (see figure below). By modifying the respective code templates, you can customize the default code for EJB classes, home and remote

interfaces, and so on, adding fields, properties, business methods, or whatever you want to the default source generated for new instances of the particular element in diagrams.

Textual patterns, or templates, can be regarded as an abstraction, like a form ready for "filling in" for a specific instance. Templates reside in the \$TGH\$\Templates folder, which contains separate subfolders for templates in supported languages. Each language provides support for class, link, and member templates.

A class template is stored in a folder whose name corresponds to the name of this template. This folder contains the file %Name%. * (with an extension specific to the selected language), and an optional properties file. Link and member templates have the template name with the *.link or *.member extension, respectively.

Template properties

Properties of the template are defined in an optional file `template_name.properties` in the same subfolder. This file includes values that will be substituted instead of the macros when a new object is generated, a flag that specifies whether this template will be displayed in the Choose Pattern dialog, and other information. The properties available are presented in the following table:

Table 47 Code Template properties

Property	Description
defaultName	The name of the created object, which is used as a starting value. For example, Class1 for the first generated class, Class2 for the next one, and so on.
defaultType	Defines types of the attributes and return types of the operations.
hideInChooseList	If this property is present in the *.properties file, the template is ignored by the Pattern Chooser panel.
generatePrologueEpilogue	If true, a pre-defined prologue and epilogue will be generated.
pasteClassesToOneFile	Some of the class and interface templates stipulate generation of two classes. If this flag is true, both objects are generated in a single file.
singleOccurrencePerClass	This property refers to operators and members, and specifies that this operator or member can occur in the generated class only once.
patternDescription	Contains a brief description of this template in HTML format.
doNotKeepTag	Contains the tag name that should not be preserved when an object is replaced with another one. For example, doNotKeepTag=link means that the link tag should be omitted in a new link.

The textual pattern `Default_Class` is used every time a new class is created. The same is true of the interfaces, associations, aggregations, dependencies, and so on, whose names begin with "Default_". These templates never show up in the Pattern Chooser panel.

Using template macros

You will notice in the default code delivered with Together that several "template macros" are used. For example, the default class code uses the %Name% macro:

```
public class %Name% {  
}
```

This macro expands to the default class name specified in the properties (Class1 by default). You can use any of the available macros in the code. Template macros are documented in the Templates Appendix of this guide. Each template type handles certain pre-defined macros:

Table 48 Template types and corresponding macros

Template type	Macros handled
Class	%Name%, %Class_Name%
Member	%Name%, %Type%, %Class_Name%
Link	%Name%, %Type%, %Dst%

The macros not supported by a certain template type are ignored by the parser, and a template containing unsupported macros will be inapplicable.

In contrast to pre-defined macros, each template type can handle an unlimited number of user-defined macros.

Default properties in templates

The template node for each modeling element contains two subnodes: the first for the default source code, the second for the default properties.

Each template contains a set of default properties whose values you can modify. For example, the default name is specified in the properties. Thus, to change the default code for an element, open and edit the source code node; to change default values of properties, edit the properties node. (See "Editing code templates" below.)

WARNING! NEVER edit the default templates. Improper modifications can result in incorrect functioning of Together, and make it impossible to create diagram elements.

Browsing the available code templates

You can browse through the available code templates using the Directory tab of the Explorer. (It doesn't matter whether a project is open or not.) The categories correspond to different programming languages.

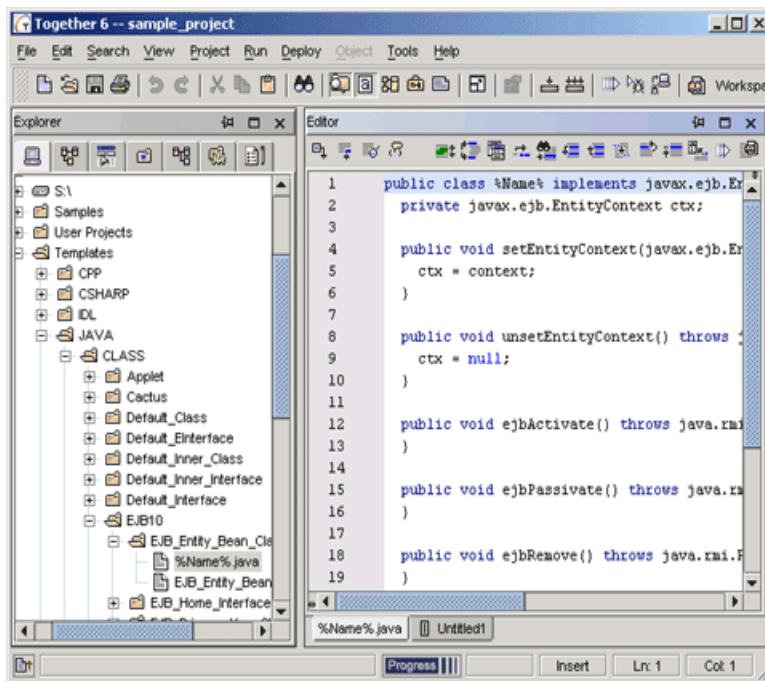
To see the main template categories:

1. Click on the Directory tab of the Explorer.
2. Expand the Templates node.
3. Expand the nodes for the various languages and observe the categories of templates.
4. Expand any node in the tree-view to see the available templates.

Note that while all templates are available to view and edit, not all templates function in all products. For example, Java templates don't work in products that support only C++. Functionality depends upon your product license.

Each main category has subcategories for class templates, link templates, and member templates. The figure below shows the structure of the Templates node in the Directory tab, and the default code for the selected element. This is the code generated for this element (in this case an Entity EJB) whenever a new one is created in the visual model. You can edit this code (see the next section) to change the default source generated for new instances of the modeling element. For example, for the EJB shown below you could add default declarations for finder or business methods.

Figure 124 Code templates in the Explorer, and the default code for an entity EJB



Editing code templates

You can edit the default source code specified by a template, the default values of properties, or both using the Together Editor. You can open multiple source templates or properties in the editor at once, thus facilitating clipboard operations between templates. Changes to source code or properties are automatically saved when you leave the Editor. You can manually save changes using the File menu or the main toolbar.

To open one template for editing:

1. Navigate to the template's node in the Directory tab of the Explorer and select it.
2. On the right-click menu of the node, choose **Edit**, or **Edit in New Tab**.

To open multiple templates for editing:

1. Navigate to a template node in the Directory tab of the Explorer and select it.
2. On the right-click menu of the node, choose **Edit in New Tab**.
3. Repeat steps 1 and 2 for additional templates as desired.

NOTE: You can use the currently configured External Editor to edit the code templates. On the right-click menu of the template, choose **Tools | External Editor**.

Applying source formatting

Source code formatting applies to templates in the same way as to other sources. You can configure a number of source code formatting options in the Source Code page of the Options dialog, such as indenting, comment format, treatment of spaces, and so on. Once you have edited a template to its final state, you can apply the currently configured source formatting options to the code.

To apply source formatting options:

1. Select the source code node (not the *properties* node) of the template in the Explorer.
2. On the right-click menu of the node, choose **Tools | Format Source**.

TIP: The same command is available on the right-click menu of the Editor.

Custom code templates

Using Together you can create your own templates and groups of templates, either using an Expert, or manually. You can also collect appropriate templates into groups, provide group-level descriptions, and customize the way the templates show up in the Pattern Chooser.

When you create custom code templates, keep in mind that there is a rigid dependency between the template type and the contents of the Inspector. For classes and interfaces, a single template-related control with the class name appears in the Inspector. The value of the class name replaces the `%Name%` macro in the template.

For links, the *Name* field refers to the `%Name%` macro, and the *Link destination* field refers to the `%Dst%` macro.

For members (attributes or operations), there is the *Name* field for the `%Name%` macro, and the *Type* field for the `%Type%` macro.

If a certain macro is not included in the template body, the corresponding field does not show up in the Inspector. However, manipulations with the pre-defined macros require special consideration, to avoid producing useless templates.

Working with the Code Template Expert

Together provides a handy way to create custom code templates using the Code Template Expert, which is invoked by the relevant command from the Tools menu, or from the right-click menu of a diagram object. Considering the selected language, the expert creates template and properties files according to the rules described above.

In the Expert you can:

- Define template properties.
- Edit template contents using template macros.
- Delete custom code templates.

To create a new code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the target language and category of template, and click **Next** to continue.
3. On the second page of the Expert, click **New Template** to add a new template.

NOTE: Click the **New Folder** button to create a group of templates.

4. On the third page of the Expert, specify the template name, and set the options for template properties.
5. Click **Edit Template Code**.
6. In the Edit Code Template window, type in the source code, using the macros buttons. Apply source formatting if necessary, and click **OK** to complete editing.
7. On the third page of the Expert, click **Finish** to complete.

NOTE: The controls of the Edit Code Template dialog are described in the relevant topic of the online help system.

The newly created templates are displayed in the Explorer pane. Now you can use them to create classes, members, and links by patterns. This technique is described in the sections “Creating classes and links by patterns” and “Creating members by pattern” in this chapter.

To edit an existing code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the language and category of template to be modified, and click **Next** to continue.
3. On the second page of the Expert, select the desired template and click **Next**.
4. On the third page of the Expert, edit the template name, and set the options for template properties as required.
5. To edit the source code, click **Edit Template Code**.
6. In the Edit Code Template window, modify the source code, using the macros buttons. Apply source formatting if necessary, and click **OK** to complete editing.
7. Click **Finish** to complete.

To delete a code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the language and category of template to be deleted, and click **Next** to continue.
3. On the second page of the Expert, select the desired template and click **Remove Template**.

4. Confirm deletion, and proceed with the other templates or close the Expert.

WARNING! The Expert protects the integrity of Together. Default templates used to create one-click diagram elements, whose names start with "Default," may not be deleted by any means. For this reason, if you select any default template in the Expert's tree, the Remove button is disabled. However, it is still possible to modify the default templates.

Creating custom code templates manually

If you so wish, you can modify the code templates by means of editing the relevant files in the Templates folder. In this case you are fully responsible for the correctness of created templates.

To create a custom code template manually:

1. According to the required template type (class, link, or member) and language, create a folder in the appropriate location. The name of this folder should be the same as the name of the template being created. Spaces are not allowed in the folder name and should be replaced with underscores. Further, when this pattern is displayed in the Pattern Chooser panel, the underscores in the folder name will be substituted with spaces.

For example: `Templates\CPP\CLASS\My_pattern`.

2. In this folder, create the file `%Name%. *` (with an extension appropriate to the target language) and optional properties file, whose name is the same as the template folder name.
3. Create the source code for the template file.
4. Write the contents of the properties file.

TIP: If you want your template to show up in the Pattern Chooser panel, do not include the `hideInChooseList=true` property in the properties file. Even if you set this flag to false, the Pattern Chooser panel still ignores the template. This line should be omitted.

The newly created templates show up in the Together Explorer pane when you restart Together. Now you can use the new templates to create classes and links by patterns.

Groups of templates

For more convenience, you can gather relevant patterns into folders and provide general descriptions for the groups of patterns.

To create a folder:

1. In the Code Template Expert, click **New Folder**.
2. In the Create New Folder dialog, enter the desired name.

Further, using the Pattern Chooser, you may want to see group-level descriptions.

To provide a description for a group of patterns, create the file `description.html` under the folder in reference, and write the necessary information. The description displays in the Description area of the Pattern Chooser when you restart Together.

Displaying custom template names

As mentioned earlier, templates are stored in folders with appropriate names. However, users might want to see more sensible names in the Pattern Chooser, with spaces, quotes, and apostrophes. To change the display name, add the following line to the template's *.properties file:

```
patternDisplayName=new name
```

The new name immediately shows up in the Pattern Chooser, but the actual name still displays in the Templates node of the Explorer.

It is also possible to rename template folders in the Pattern Chooser. To do that, create the file `folder_name.properties` in the upper-level directory. This file should contain only one line:

```
patternDisplayName=new name
```

When you restart Together, the new name shows up in the Pattern Chooser.

Example

Let us rename the group of templates *Robustness*, which resides in `$TGH$/templates/java/class`, to *Robustness Diagram*.

To rename the group:

1. Create the file `$TGH$/templates/java/class/Robustness.properties`.
2. Add the following line:

```
patternDisplayName=Robustness Diagram
```

3. Restart Together.
4. Open the Choose Pattern dialog and observe the new name in the Pattern Chooser panel.

User-defined macros

In addition to the standard macros, an unlimited number of user-defined macros are allowed for all template types. The names of the user-defined macros follow the same syntax rules as the template folder names (spaces not allowed). The Inspector provides a special control for each macro with the name of this macro, and the underscores replaced with spaces.

Example: The control "Custom Attribute Name" corresponds to the macro `$Custom_Attribute_Name$`.

It is possible to assign default values to the user-defined macros. These values initialize appropriate controls in the Choose Pattern dialog.

To assign a default value to a user-defined macro:

Under the Templates node of the Explorer, navigate to the desired template folder.

1. Open the properties file of the template for editing.
2. In the Editor, add the line `default.$Custom_Attribute_Name$=<defaultValue>`.
3. Save changes.

The macros without pre-defined default values are initialized with an empty string.

Creating templates from diagram elements

You can create a template based on your existing classes, links or members. Create a class, populate it with the necessary attributes and operations, and make a template from it.

To create a code template from a diagram element:

1. On the diagram, select an element that will become a source for a new template;
2. On the right-click menu of this element, choose **Tools | Edit Code Template** command to open the Code Template Expert, which displays the second page, the language and category of the future template being automatically assigned based on the selected element.
3. In the Code Template Expert, click **New Template** button and specify template name.
4. In the tree-view of templates, click on the created template name and press Next to proceed with the settings.
5. On the third page of the Expert, press **Edit Code Template** button to open the template editor.
6. In the Code Template Editor, click **Insert Element Text** button to enter the source code of the selected diagram element.
7. Edit the source code, using template macros buttons.
8. Click **Format Source** button if needed.
9. Click **OK** to complete editing and close the Template Editor.
10. In the Code Template Expert, click **Finish**.

The newly created template appears in the appropriate location of the Templates folder.

Patterns

Together delivers a unique capability to extend its functionality externally by using modules and patterns. Modules are discussed in the Chapter 43 “Together Open API”. In this topic you will find the basic concept of patterns, learn how to use the “pre-fab” patterns included with Together, and how to deploy your own patterns.

Concept of Patterns

Together patterns are public Java classes that implement the interface `com.togethersoftware.openapi.sci.pattern.SciPattern`.

Patterns are intended to:

- create frequently used elements,
- modify existing elements,
- implement useful source code constructions or solutions in your model.

Among the patterns provided with Together there are the Coad Components, GoF patterns including Visitor, Observer, Singleton, a set of EJB classes for various specifications, a set of TagLib patterns etc.

You can develop your own patterns and reuse them in Together. Patterns are implemented using the SCI API. To develop your own patterns, follow the basic procedure outlined in the section “Creating patterns”. You will also need to study the documentation for the Together API. (For more information on the API documentation see Chapter 43 “Together Open API”.) You can also study examples of mature patterns in the \$TGH\$\modules\com\togethersofter\modules\patterns directory in your installation and obtain the sources of the patterns at the community site <http://www.togethercommunity.com/>.

Working with various languages

Some patterns work only with a specific language (for example, Enterprise Java Bean pattern can work only with the Java language), and some can be applied to any language (for example, `com.togethersofter.modules.patterns.UNIVERSAL.MEMBER.Stub_implementation`).

Together determines the target language for a pattern automatically based its location in the installation. The package of a pattern defines the target language for the pattern (or elements produced by the pattern).

Table 49 Languages and corresponding patterns

Language	Package
Java	<code>com.togethersofter.modules.patterns.JAVA</code>
C++	<code>com.togethersofter.modules.patterns.Cpp</code>
All languages	<code>com.togethersofter.modules.patterns.UNIVERSAL</code>

Elements generated by patterns

The package of a pattern and its subpackages define the target elements for the pattern (or elements produced by the pattern).

Table 50 Elements generated by patterns

Element	Package
Class	<code>com.togethersofter.modules.patterns.JAVA.CLASS</code> , <code>com.togethersofter.modules.patterns.Cpp.CLASS</code> , <code>com.togethersofter.modules.patterns.UNIVERSAL.CLASS</code>
Link	<code>com.togethersofter.modules.patterns.JAVA.LINK</code> , <code>com.togethersofter.modules.patterns.UNIVERSAL.LINK</code>
Member	<code>com.togethersofter.modules.patterns.JAVA.MEMBER</code> , <code>com.togethersofter.modules.patterns.UNIVERSAL.MEMBER</code>

Behavior of a pattern

Behavior of a pattern is defined by its properties set. The `SciPattern` interface defines the methods that you should implement in your pattern.

Table 51 Methods of the `SciPattern` interface

Method	Description
<code>apply</code>	Makes the pattern perform the desired actions.
<code>canApply</code>	Checks whether the pattern can be applied to the target object (objects) with the current values of pattern's properties.
<code>prepare</code>	Checks if it is possible to apply this pattern to the target object (objects) at all, and makes some startup preparations for the pattern. It returns <code>true</code> if everything is okay, and <code>false</code> if the pattern cannot be applied to the target object (objects) at all.
<code>getProperties</code>	Returns an instance of the <code>PropertyMap</code> that contains the set of all the pattern properties.

Other information about pattern-related interfaces and pattern properties can be found in the main API documentation (`/doc/api` folder in your Together installation).

Creating patterns

It is possible to create a pattern hierarchy within the packages described above. In this case Together will use a name of a subpackage as a name for a group of patterns defined in this package.

For example, a pattern defined in a package

```
com.togethersoft.modules.patterns.JAVA.CLASS.foo_Pattern
```

will be displayed in a separate group called "foo Pattern" in class pattern chooser dialog (Together automatically replaces an underscore sign with a space).

Consider the development of the simplest pattern step by step, using Together. This pattern does nothing, but nevertheless, it will help explain the key point of pattern deployment.

1. Create a folder `foo_Pattern` under `com.togethersoft.modules.patterns.JAVA.CLASS`. This pattern will work with Java classes only.
2. Create a Together project around this folder.
3. Add new class to the default diagram, and change its name to `Pattern.java`.
4. Add the following code in the editor pane:

```
package com.togethersoft.modules.patterns.JAVA.CLASS.Pattern;
import com.togethersoft.openapi.sci.pattern.SciPattern;
import com.togethersoft.openapi.util.propertyMap.PropertyMap;
import com.togethersoft.openapi.util.propertyMap.DefaultPropertyMap;
public class Pattern implements SciPattern {
    public boolean prepare () {
        return true;
    }
    public boolean canApply() {
```

```

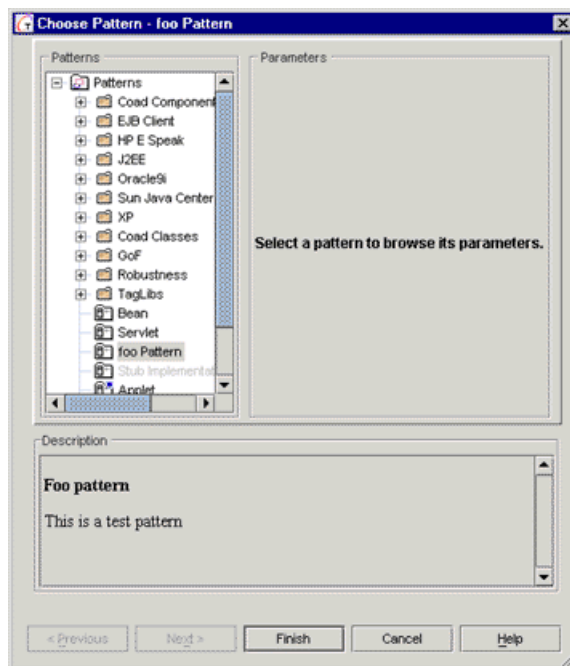
return true;
}
public void apply() {
}
public PropertyMap getProperties(){
return myProperties;
}
private PropertyMap myProperties = new DefaultPropertyMap();
}

```

5. On the project's right-click menu, choose **Project Properties** command, and check the flag *Include Classpath* in the Search/Classpath tab.
6. Adjust the compiler output on the project level:
 - On the main menu choose **Tools | Options | Project Options**
 - Choose **Builder | Built-in Javac | Compiler options**.
 - In the *Destination directory* field, change the default destination folder to \$TGHS\$\modules\com\togethersoftware\modules\patterns\JAVA\CLASS\foo_Pattern\
7. On the right-click menu of the class, choose **Tools | Make Node** to compile the class. The file Pattern.class is written to the destination folder specified above.

Now this pattern is ready, and you can see it in the Choose Pattern dialog. You can immediately try this out by clicking Class by Pattern icon. The tree-view of available patterns and templates shows the new pattern:

Figure 125 New pattern in the Choose Pattern dialog



Note that the new pattern is not marked with an asterisk in the patterns tree-view. This denotes the difference between binary patterns and textual templates.

Providing pattern description

To do that you have to place a file named `Description.html` in a directory with your pattern. The contents of this file will be displayed in the Pattern Chooser dialog in the description section.

Using templates and patterns

Though different in nature, both templates and patterns are used to create or refactor diagram elements via the Choose Pattern dialog. Textual templates in the pattern choose are marked with an asterisk, to tell them from the binary patterns. However, both reusable components are applied in the same way.

If a textual template is selected in the tree-view of available patterns, the preview of the code is displayed in the right pane. For the binary patterns preview is not available. You can find detailed description of controls in the online help system.

Creating classes and links by patterns

Together makes it easy for you to apply patterns and templates when creating classes or links. To create classes or links during modeling, you can use the following icons on the Class Diagram Toolbar:

Table 52 Toolbar buttons that use templates and patterns

Tool-tip	Icon	Description
Class by Pattern		Create a new class using selected pattern to define how code is generated
Link by Pattern		Create a relationship link using selected pattern to define how code is generated

Both icons launch the Choose Pattern dialog displaying the available patterns for the respective operation. You can access Together's predefined patterns plus any that you implement yourself.

To create a class or link by pattern:

1. Click the appropriate toolbar icon.
2. If creating a class, click on the Diagram pane to display the Choose pattern dialog for classes. If creating a link, drag it from the source class to the destination and drop it to display the Choose pattern dialog for links.
3. Select the pattern you want for the new class or link.
4. If the Next button is enabled, there are properties or parameters for the pattern. Click Next to display them. (If you know the pattern and want to accept the defaults, click Finish).
5. Set properties or parameters as desired and click Finish.

NOTE: Users of the Microsoft JVM should note that java.rmi classes, required for compilation of the EJB pattern, are not present in this JVM.

WARNING! The result of applying a pattern to a class depends on the way it was invoked. If you are going to create a new class by pattern, make sure you click on the Diagram pane, rather than on a class shape. In the latter case, the class where you click is regarded as a container class, and the newly created class by pattern is added as an inner class.

Invoking Choose Pattern command from the class right-click menu enables refactoring of the class according to the selected pattern.

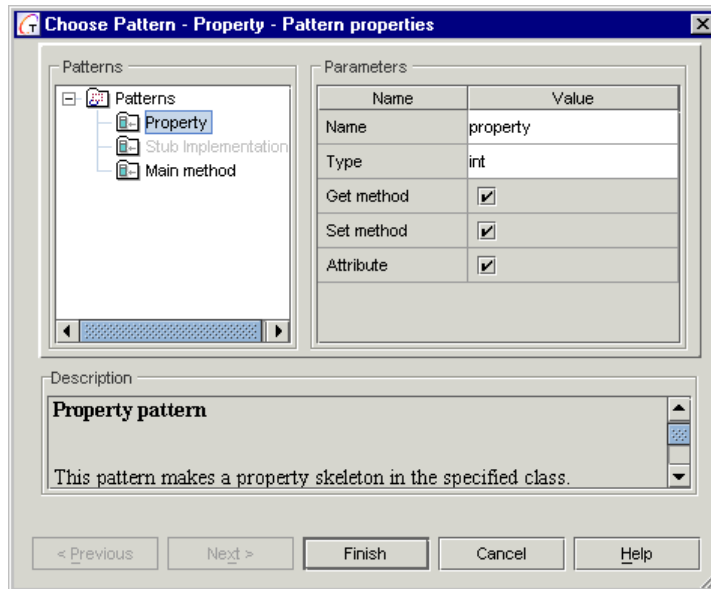
Creating members by pattern

Unlike classes and links, there is no toolbar button for this operation.

To create a member by pattern:

1. Open the class right-click menu.
2. Choose **New | Member by Pattern**.
3. In the [Choose Pattern dialog](#) select the desired property pattern.
4. Enter property name and type, and check the boxes for accessor methods and attributes if necessary.

Figure 126 Choose Pattern dialog for Members



Choosing a pattern for a Member

You can also apply patterns to control the way implementation code is generated for an attribute or operation.

The Choose Pattern dialog displays the *Member* and *Link* nodes. Member patterns can make the selected member an attribute, an operation, or even regenerate the code for the property with get and set method. The Link patterns can make the selected member a link.

To apply a pattern to a member:

1. Select an attribute or operation in a class.
2. On the right-click menu of the member, click **Choose Pattern**. The Choose Pattern dialog is opened displaying the available patterns for members.
3. Select the pattern you want for the member.
4. Click **Next** button to display the page with the properties or parameters of the pattern, or click **Finish** to accept the defaults. (If there are no properties or parameters, the Next button is disabled).
5. Set properties or parameters as desired and click **Finish**.

NOTE: The Choose Pattern command is enabled only if the selected element satisfies certain criteria which make it possible to regenerate the code without any losses. Thus, if an operation already has some code in its body, the command is disabled.

Refactoring with patterns

You can also refactor existing classes or links replacing an existing pattern with a better one. You can do this singly, or to multiple selected elements.

To refactor classes or links:

1. Select the elements in the Diagram pane.
2. Click **Choose Pattern** on the right-click menu of the selected elements to display the appropriate Choose Pattern dialog.
3. As above, select the desired pattern for refactoring, set any properties or parameters, and click **Finish**.

TIP: You can also access the Choose Pattern dialog from the right-click menus of Class diagram elements in the Explorer.

CHAPTER 25

AUDITS AND METRICS

This chapter includes the following topics:

- “Overview of audits and metrics analysis” on page 438
- “Running audits and metrics in Together” on page 440
- “Language-specific metrics and audits” on page 452
- “Running audits and metrics from the command line” on page 457
- “Additional information sources” on page 459

Overview of audits and metrics analysis

Together provides audits and metrics as Quality Assurance features to unobtrusively help you enforce company standards and conventions, capture real metrics, and improve what you do. Although audits and metrics are similar in that they both analyze your code, they serve different purposes.

When you run audits, you select specific rules to which your source code should conform. The results display only the violations of those rules so that you can examine each problem and decide whether to correct the source code or not. Together provides a wide variety of audits to choose from, ranging from design issues to naming conventions, along with descriptions of what each audit looks for and how to fix violations.

Metrics, on the other hand, quantify your code. It is up to you to examine the results and decide whether they are acceptable. Metrics results can highlight parts of code that need to be redesigned, or they can be used for creating reports and for comparing the overall impact of changes in a project. Along with the full set of metrics, Together provides tips for using metrics and interpreting results.

Audits and metrics are run as separate processes. Because the results of these two processes are different in nature, Together provides different features for interpreting and organizing the results. Note that some of the features and procedures described in this chapter apply to both audits and metrics, while some are specific to either one or the other.

How should metrics be used?

The following recommendations are taken from the book *Object-Oriented Metrics: Measures of Complexity* by Brian Henderson-Sellers. For other resources, see the list “Additional information sources” on page 459 at the end of this chapter.

The use of metrics depends on the maturity of the company, and therefore on the level of organization and control of the development process. The set of metrics in use can be juxtaposed to the organizational level, as shown in the table below.

Table 53 Identification of the organizational level in the Capability Maturity Model, and metrics usage

Level	Characteristic	Metrics to use
1. Initial	Ad hoc	Baseline
2. Repeatable	Process dependent on individuals	Product
3. Defined	Process defined, institutionalized	Product
4. Managed	Measured process (quantitative)	Process + feedback to control
5. Optimizing	Improvement fed back to process	Process + feedback for changing process

Baseline metrics – “no order” in metrics using.

Product metrics – metrics on estimating and measuring size.

Process metrics – metrics on estimating effort (depends on size and team productivity).

However, dividing metrics into two groups (the product and the process) is relative because the same metric might represent product size and also impact productivity.

Metrics can be collected for various purposes such as cost estimation, project management, quality assessment, calculating productivity, detecting defects, or program maintenance. To apply metrics successfully, it is important for managers and developers to observe a number of rules, summarized in the table below.

Table 54 Rules of etiquette for applying software metrics

Management level	Rules
Functional management	1. Do not allow anyone in your organization to use metrics to measure individuals. 2. Set clear goals and get your staff to help define metrics for success. 3. Understand the data that your people take pride in reporting; do not use it against them; do not ever hint that you might. 4. Do not emphasize one metric to the exclusion of another. 5. Support your people when their reports are backed by data useful to the organization.

Table 54 Rules of etiquette for applying software metrics (continued)

Management level	Rules
Project management	6. Do not try to measure individuals. 7. Gain agreement with your team on the metrics that you will track, and define them in a project plan. 8. Provide regular feedback to the team about the data it helped collect. 9. Know the strategic focus of your organization, and emphasize metrics that support the strategy in your reports.
Project team	10. Do your best to report accurate, timely data. 11. Help your managers to focus project data on improving your process. 12. Do not use metrics to brag about how good you are or you will encourage others to use other data to show the opposite.

Together provides tips on how to use the results of specific metrics. These tips are displayed on the Advice tab for the Distribution graph (see “Distribution graph” on page 448). Remember that when interpreting values for metrics, it is not a good idea to apply the suggested limits dogmatically. For every limit, there are good arguments for exceeding the maximum. However, consider such cases as exceptions, rather than the rule.

Running audits and metrics in Together

When the Quality Assurance module is activated, the commands Audits, Metrics, and Load Metrics Results are available on the Tools menu.

- To process the entire project, choose Audits or Metrics from the Tools menu.
- To process specific classes, packages, or diagrams, select the elements either on the model tree in the Explorer or on the diagram, and choose Audits or Metrics from the right-click menu.

You can create, save, and reuse custom sets of audits and metrics. Together ships with a pre-defined saved Audit set for the *Sun Code Conventions for Java*, which you can load and use in place of the default audit set, or any custom sets you create. For more information, see “Creating and using saved sets of metrics or audits” on page 451.

NOTE: It is worth mentioning that both audits and metrics are only valid for source code that can compile. If your source code contains errors, or some libraries and paths are not included, audits and metrics might produce inaccurate results.

Available audits and metrics

Availability of audit and metric features depends on the project language. Java projects support a wide range of metrics and audits. Other languages have smaller sets of metrics and audits that have been adapted or created to fit the particular language. For lists of the metrics and audits available for languages other than Java, see “Language-specific metrics and audits” on page 452.

Some audits and metrics accompany specific Together modules. Currently, these include audits and/or metrics that support the Real-Time and UI Builder modules. These audits and metrics are available only if the corresponding module is activated in Tools | Activate/Deactivate Features.

How to perform metrics analysis

Metrics analysis evaluates object model complexity to support quality assurance.

To perform metrics analysis:

1. Open a project.
2. Make sure that the Quality Assurance module is activated (**Tools | Activate/Deactivate Features - Together Features: Quality Assurance**).
3. Choose **Tools | Metrics** on the main menu.
4. Select the metrics you want to analyze. Each metric displays a description in the lower pane of the Metrics dialog box. For each metric, there are settings for options such as limits and granularity in the right-hand pane of the Metrics dialog box. Change the settings if necessary.
5. When you have selected your set of metrics, click **Start**.
6. The results are displayed on the Metrics tab in the Message pane. In the results table, select any element and choose **Open** from the right-click menu to navigate directly to it in the diagram or source code.

Metrics for audit violations

You can also run metrics on the most recent audit results. In the Metrics dialog, the available audits are listed in the *Audit Violations* group. The metrics for audit violations have the same abbreviations as the corresponding audits, only with the prefix *av*. The metrics for audit violations are calculated as the number of violations in each class for the selected audit. The total sum of audit violations can also be calculated (run the metric TAV).

How to perform audits

Audits automatically check for conformance to standard or user-defined style, maintenance, and robustness guidelines.

To perform audits:

1. Open a project.
2. Make sure that the Quality Assurance module is activated (**Tools | Activate/Deactivate Features - Together Features: Quality Assurance**).
3. Choose **Tools | Audits** on the main menu.
4. Select the audits you want to run. Each audit displays a description in the lower pane of the Audits dialog box. For each audit, the severity level (and other audit-specific options, in some cases) are displayed in the right-hand pane of the Audits dialog box. Change the settings if necessary.
5. When you have selected your set of audits, click **Start**.

6. The results are displayed on the Audits tab in the Message pane. In the results table, select any element and choose **Open** from the right-click menu to navigate directly to it in the diagram or source code.

TIP: If you do not want to process the entire project, run audits or metrics on specific packages, classes, or diagrams. In step 3 above, select the elements either on the model tree in the Explorer or on the diagram, and choose Audits or Metrics from the right-click menu.

Output for audits and metrics

Audits and metrics display reports of analysis results in the Message pane. Metrics results are displayed as a table: the rows show the classes, packages, or diagrams that were analyzed, and the columns show the corresponding values of selected metrics. Audit results display a list of audit violations and where they occurred.

The results of both audits and metrics analysis are tightly connected with source code. From any line of the results table, you can navigate to the appropriate location both in the diagram and in the source code. Select the row in the results table that is of interest to you, and double-click it, or choose **Open** on the right-click menu. The same functionality is available for metrics on the bar graph (see “Bar graph” on page 447).

Sorting results

When viewing the output of audits or metrics, you will probably want to compare and organize the items in the results table. There are several ways to sort results.

- Sorting by one column.
 - To sort all the items according to the values for a specific column, right-click anywhere in the column and choose **Sort Ascending** or **Sort Descending**.
 - Alternatively, click on the column heading. The arrow in the heading displays whether sorting is ascending or descending. Click again to change the direction.
- Complex sorting.
 - To prioritize several columns for sorting, right-click anywhere in the table and choose **Sort**. In the Sort dialog, you can choose up to three columns to use for ordering the items.
 - Alternatively, hold down the Shift key and click on a column heading to add it to the sorting list. The columns selected for sorting are marked with the sorting order (1, 2, 3, and so on). Shift + click again to change the sorting direction. Clicking on any column heading (without Shift) cancels complex sorting and sets sorting to that column only.
- Grouping audit results.
 - To group items according to the current column, right-click in the Audit table and choose **Group By**. This enables you to organize the results by changing the relationship of rows and columns.

Updating results

You can update or refresh the results table from the table’s right-click menu.

- Click **Refresh** to recalculate the results that are currently displayed.

- Click **Restart** to open the Audits or Metrics dialog, where you can re-select the set that you want to run, and change options if necessary.
- For audits, click **Refresh in Results** to check only the classes that are displayed in the results table. This does not re-run classes that did not produce any violations. This is useful for comparison after making corrections, and it is considerably faster than running audits on all the classes.
- For audits, click **Turn Off Selected Rules** to remove the selected audit rules from the results table after refreshing the results. This excludes the selected audits from the current set of audits to run, so that when you perform Refresh or Refresh in Results, these rules are ignored.

Exporting results

To save the results of metrics or audits in a separate file:

1. On the right-click menu of the results table, point to **Export**, and choose the scope of the results to export (**Entire Table** or **Selected Rows**).
2. In the Export Results to File dialog box, specify the target file location and name.
3. In the *Output type* drop-down list, select the format for the exported file.
 - *Separated by tabs* - The results table is saved in a text file with tabs between the columns.
 - *Aligned with spaces* - The results table is saved in a text file with columns aligned by spaces.
 - *Generate HTML file* - The results table is saved in table format in an HTML file.
 - *Create multifile HTML report (metrics only)* - The results table is saved as a set of HTML files, with hyperlinks for navigating through the project tree model (see “Creating linked HTML reports for metrics” on page 443).
 - *Save in loadable format (metrics only)* - The results table is saved in a *.mtbl file that can be re-loaded in Together independently of the project (see “Saving and loading metrics results” on page 444).
4. Click **Ok** to save the results in the specified location.

Creating linked HTML reports for metrics

For metrics results, it is possible to generate a report in HTML format as a set of linked HTML files. The set of files representing the results of metrics analysis includes separate files for each package and class involved in the analysis, a file that covers results for all packages, and a file for all classes. All these files are hyperlinked to each other. This makes it possible to navigate through the project tree model.

To generate a linked HTML report for metrics:

1. On the right-click menu of the results table, point to **Export**, and choose the scope of the results to export (**Entire Table** or **Selected Rows**).
2. In *Output type*, select *Create multifile HTML report*.
3. In *Report directory*, enter the path to the folder where you want to save the report files, or click the chooser button to select a path in the file chooser.
4. The settings under *HTML Options* specify where to save the set of HTML files that contain individual descriptions of each metric. These are the same descriptions that display in the Metrics dialog, and they can be referenced from the HTML report. By

default, these files are saved in a folder called \descriptions under the directory that you specified for output. If you need to specify a folder other than the default, uncheck *Standard location*. (Click **Help** in the dialog for more details on how to set these options.)

5. Check *Launch viewer* to automatically open the report in the browser.
6. Click **Ok**. While viewing the report, use the links to see all classes or all packages. Click on a column heading (metric abbreviation) to view the full name and description for that metric.

Saving and loading metrics results

After you run a metrics analysis, you can choose to save the results and later view the results table independently of the project. You can also use results files to share metrics results with other Together users. If you do not save results, you cannot view them again after you close the results table. Saved metrics results have the .mtbl extension.

To save a set of metrics results:

1. Right-click anywhere in the results table and choose **Export | Entire Table** from the right-click menu.
2. In the Export Results to File dialog, specify the path to the directory where you want to save the file and give the file a name.
3. In *Output type*, choose *Save in loadable format*.
4. In *Comment*, type a brief description of the results table that you are saving. This text is displayed as the title of the table when you open the file. If you leave this field empty, the filename is used as the table title instead.

The file is saved with the .mtbl extension.

To load a saved set of metrics results:

1. Choose **Tools | Load Metrics Results** on the main menu. It is not necessary to have a project open.
2. In the Load Metrics Results dialog, navigate to the *.mtbl file that contains the results you want to view, and click **Ok**. You can choose any existing *.mtbl file; it does not depend on the project.

The results table opens in the Message Pane on the Metrics tab.

NOTE: The subtab for the table displays the name or text that was typed in the *Comment* field when the file was saved. If there is no *Comment*, the short filename is used instead. When you focus on the name, a tool tip displays the full path and filename of the saved *.mtbl file.

Comparing metrics results

If you have several sets of metrics results open, you can compare the values in one table against the values in another table. Use this feature for comparing projects or for comparing changes in a project over time.

Differences between results can be both highlighted in the table, and listed in tool tips.

- For comparing one table in relation to another, colors are used to highlight table cells with values that are higher or lower than the values in the other table.

- For comparing specific values in two or more tables, a tool tip over each table cell shows a list of tables with their values for the corresponding cell (that is, the results for the same metric in the same object).

You can also quickly navigate from a selected object in the current table to the same object in a different table, using the *Go to Object in Table* command.

To compare tables:

1. Make sure that all of the tables you want to compare are open. The differences will be displayed in the table that is currently focused.
2. Right-click anywhere in the table and choose **Compare** on the right-click menu.
3. In the Compare dialog, you can optionally change the color scheme for highlighting differences. Use the color chooser buttons to make changes.
 - *Higher* is the color used to highlight table cells that have higher values than in the other table.
 - *Lower* is the color used to highlight table cells that have lower values than in the other table.
 - *Not found* is the color used to highlight table cells that do not exist in the other table.
 - If the values are the same, they are not highlighted.
4. In *Compare with*, the drop-down list displays all the open tables. Choose the table to visually compare results against. The box above displays an informational message if the second table does not have some of the columns that are in the first table.

If you select *No comparison*, nothing will be highlighted.

5. Under *Show in Tool Tips*, select the names of the tables that you want to include in the tool tips for comparing cell values. Add them to the *Selected tables* box.

If you do not want to display tool tips, leave the *Selected tables* box empty.

6. Click **Ok**.

To remove comparison highlighting and tool tips:

1. Right-click anywhere in the table and choose **Compare** on the right-click menu.
2. In *Compare with*, choose *No comparison* and click **Ok**.

The table is restored to the way it looked before you performed the comparison.

To find a corresponding object in another table:

1. Right-click anywhere in the row of the package, class, or operation that you want to find. Choose **Go to Object in Table** on the right-click menu.
2. The submenu displays a list of open tables. Choose the one you want to find the object in.

If the object exists in the results table you selected, the table opens, with the corresponding object selected. If the table does not open, it means that the object was not found.

Graphic output of metrics results

Metrics results can be viewed as graphic output. Right-click on a table cell that contains results for a metric, and choose a type of graph to display (Bar Graph, Distribution Graph, or Kiviat Graph).

NOTE: Bar graphs and distribution graphs display only the nodes that are expanded in the results table. If it seems that some objects are missing in a bar graph or distribution graph, it is probably because they are hidden in the table. Use the +/- buttons to the left of the table to expand or collapse branches before generating a graph.

Kiviat graph

The Kiviat graph (see Figure 127) demonstrates the analysis results of the currently selected class or package for all the metrics that have pre-defined limiting values. The metrics results are arranged along the axes that originate from the center of the graph.

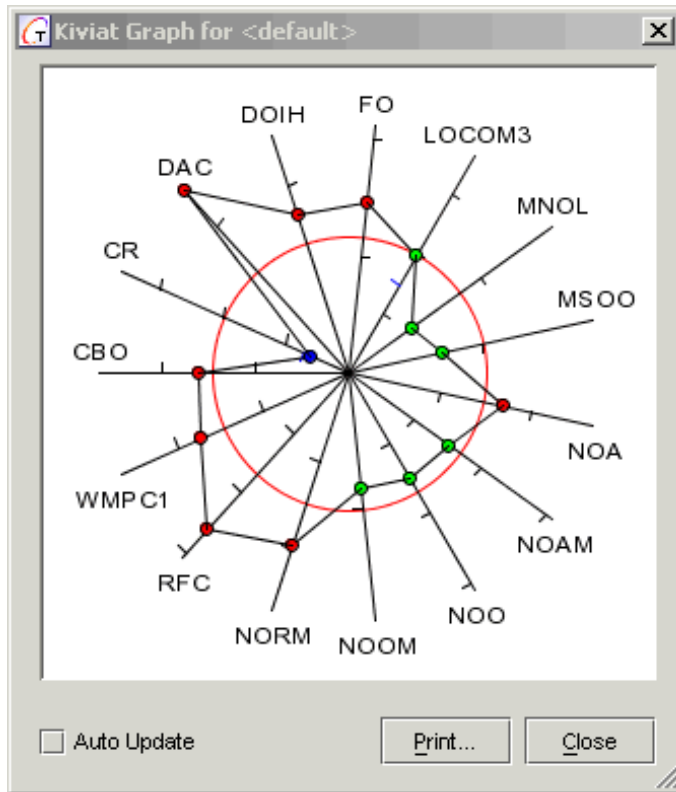
Each axis has a logarithmic scale with the logarithmic base being the axis metric upper limit, so that all limiting values are equidistant from the center.

In this way, the limiting values are represented by a red circle, and the actual metrics show up in the form of a star. If a point falls outside of the circle, it means that the relevant metric is exceeded.

The following notation is used in the Kiviat graph:

- Green points represent normal values.
- Blue points represent values below the lower limit.
- Red points represent values over the upper limit.
- Scale marks are displayed as clockwise directional ticks perpendicular to the Kiviat ray.
- Low limit labels are displayed as counter-clockwise directional blue ticks perpendicular to the Kiviat ray.

Figure 127 Kiviatt Graph



Bar graph

The bar graph (see Figure 128) displays the results of a selected metric for all packages, classes, and/or operations.

The bar color reflects conformance to the limiting values of the metric in reference:

- Green represents values that fall within the permissible range.
- Red represents values that exceed the upper limit.
- Blue represents values that are lower than the minimal permissible value.
- A thin vertical red line represents the upper limit, and a thin vertical blue line represents the lower limit.

To change the metric to display results for, select from the drop-down list of metrics in the dialog box. You can refer to the explanation of the current metric on the Description tab.

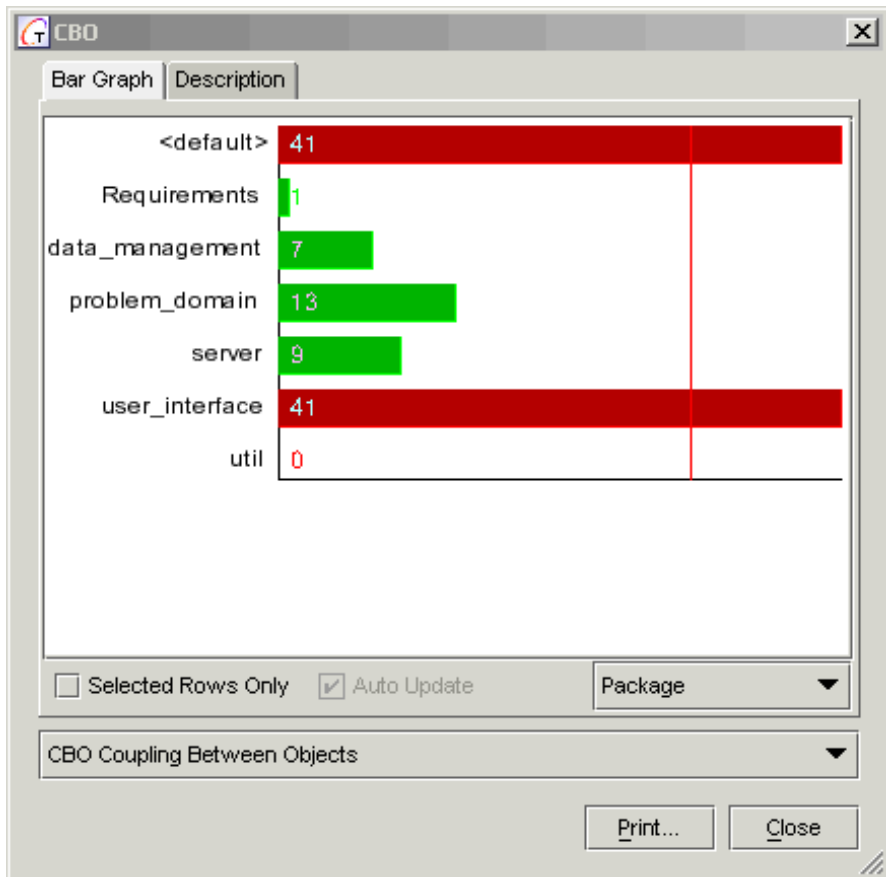
To set the scope of the results to display, choose *Package*, *Class*, or *Operation* from the drop-down list in the dialog box.

To view results only for the highlighted rows of the table (instead of all rows), check *Selected rows only*. To browse through the table and view the graph, check *Auto update*.

If *Auto update* is unchecked, the graph displays the original selected rows and does not change if you select different rows in the table.

You can navigate directly from the bar graph to the source code in the Editor pane and the corresponding element in the Diagram pane. Double-click on a bar in the graph, or right-click and choose **Open**.

Figure 128 Bar Graph



Distribution graph

The distribution graph (see Figure 129) displays the distribution of values of all packages, classes, and/or operations for a selected metric. This graph is particularly useful for getting an overview of an entire project based on a specific metric.

A thin vertical red line represents the upper limit, and a thin vertical blue line represents the lower limit.

Check *Log X* to display the x-axis on a logarithmic scale. In most cases, this makes the graph easier to interpret correctly.

To change the metric to display results for, select from the drop-down list of metrics in the dialog box.

To set the scope of the results to display, choose *Package*, *Class*, or *Operation* from the drop-down list in the dialog box.

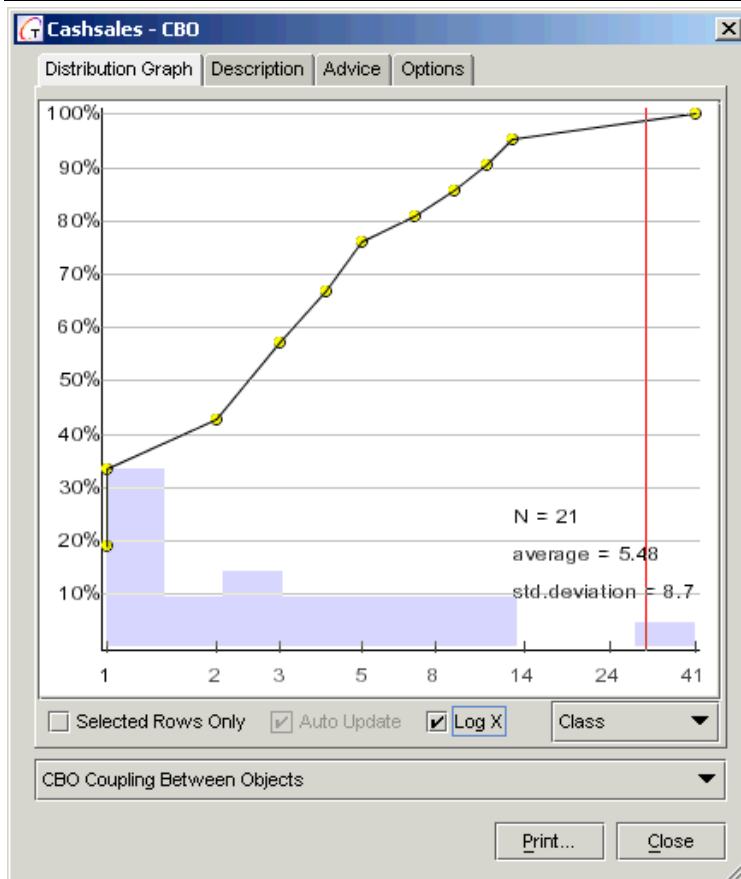
To view results only for the highlighted rows of the table (instead of all rows), check *Selected rows only*. To browse through the table and view the graph, check *Auto update*. If *Auto update* is unchecked, the graph displays the original selected rows and does not change if you select different rows in the table.

For a general explanation of the current metric, refer to the Description tab.

For tips on how to interpret the results and how to improve them, refer to the Advice tab. This tab also displays the extreme highs and/or lows for the current metric, in tables called *Top of metric* and *Bottom of metric*. These tables point you to packages or classes that you should consider revising.

To review the upper and lower limits and other options that are set for the current metric, refer to the Options tab.

Figure 129 Distribution Graph



Printing results

Print results either by printing the table displayed in the Message pane or by printing graphs when you view them.

Printing results from the table

To print the entire table, right-click anywhere in the table and choose **Print** from the right-click menu. In the Print dialog, you can preview the output and set the printing scale.

To print the results for specific metrics:

1. Right-click anywhere in the table and choose **Print** from the right-click menu.
2. Under *Select Columns to Print*, check the boxes corresponding to the metrics you want to print results for (all columns are checked by default, so you need to uncheck those metrics that you do not want to print).
3. Under *Scope*, check *Expand nodes* if you want to print the contents of collapsed nodes. If *Expand nodes* is not checked, only the items that are currently visible in the table will be printed.
4. Preview the output and set the printing scale if necessary.

To print the results for specific rows:

1. First select the rows in the table that you want to print.
2. Choose **Print** from the right-click menu. The Print dialog opens.
3. Under *Scope*, choose *Selected rows*.
4. Under *Scope*, check *Expand nodes* if you want to print the contents of collapsed nodes. If *Expand nodes* is not checked, only the items that are currently visible in the selected rows will be printed.
5. Under *Select Columns to Print*, check the boxes corresponding to the columns you want to print for the selected rows (all columns are checked by default).
6. Preview the output and set the printing scale if necessary.

Printing graphic results

All graphs can be printed. To print a currently displayed graph, click the **Print** button in the graph dialog box. In the Print dialog, preview the output and set the printing scale.

Automatic correction of audit violations

Some of the audit rules provide automatic correction for violations. This helps you quickly fix certain problems when you run audits to check your own code. In the results table, violations that can be automatically corrected are marked with the letter F in the *Fix* column. A lock symbol is displayed when a violation cannot be fixed because it is in a read-only file.

To automatically correct violations:

1. In the table, select the rows with violations you want to fix.

2. Choose **Auto Correct** on the right-click menu.
3. For each violation, you are prompted to confirm automatic correction. In the Auto Correct dialog, click Yes to fix the violation that is displayed, or No to skip it.
4. Optionally set the scope of your confirmation under *Assume this answer for*. In this way, you can choose to either handle each violation individually or immediately fix or skip a group of violations.

The violations that have been fixed are marked with a green checkmark in the *Fix* column. The corrected source code is highlighted in the Editor pane.

Table 55 Audits that have automatic correction features

Audit Group	Abbreviation	Audit Name
Documentation	DCBA	Detect Collection-Based Associations
Declaration Style	CPASBF	Constant Private Attributes Should Be Final
Possible Errors	CSF	Call <code>super.finalize()</code> from <code>finalize()</code>
Possible Errors	UL	Use ‘L’ Instead of ‘l’ at the End of Integer Constants
Possible Errors	AUVK (C++ only)	Always Use ‘virtual’ Keyword
Possible Errors	UVD (C++ only)	Use Virtual Destructor
Superfluous Content	ILC	Import List Construction
Superfluous Content	OIM	Obsolete Interface Modifiers
Superfluous Content	UIMM	Unnecessary Interface Member Modifiers
Superfluous Content	UPCM	Unused Private Class Members

Creating and using saved sets of metrics or audits

The Metrics dialog and the Audits dialog display the set of all available metrics and audits respectively. These are specific to the project's target programming language. When you open a project, a default subset of all available metrics and audits for the language is active in each dialog. Active metrics/audits are indicated by a checkmark. If you open one of the dialogs and click *Start*, all of the active metrics/audits are processed.

You will not want to run every metric/audit in the default active set every time, but rather some specific subset of available metrics/audits. Together allows you to create saved sets of active metrics and audits that can be loaded and processed as you choose. You can always restore the default active set using the *Set Defaults* button in the dialogs.

Use the default active Metrics/Audits set or any saved set as the basis for creating a new saved set. For example, you could load the saved audit set *SunCodeConventionsforJava.adt* and create a new saved Audits set based on it.

The default location for saved Metrics/Audits sets is: `$TGH$/modules/com/togethersoft/modules/qa/config`.

To create a saved set of active metrics/audits:

1. Open the relevant dialog (Metrics or Audits) as described in “Running audits and metrics in Together” on page 440.

2. If you want to base your new saved set on the default set, click **Set Defaults**. If you want to base it on a previously created custom set, click **Load Set**, then navigate to and select the desired saved set file (.adt for audit sets, .mts for metrics).
3. Go through the individual metrics/audits and check those you want to include in the set, or uncheck those you do not want to include. Select all the items in a group by checking the group name.
4. When you complete your selection, click **Save Set As** and specify the location and filename for the saved set file.

To use a saved set of active metrics/audits:

1. Open the relevant dialog (Metrics or Audits) as described in “Running audits and metrics in Together” on page 440.
2. Click **Load Set** and navigate to the saved set file you want to use.
3. Click **Start**.

TIP: You might want to include the .adt and .mts files in your backup routine.

Language-specific metrics and audits

Metrics and Audits for languages other than Java are not fully supported. This can be explained by the lack of deep parsing in C#, Visual Basic 6 and Visual Basic .NET. Deep parsing in C++ is also insufficient. For this reason, these languages support declaration-based metrics and do not support the ones that require deep parsing. Audits are not supported for the same reason because deep parsing is involved in nearly every audit.

C++ support includes an extensive set of metrics and audits that have been adapted for C++, as well as several that are specific to C++ and are not supported for any other languages.

The sets of metrics that have been adapted for C# and Visual Basic .NET are similar to each other.

Visual Basic 6 lacks inheritance, so all metrics related to inheritance and polymorphism are not available.

Table 56 C++ Metrics

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
CBO	Coupling Between Objects
CC	Cyclomatic Complexity
CDBC	Change Dependency Between Classes
CF	Coupling Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling

Table 56 C++ Metrics (continued)

DOIH	Depth Of Inheritance Hierarchy
FO	FanOut
HDiff	Halstead Difficulty
HEff	Halstead Effort
HPLen	Halstead Program Length
HPVoc	Halstead Program Vocabulary
HPVol	Halstead Program Volume
LOC	Lines Of Code
LOCOM1	Lack of Cohesion of Methods 1
LOCOM2	Lack of Cohesion of Methods 2
LOCOM3	Lack of Cohesion of Methods 3
MHF	Method Hiding Factor
MIC	Method Invocation Coupling
MIF	Method Inheritance Factor
MNOL	Maximum Number of Levels
MNOP	Maximum Number Of Parameters
MSOO	Maximum Size of Operation
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOCON	Number of Constructors
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
NOprnd	Number of Operands
NOprtr	Number of Operators
NORM	Number of Remote Methods
NUOprnd	Number of Unique Operands
NUOprtr	Number of Unique Operators
PF	Polymorphism Factor
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members
RFC	Response for Class
TAV	Total Audit Violations
TCR	True Comment Ratio

Table 56 C++ Metrics (continued)

TRAu	Total Reuse from Ancestors unitary
TRAp	Total Reuse from Ancestors percentage
TRDu	Total Reuse in Descendants unitary
TRDp	Total Reuse in Descendants percentage
VOD	Violations of Demeters Law
WMPC1	Weighted Methods Per Class 1
WMPC2	Weighted Methods Per Class 2

Table 57 C++ Audits

AUVK	Always Use “Virtual” Keyword
CA	Complex Assignments
CDPMD	Call Delete on Pointer Members in Destructors
CIUCFL	Complex Initialization or Update Clauses in ‘for’ Loops
CFPT	Comparing Floating Point Types
CLE	Complex Loop Expressions
CSC	C-Style Casting
CVFCD	Calling Virtual Functions from Constructors and Destructors
DOCE	Declare One-argument Constructors Explicit
EBWB	Enclosing Body Within a Block
ECB	Empty Catch Blocks
FVAL	Functions with Variable Argument Lists
HIA	Hiding Inherited Attributes
HISM	Hiding Inherited Static Methods
HN	Hiding Names
LF	Long Files
LL	Long Lines
MA	Memory Allocation
MVDWSN	Multiple Visible Declarations With Same Name
NC	Naming Conventions
ONAMAM	Overriding a Non-Abstract Method with an Abstract Method
RO	Related Operators
OWS	Overloading with a Subclass
PPA	Public and Package Attributes
RINF	Redefining an Inherited Nonvirtual Function
UVBC	Using Virtual Base Class
UVD	Use Virtual Destructor

Table 58 Metrics supported for Visual Basic 6

AC	Attribute Complexity
AHF	Attribute Hiding Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling
LOC	Lines Of Code
MHF	Method Hiding Factor
MNOP	Maximum Number Of Parameters
NOA	Number Of Attributes
NOC	Number Of Classes
NOCON	Number of Constructors
NOM	Number Of Members
NOO	Number Of Operations
PFriM	Percentage of Friend Members
PPrivM	Percentage of Private Members
PPubM	Percentage of Public Members
TCR	True Comment Ratio
WMPC2	Weighted Methods Per Class 2

Table 59 Metrics supported for C#

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF**	Attribute Inheritance Factor
CBO	Coupling Between Objects
CC	Cyclomatic Complexity
CDBC	Change Dependency Between Classes
CF	Coupling Factor
CR	Comment Ratio
FO	FanOut
DAC	Data Abstraction Coupling
DOIH	Depth Of Inheritance Hierarchy
HDiff	Halstead Difficulty
HEff	Halstead Effort
HPLen	Halstead Program Length
HPVoc	Halstead Program Vocabulary
HPVol	Halstead Program Volume

Table 59 Metrics supported for C# (continued)

LOC	Lines Of Code
LOCOM1	Lack of Cohesion of Methods 1
LOCOM2	Lack of Cohesion of Methods 2
LOCOM3	Lack of Cohesion of Methods 3
MHF	Method Hiding Factor
MIC	Method Invocation Coupling
MIF**	Method Inheritance Factor
MNOL	Maximum Number of Levels
MNOP	Maximum Number Of Parameters
MSOO	Maximum Size of Operation
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOCON	Number Of Constructors
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
NOprnd	Number of Operands
NOprtr	Number of Operators
NORM	Number of Remote Methods
NUOprnd	Number of Unique Operands
NUOprtr	Number of Unique Operators
PF**	Polymorphism Factor
PIntM *	Percentage of Internal Members
PPIntM *	Percentage of Protected Internal Members
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members
RFC	Response for Class
TCR	True Comment Ratio
TRAu	Total Reuse from Ancestors unitary
TRAp	Total Reuse from Ancestors percentage
TRDu	Total Reuse in Descendants unitary
TRDp	Total Reuse in Descendants percentage
VOD	Violations of Demeters Law
WMPC1	Weighted Methods Per Class 1

Table 59 Metrics supported for C# (continued)

WMPC2	Weighted Methods Per Class 2

Notes:
* This metric is specific for C# only.
** SCI cannot work with compilation units. Therefore, if a project consists of several compilation units, and some members have *internal* or *internal protected* access modifier, this metric produces the wrong result.

Metrics supported for Visual Basic .NET

The metrics supported for Visual Basic .NET are the same as those listed in Table 59, “Metrics supported for C#” with two exceptions:

- PFriM (Percentage of Friend Members) replaces the metric PIntM (Percentage of Internal Members).
- PPFriM (Percentage of Protected Friend Members) replaces the metric PPIntM (Percentage of Protected Internal Members).

Running audits and metrics from the command line

It is possible to run audits or metrics from the command line. This is useful for including quality assurance analysis as part of an automated daily build or other process. This section documents the command-line syntax and options.

Usage

The following syntax runs QA modules in console mode (that is, without the GUI).
TgStarter -script:com.togethersoft.modules.qa.QA [options] PrjName

Where:

TgStarter is one of the following:

%TGH%\bin\Together.bat
%TGH%\bin\TogetherCon.exe
%TGH%\bin\Together.exe -con

PrjName is a fully qualified project name, such as:

%TGH%\samples\java\CashSales\CashSales.tpr

NOTE: Specify paths above using either a slash or backslash, as required by your operating system.

Options

Table 60 Command Line Options

Name	Description
-?,-h,-help	Print this usage message and exit
-audit [out:file] [sort:[-]column] [cfg:optset]	Run audit process with specified parameters file = output file (default %TgHome%/out/audit) column = [column name]. If the column name is prefixed by a dash (-), it specifies the reverse order. severity abbreviation (default) explanation element item file line optset = file containing the previously saved set of options (default is current.adt).
-metrics [out:file] [sort:[-]column] [cfg:optset]	Run metrics with the specified parameters file = output file (default is %TgHome%/out/metrics) column = [column name]. If the column name is prefixed by a dash (-), it specifies the reverse order. abbr (metric abbreviation) item (default) optset = file containing the previously saved set of options (default is current.mts).
-pkg [pkg1 [pkg2 [...]]]	Process specified packages only
-cls [cls1 [cls2 [...]]]	Process specified classes only
-fmt:[tab align html report loadable]	Output format: tab = separate columns by tabs (default) align = align columns with spaces html = generate HTML file report = create HTML report (for metrics only) loadable = create loadable metrics results file (*.mtbl)
-dcpy:[directory]	Target folder where the descriptions are copied. This option is only valid for HTML format.

Table 60 Command Line Options (continued)

Name	Description
-dref:[<i>html-ref</i>]	Reference to this folder in HTML file. This option is only valid for HTML format.
-cmt:[<i>comment</i>]	Comment for loadable metrics result file (<i>comment</i> is the text to display as the table title). This option is only valid for loadable format.

Sets of options have the default extensions `.adt` for audits and `.mts` for metrics. If the set file name contains no path, it is searched for in the current directory first and then in the directory specified in project options (Options | Project Level | QA: *Path to current adt/mts files*). If it is not found in either of these directories, it is searched for in the directory where settings are stored by default (`%TGH%/modules/com/togethersoft/modules/qa/config`).

NOTE: QA plug-ins reside in `$TGH$/modules/com/togethersoft/modules/qa`. You can find detailed API for the QA plug-ins in `$TGH$/modules/com/togethersoft/modules/qa/doc`.

Additional information sources

Shyam R. Chidamber and Chris F. Kemerer, *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, 20(6), pp476-493, 1994.

Thomas J. McCabe, *Complexity Measure*. IEEE Transactions on Software Engineering, Volume 2, No 4, pp 308-320, December 1976.

Arthur H. Watson and Thomas J. McCabe, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-0001, September 1996.

Halstead, *Elements of Software Science*. New York, Elsevier North-Holland, 1977.

Brian Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, December 1995.

Object-Oriented Metrics: an Annotated Bibliography: <http://dec.bournemouth.ac.uk/ESERG/alpha.html>

J2EE DEVELOPMENT

- Chapter 26, “J2EE Support”
- Chapter 27, “Designing and Developing EJB Modules”
- Chapter 28, “Working with Container-Transaction Elements and References”
- Chapter 29, “Designing and Developing Web Applications”
- Chapter 30, “Designing and Developing Enterprise Applications”
- Chapter 31, “Designing and Developing Application Clients”
- Chapter 32, “Designing and Developing Resource Adapters”
- Chapter 33, “J2EE Platform Security Support”
- Chapter 34, “Deploying J2EE”

CHAPTER 26

J2EE SUPPORT

This chapter includes the following topics:

- “Supported J2EE technologies” on page 461
- “J2EE-supported diagrams” on page 462
- “References support” on page 464
- “Security support” on page 465

Supported J2EE technologies

Information security and safety, rapid and safe client-server connections, effective implementation of business logic, and high-quality GUI design are the main requirements for e-commerce applications. Using Together ControlCenter, you can create servlets, JSP files, EJBs, Web applications, and Enterprise applications, and then deploy them to the current application server.

In multi-tier client-server applications, developers usually use EJBs to link to databases and to implement business logic. HTML pages, JSPs, servlets, or applets are used for interfaces. A software product can include EJBs, servlets, JSPs, applets, HTML pages, GUI files, Java classes, and so on. The user application archive is placed on an application server that supports access, connection to, and operation of the client. J2EE technologies provide successful and safe transactions.

The most time-consuming tasks involve creating a deployment descriptor and performing the deployment process. Together simplifies these tasks: now it is possible to generate a deployment descriptor *.xml file and perform deployment automatically.

E-commerce features are implemented as a module that can be activated or deactivated. The e-commerce module is activated in the configuration by default. If it is not activated in your configuration, choose **Tools | Activate/Deactivate Features** on the main menu and check **E-Commerce**.

You can familiarize yourself with the J2EE specification at <http://java.sun.com/j2ee>. You can deploy your applications from a Class, Web Application, EJB Assembler, or Enterprise Application diagram.

Together automates deployment of the following components:

- Web files (HTML, GIF, and so on)
 - Servlets and JSP files
 - EJBs
 - Classes required for Servlets, JSPs and EJBs
- Create and deploy EJB modules, Web applications, and Enterprise applications using Together.

J2EE-supported diagrams

The J2EE-supported diagrams are not specified among the canonical UML diagrams. They are specific to Together as a part of the J2EE specification support.

Together provides the following types of diagrams supporting the J2EE platform:

- EJB Assembler diagram
- Web Application diagram
- Enterprise Application diagram
- Taglib diagram
- Application Client diagram
- Resource Adapter diagram

EJB Assembler diagram

An EJB Assembler diagram enables visual creation of a JAR archive that stores all EJBs (entity, session, and message-driven beans) and additional classes required for the EJBs (exceptions, utility classes, and so on).

To develop an EJB module, start by creating an EJB Assembler diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose EJB Assembler.)

Alternatively, open the EJB Module Expert for creating an EJB module. In this case the default EJB Assembler diagram automatically creates. For instructions on how to create this diagram and use it for developing EJB modules, refer to “Designing and Developing EJB Modules” on page 466.

Use the J2EE Deployment Expert for generating a deployment descriptor and deploying an EJB module to the current application server.

Web Application diagram

A Web Application diagram enables visual creation of a deployment descriptor and a Web application archive (WAR) that stores all JSPs, servlets, taglibs, classes, and Web files.

To develop a Web application, start by creating a Web Application diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose Web Application.)

Alternatively, open the Web Application Expert for creating a Web application (in this case the default Web Application diagram is created automatically). For instructions on how to create this diagram and use it for developing Web applications, refer to “Designing and Developing Web Applications” on page 526.

Use the J2EE Deployment Expert for generating a deployment descriptor and deploying a Web application to the current application server.

Enterprise Application diagram

An Enterprise Application diagram enables you to combine as many WARs and JARs as you want to include in an enterprise application archive (EAR). The Enterprise Application diagram can also be used for modeling global security constraints via Security Roles used for EJB/WAR modules.

To develop an enterprise application, start by creating an Enterprise Application diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose Enterprise Application.)

Alternatively, open the Enterprise Application Expert for creating an enterprise application (in this case the default Enterprise Application diagram is created automatically). For instructions on how to create this diagram and use it for developing enterprise applications, refer to “Designing and Developing Enterprise Applications” on page 558.

Use the J2EE Deployment Expert for generating deployment descriptors and deploying an enterprise application to the current application server.

Taglib diagram

A Taglib diagram helps you use and develop JSP tag libraries. In addition to the standard actions, JSP v1.1 technology supports the development of reusable modules called custom actions. Custom actions provide for using a custom tag in a JSP page. A tag library is a collection of custom tags. Use a Taglib diagram for solving different tasks including message managing, accessing databases, and so on.

To develop a tag library or to use an existing one, start by creating a Taglib diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose Taglib.) For instructions on how to create this diagram and use it for developing tag libraries, refer to “Taglib diagram” on page 463.

Shortcuts to Taglib diagrams can be placed on a Web Application or Enterprise Application diagram for later deployment.

Application Client diagram

Application clients are client tier programs that execute in users’ Java virtual machines. Application clients are invoked at their main method and run until the virtual machine is terminated. Application clients depend on a container to provide system services.

To develop an application client or to use an existing one, start by creating an Application Client diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose Application Client.) For instructions on how to create this diagram and use it for developing tag libraries, refer to Chapter 31, “Designing and Developing Application Clients” on page 564.

Shortcuts of Client Application diagrams can be placed on an EJB Assembler or an Enterprise Application diagram for later deployment.

Resource Adapter diagram

A Resource Adapter diagram is used for integrating J2EE applications with Enterprise Information Systems (EISs) that are not relational databases. A Resource Adapter diagram is developed for connector architecture that defines a standard Service Provider Interface (SPI) for integrating the transaction, security, and connection management facilities of an application server with those of a transactional resource manager.

Resource adapter elements provide access to any resources anywhere.

A Resource Adapter diagram enables visual creation of a RAR archive that stores all elements from this diagram.

To develop a resource adapter, start by creating a Resource Adapter diagram. (Click the **New Diagram** button on the diagram toolbar, activate the Together tab, and choose Resource Adapter.) For instructions on how to create this diagram and use it for developing tag libraries, refer to Chapter 32, “Designing and Developing Resource Adapters” on page 569.

Use the J2EE Deployment Expert for generating a deployment descriptor and deploying a resource adapter to the current application server.

References support

Together supports the following types of references:

- EJB references (references to other EJBs)
- Security references (references to possible user groups) with different access rights
- Resource references (references to possible resources)
- Environment references (references to constants in the environment)

For EJB 2.0 specification Together supports additional types of references:

- EJB local references (references to other EJBs located in the same container)
- Resource-environment references

Information about references is saved in a deployment descriptor. An EJB can request the current status of transactions, security, links between EJBs, and so on, via the descriptor context. To use the references mechanism, you should define the corresponding reference as an additional attribute to the class.

To add references, choose **New | EJB Reference** on the right-click menu.

Alternatively, you can add references and make changes on the Reference tab in the EJB Properties Inspector.

For instructions on how to work with references, refer to Chapter 28, “Working with Container-Transaction Elements and References” on page 510.

Security support

Together supports J2EE platform security mechanisms providing security policies in an operational environment.

Almost every J2EE-oriented diagram contains visual design elements connected with security.

- EJB Assembler diagram: *Security Role, Method Permission, EJB Security Reference*
- Web Application diagram: *Security Role, EJB Security Reference*
- EJB Assembler diagram: *Security Role*
- Resource Adapter diagram: *Security Permission, Authentication Mechanism*
- Application client diagram: *EJB Security Reference*

Security information from these diagrams is included in the deployment descriptor, which is generated during the deployment process.

For instructions on how to work with references, refer to Chapter 33, “J2EE Platform Security Support” on page 574.

CHAPTER 27

DESIGNING AND DEVELOPING EJB MODULES

This chapter discusses the following topics:

- “Visual assembly of EJB applications for deployment” on page 466
- “Creating EJB Modules” on page 467
- “Creating and developing EJBs” on page 471
- “Entity beans” on page 481
- “Session beans” on page 494
- “Message-driven beans” on page 501
- “Creating EJB shortcuts” on page 504
- “Verification and Correction of EJBs” on page 506
- “How an EJB Assembler diagram relates to the J2EE Deployment Expert” on page 507
- “Tips for the EJB Assembler diagram” on page 508

Visual assembly of EJB applications for deployment

Enterprise JavaBeans (EJBs) can be used in multi-tier distributed applications in which the business logic is usually implemented as a set of EJBs.

An EJB module joins EJBs in a group, and the current application server works with this group and creates one *.jar file for the EJB module.

In terms of Together design elements, the term *EJB module* is the same as the term *EJB Assembler diagram*.

The Together EJB Assembler diagram (or EJB module) enables users to model the way EJBs are assembled into an EJB application. With the help of the EJB Assembler

diagram, users can combine several EJBs in a JAR and deploy it to an application server. EJBs are added to the EJB Assembler diagram as shortcuts from Class diagrams.

The EJB Assembler diagram supports the J2EE platform: EJB references, local EJB references, security references, resource references, resource-environment references, and environment references. You can show method permissions for the business methods and link these to security roles that you define. All references are implemented as separate visual design elements. This method is useful for linking EJBs, resources, and other elements, and for providing reusability.

To create an EJB Application:

1. Create all necessary Class diagrams that contain EJBs.
2. Create a new EJB Assembler diagram.
3. Create shortcuts to the necessary EJBs and add shortcuts to the EJB Assembler diagram.
4. Define EJB References and other references as EJB attributes.
5. Add shortcuts to the necessary classes.
6. Define references as separate visual elements.
7. Define principals (users or groups) and add visual elements to the EJB Assembler diagram.
8. Define security roles as visual elements.
9. Draw links for the references.
10. Draw links for the security roles.
11. Define container elements.
12. Draw all the necessary links.
13. Invoke J2EE Deployment Expert.

For instructions on how to use security elements, refer to Chapter 33, “J2EE Platform Security Support”.

For instructions on how to create EJB references, refer to Chapter 28, “Working with Container-Transaction Elements and References”.

For instructions on how to deploy EJB modules, refer to “How an EJB Assembler diagram relates to the J2EE Deployment Expert” on page 507.

Creating EJB Modules

The EJB Assembler diagram is a visual equivalent of the *Application Assembler* described by the EJB 2.0 Specification. According to the specification, the assembler “assembles enterprise beans into a single deployment unit.” It provides application assembly information to the *Application Deployer*, which is represented in Together by the J2EE Deployment Expert. Refer to page 432 of the EJB 2.0 Specification for details.

There are two ways to create an EJB module:

- Creating a new EJB Assembler diagram.
- Creating a new EJB module using the Object Gallery.

Creating a new EJB Assembler diagram

To create an *EJB Assembler diagram*:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *General* in the categories on the left, and select the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Activate the Together tab, select the *EJB Assembler diagram* icon, and click **Finish**.

Alternatively, click the New Diagram button on the diagram toolbar, go to the Together tab and select EJB Assembler.

You should see the new EJB Assembler diagram with the default name *EJB Assembler*. To change this name, use the Inspector (right-click menu | **Properties**).

Develop the content of the diagram using visual design elements from the toolbar or using new components from the Object Gallery.

Creating a new EJB module using the Object Gallery

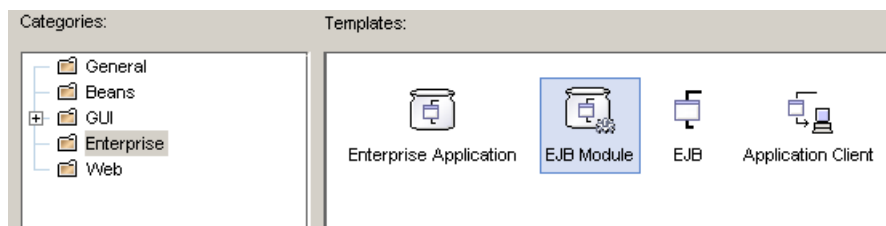
Use the Object Gallery to create a new EJB module (EJB Assembler diagram).

To create an *EJB module*:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Enterprise* in the categories on the left.
3. Select the *EJB Module* icon in the templates on the right. Click **Next**.

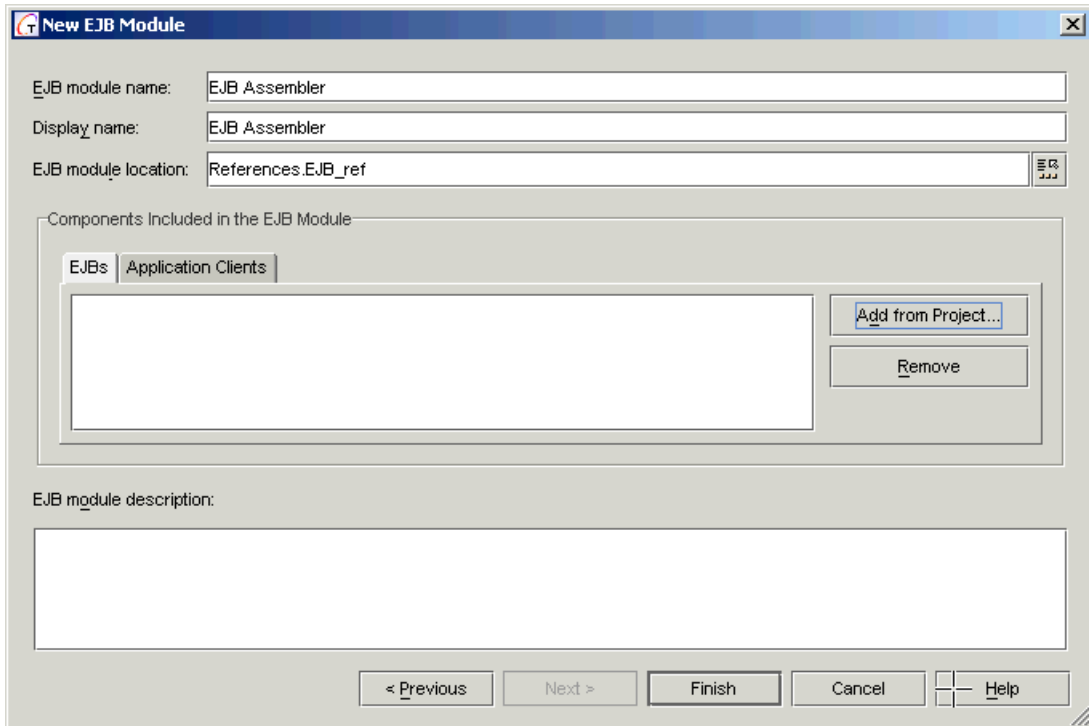
The resulting dialog box is an expert that guides you through the process of creating the EJB module. You do not have to understand how to use the EJB Assembler diagram because it is created automatically. For example, to combine several EJBs in the same module for deploying, go to EJB Module in the Object Gallery and follow the recommendations, filling in the most important parameters. This procedure is particularly helpful if you have only limited experience in the field of EJB technology.

Figure 130



4. Fill in the **New EJB Module** page in the expert. Set the name of the new EJB Assembler diagram and its location, the name of this diagram in the deployment descriptor, and add the main elements (such as EJBs or an application client) that you want to include in the module. See online help for details.

Figure 131



5. Click **Finish**. This creates the EJB Assembler diagram that represents your EJB module.

Visual design elements

All EJB Assembler visual components are represented in the toolbar on the left side of the diagram.

An EJB Assembler diagram contains the following design elements:

Table 61 EJB Assembler diagram design elements

Icon	Description
	Zoom: Magnifies the diagram. Access the Zoom tool by the right-click menu (or NumPad+). To zoom out, use the Zoom Out command on the right-click menu (or NumPad-).
	EJB Shortcuts: Opens a selection manager dialog box. Add or remove EJB shortcuts on the EJB Assembler diagram using the selection manager.

Table 61 EJB Assembler diagram design elements (continued)

Icon	Description
	Security Role: Defines the security role that stands for one of the recommended security roles for the EJB's clients. A Security Role design element in an EJB Assembler diagram presents a simplified view of the EJB application's security to the application deployer. This is used in the J2EE Deployment Expert.
	Principal: Creates a visual design component that represents a User or a Group of Users separated from their <i>Security Role</i> . Note: It is possible to change the stereotype using in-place editing.
	Method Permission: Shows the required permissions on one or more methods. Represents a binary relation between the security roles and the methods of EJB home and remote interfaces. Represents permission required to invoke methods on these interfaces.
	EJB Properties: Creates a visual design component representing EJB properties as a separate element with its own properties. Use this design element for simplifying a visual representation or for changing these properties.
	Container Transaction: Defines transaction attributes for the methods of EJB home and remote interfaces. When linked to one or more EJB methods, it specifies that the linked methods are assigned the transaction attribute value defined in the Container Transaction element's properties. All methods linked to a single Container Transaction element should belong to a single bean.
	EJB Reference: Creates an element with its own properties. In the visual model, it represents a reference to the corresponding EJB. Defines a remote type for EJB reference: EJB, or EJB local (for EJB 2.0 specification).
	Environment Reference: Creates a visual design component with the properties of some static constant, which cannot be changed after EJB deployment.
	Resource Reference: Creates a visual design component that has all properties of the referenced resource. Define the type of reference: resource, or resource-environment (for EJB 2.0 specification).
	Bundled Files: Creates a visual design component that has its own properties. Add or remove Web files on the EJB Assembler diagram using the selection manager.
	Message Web Service: Creates a message Web service for BEA WebLogic 6.1.

Table 61 EJB Assembler diagram design elements (continued)

Icon	Description
	Assembly Link: Drawn between diagram elements to show their relationships in the application.
	Note: Creates a visual Note element.
	Note Link: Creates a link between the note element and another visual component to show the note's relationship with this element.

Key elements and properties

Some of the key elements cannot be added to the diagram with the toolbar icons. An EJB Assembler diagram contains only EJB shortcuts (it cannot contain the EJBs themselves). These EJBs already exist in the EJB module. The next section explains how to create EJBs.

Create a shortcut to each EJB so show EJBs in the Web Application diagram. To include Home and Remote interfaces of an EJB, add the corresponding shortcuts to this EJB in the Web application diagram.

Notation

The EJB Assembler diagram is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

Having created this diagram, open the J2EE Deployment Expert to generate a deployment descriptor and deploy an application to the selected server.

Creating and developing EJBs

You create and develop EJBs in the context of a Together project. There are three ways you can create EJBs in a project:

- Create a Together project for existing EJB source code.
- Create a basic model and a code skeleton for a new EJB using the “one-click” EJB feature in Class diagrams. You then can specify EJB properties and add EJB fields, methods, and so on in the EJB Inspector.
- Create an EJB element using the Object Gallery.

This section explains how to use each of these techniques.

Creating a project using existing EJB code

If you have existing EJB source code, create one or more Together projects around it using Together reverse engineering. Together creates visual diagrams when it reverse

engineers the code, and then keeps the visual model synchronized with subsequent changes to the code, and vice versa. Once the code is part of a Together project, you can quickly and easily generate up-to-date documentation.

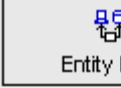
Creating a project from existing EJB code is no different from creating any other kind of Java project. Before creating a project, check what EJB specification the existing EJBs satisfy, and select the corresponding application server in Project Properties on the EJB tab. During the reverse engineering process, the corresponding visual EJB components are created on the Class diagram.

Creating “one-click” EJBs

To create a “one-click” EJB:

1. Create or open a Together Project, and create or navigate to the package where you want to create an EJB.
2. Create or open a Class diagram in the target package.
If your Together license enables EJB support, the Class diagram displays three EJB toolbar buttons: Entity EJB, Session EJB, and Message-Driven EJB.
3. On the Class diagram toolbar, click the Entity EJB button, Session EJB button, or Message-Driven EJB button to create the appropriate type of EJB.
4. Click the diagram background to generate the EJB design element.

Table 62 EJB visual design elements

Icon	Description
 Entity EJB	Creates the corresponding design element in the visual model and generates the underlying source code for a default implementation of a persistent entity EJB with skeleton declarations. See section “Entity beans” on page 481 to learn how to use the Inspector for customizing properties.
 Session EJB	Creates elements in the visual model and generates the underlying source code for a default implementation of a nonpersistent session EJB with skeleton declarations. Learn how to use the Inspector for customizing properties in the section “Session beans” on page 494.
 MessageDriven EJB	Creates elements in the visual model and generates the underlying source code for a default implementation of a message-driven EJB with skeleton declarations. Learn how to use the Inspector for customizing properties in the section “Message-driven beans” on page 501.

No properties or Business Methods are declared. There are several ways to add them:

- Add them visually in the diagram, using the **New** submenu on the implementation class’ right-click menu.
- Add them visually using the EJB Inspector (see “Configuring EJBs using EJB Inspectors” on page 478).

The resulting implementation icon displays the name of the created bean and its type, compartments for the attributes, operations, home and remote interfaces, and primary key class. Linked interfaces are listed under the bean name.

Use *EJB - Show* options in the Options dialog to populate the diagrams with only the necessary elements. You can optionally show or hide implementation classes, home/remote interfaces and the primary key classes. However, it is still possible to view the source code of hidden elements in the Editor pane. All elements, even those hidden in the Designer pane, are displayed in the Explorer and can be opened for editing.

Creating EJBs using the Object Gallery

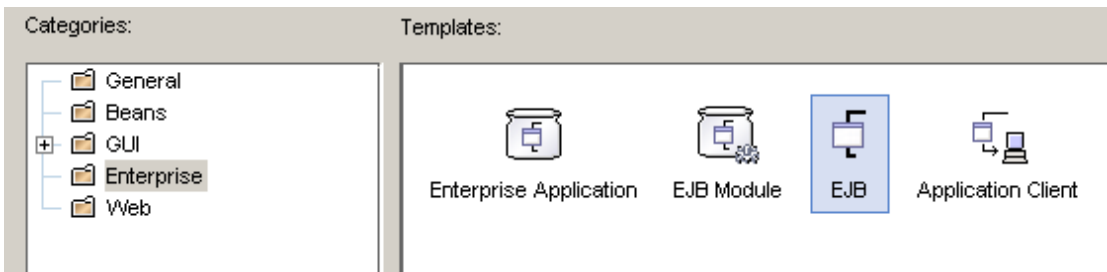
Use the Object Gallery to create EJBs.

To create a new EJB using the Object Gallery:

1. Choose **File | New** on the main menu.
2. Select the *Enterprise* node.
3. Select the *EJB* icon, and click **Next**.

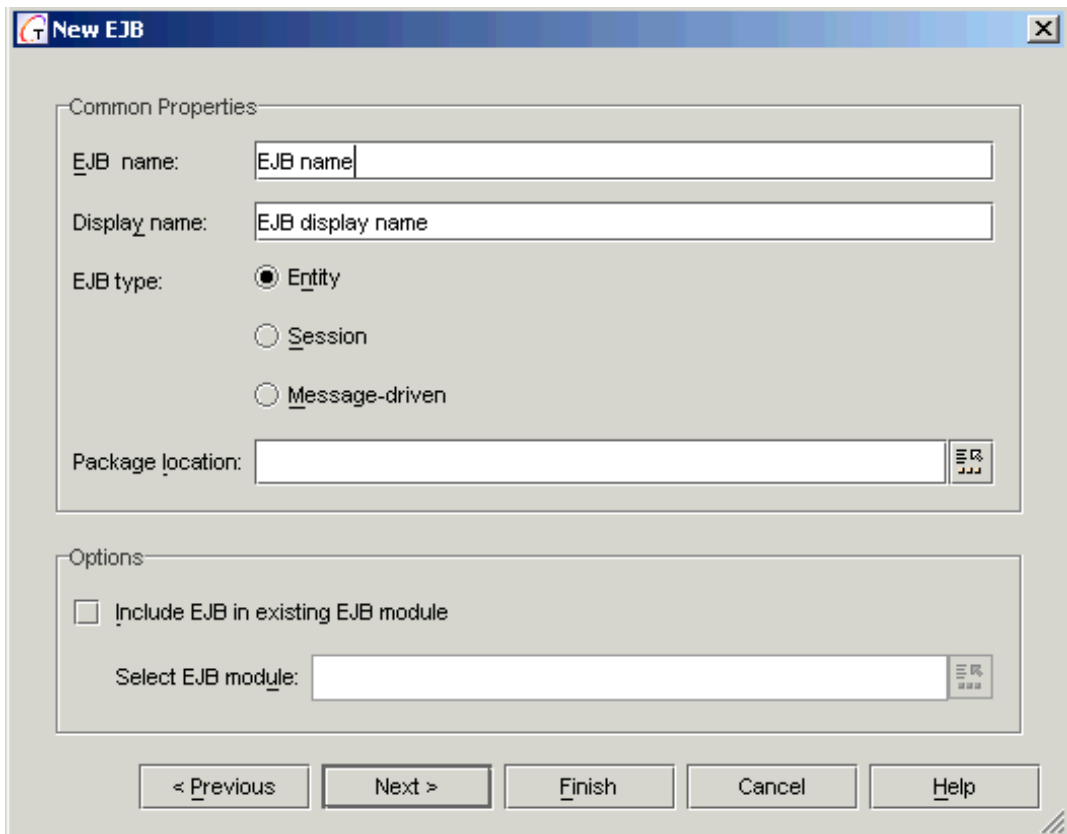
The resulting dialog box is an expert that guides you through the process of creating the EJB. Follow its recommendations, filling in the most important EJB parameters. Insert names or choose paths. This procedure is particularly helpful if you have only limited experience in creating EJBs.

Figure 132



4. In the expert, define the common EJB properties: the name of the EJB to display on the diagram and the name to use in the deployment descriptor, the location of the EJB, and its type (session, entity, or message-driven). Include this EJB in an existing EJB module. See online help for details.

Figure 133



The image shows a 'New EJB' dialog box with a title bar containing a logo and the text 'New EJB'. The dialog is divided into two main sections: 'Common Properties' and 'Options'. In the 'Common Properties' section, there are four fields: 'EJB name' with the text 'EJB name', 'Display name' with the text 'EJB display name', 'EJB type' with three radio buttons ('Entity' selected, 'Session', and 'Message-driven'), and 'Package location' with an empty text box and a browse button. The 'Options' section contains a checkbox labeled 'Include EJB in existing EJB module' which is unchecked, and a 'Select EJB module:' field with an empty text box and a browse button. At the bottom of the dialog are five buttons: '< Previous', 'Next >', 'Finish', 'Cancel', and 'Help'.

New EJB

Common Properties

EJB name: EJB name

Display name: EJB display name

EJB type: ☒ Entity
☐ Session
☐ Message-driven

Package location: [Browse]

Options

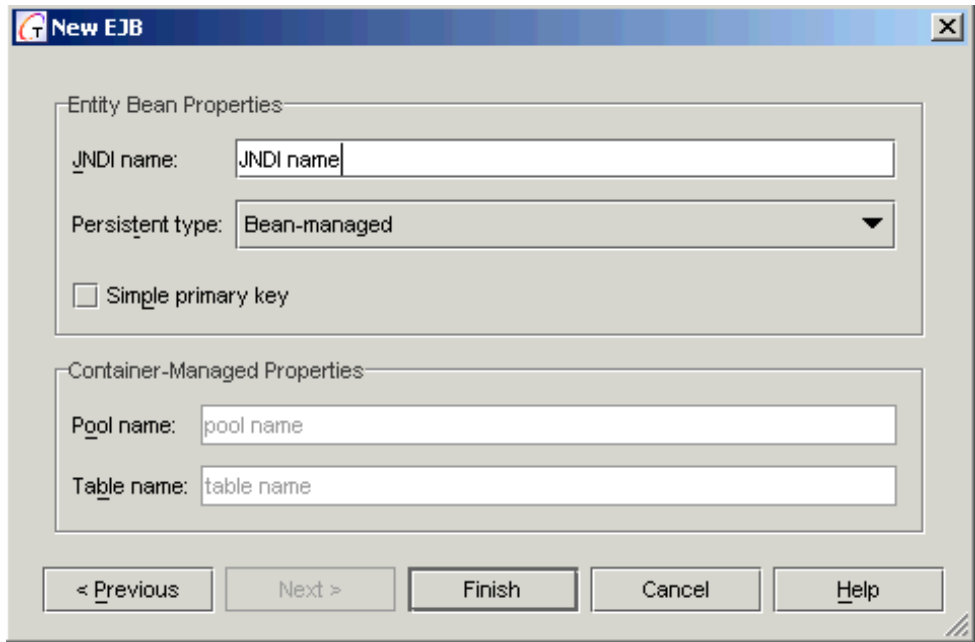
☐ Include EJB in existing EJB module

Select EJB module: [Browse]

< Previous Next > Finish Cancel Help

5. Click **Finish** if you do not want to set type-specific properties for your EJB. Otherwise, click **Next** to continue.
6. Set the EJB type-specific fields on the next page. For example, for a container-managed EJB, set the JNDI name, the persistence type, the pool name, and the table name. See online help for details.

Figure 134

The image shows a 'New EJB' dialog box with a blue title bar. It is divided into two main sections: 'Entity Bean Properties' and 'Container-Managed Properties'. In the 'Entity Bean Properties' section, there is a text field for 'JNDI name' containing 'JNDI name', a dropdown menu for 'Persistent type' set to 'Bean-managed', and an unchecked checkbox for 'Simple primary key'. The 'Container-Managed Properties' section contains two text fields: 'Pool name' with 'pool name' and 'Table name' with 'table name'. At the bottom, there are five buttons: '< Previous', 'Next >', 'Finish' (which is highlighted with a dark border), 'Cancel', and 'Help'.

7. Click **Finish** to create the EJB.

Local interfaces

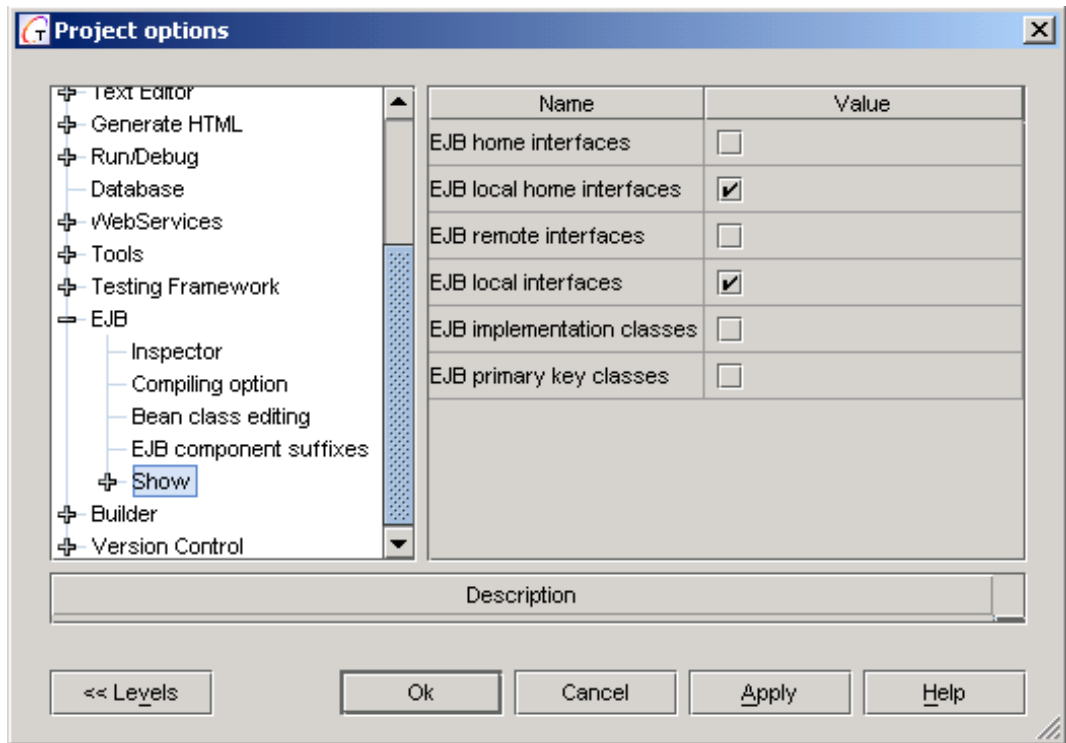
If the EJB satisfies the EJB 2.0 specification, it has a local interface.

EJB local home interfaces and EJB local interfaces are hidden by default.

Include certain interfaces in the set of components that are displayed. Choose **Tools | Options | Default Level** or **Project Level**. In the Options dialog, expand *View Management - Show - EJB*, and select the necessary boxes.

If they are checked in the Options dialog, EJB home interfaces, EJB remote interfaces, EJB implementation classes, and EJB primary key classes are added to the Class diagram automatically.

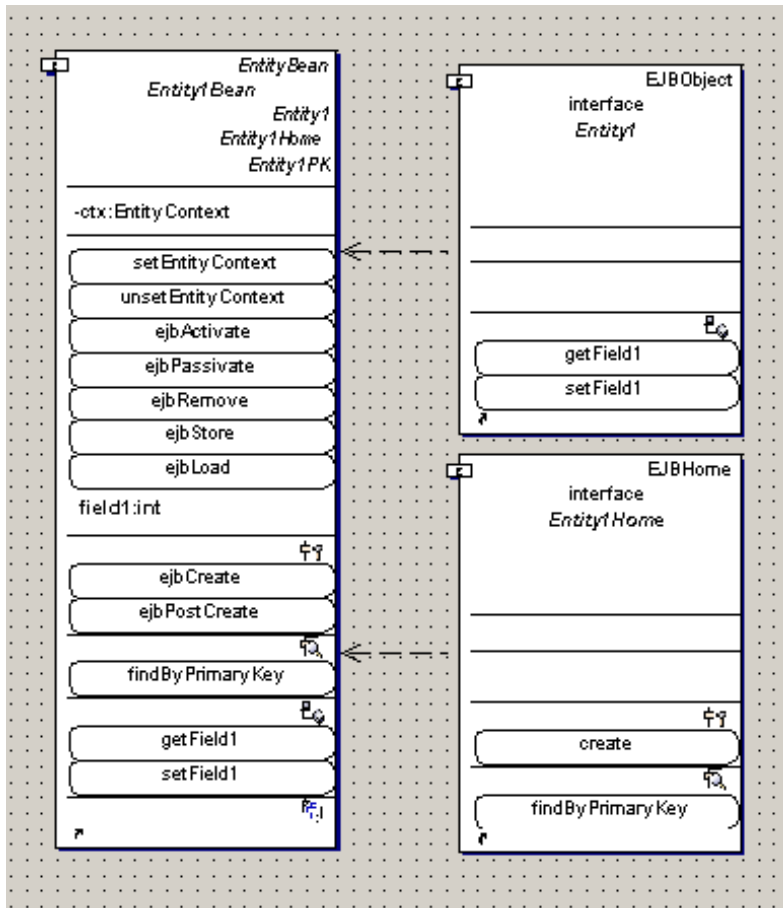
Figure 135



EJB local and local home interfaces, on the other hand, have to be created on the EJB after checking them in the Options dialog.

To view EJB local and EJB local home interfaces, focus on the EJB visual element, and choose **New | Create Local Interfaces** on the right-click menu. Local interfaces with links are added to the Class diagram.

Figure 136



Customizing the default code for EJBs

The default code generated by one-click EJBs is an adequate starting point for most developers. However, the default code is customizable by modifying the appropriate code template. Customize the templates for the entity bean class, session bean class, message-driven bean class, primary key class, home interface, remote interface, local home interface, and local interface.

Find the main EJB templates in the following directories:

- For EJB 1.0 specification: %TG_Home%/templates/JAVA/CLASS/EJB10
- For EJB 1.1 specification: %TG_Home%/templates/JAVA/CLASS/EJB11
- For EJB 2.0 specification: %TG_Home%/templates/JAVA/CLASS/EJB20

NOTE: Customizing templates is an advanced procedure that is not necessary in most cases. If you need to make changes to template files, please use caution.

Configuring EJBs using EJB Inspectors

Once you have a skeleton created by the one-click feature, use the EJB Inspector to edit its properties.

To open the EJB Inspector:

1. Select the EJB in the diagram or the Explorer.
2. Choose **Properties** on the right-click menu to display the EJB Inspector.

Alternatively, press **Alt+Enter**.

Develop EJBs visually using the EJB Inspector for bean classes. The Inspector has pages common to all classes but adds additional pages specifically for working with EJBs:

- For entity EJBs, pay attention to the *Entity EJB* page.
- For session EJBs, pay attention to the *Session EJB* page.
- For message-driven EJBs, pay attention to the *Message-driven EJB* page.

The type-specific pages display a lower tabset that provide the means of specifying general EJB properties, adding and removing fields and methods, defining references, relationships, specifying bean type-specific properties, deployment properties, and so on. The visual appearance of the EJB Inspector varies, depending on the bean type and the current application server (EJB specification).

Before working with the EJB Inspector, be sure you are working with the appropriate EJB specification.

To set the EJB specification:

1. Choose **Tools | Options | Default Level** or **Project Level** on the main menu.
2. In the Options dialog, expand the *EJB* node.
3. Check the *Application Server* setting and change it if necessary.

Mandatory subtabs

In the Inspector, there are three mandatory subtabs for the Entity and Session EJB elements:

- General
- EJB references
- Deployment properties

For the Message-Driven element there are only two mandatory subtabs:

- Deployment properties
- EJB references

Optional subtabs

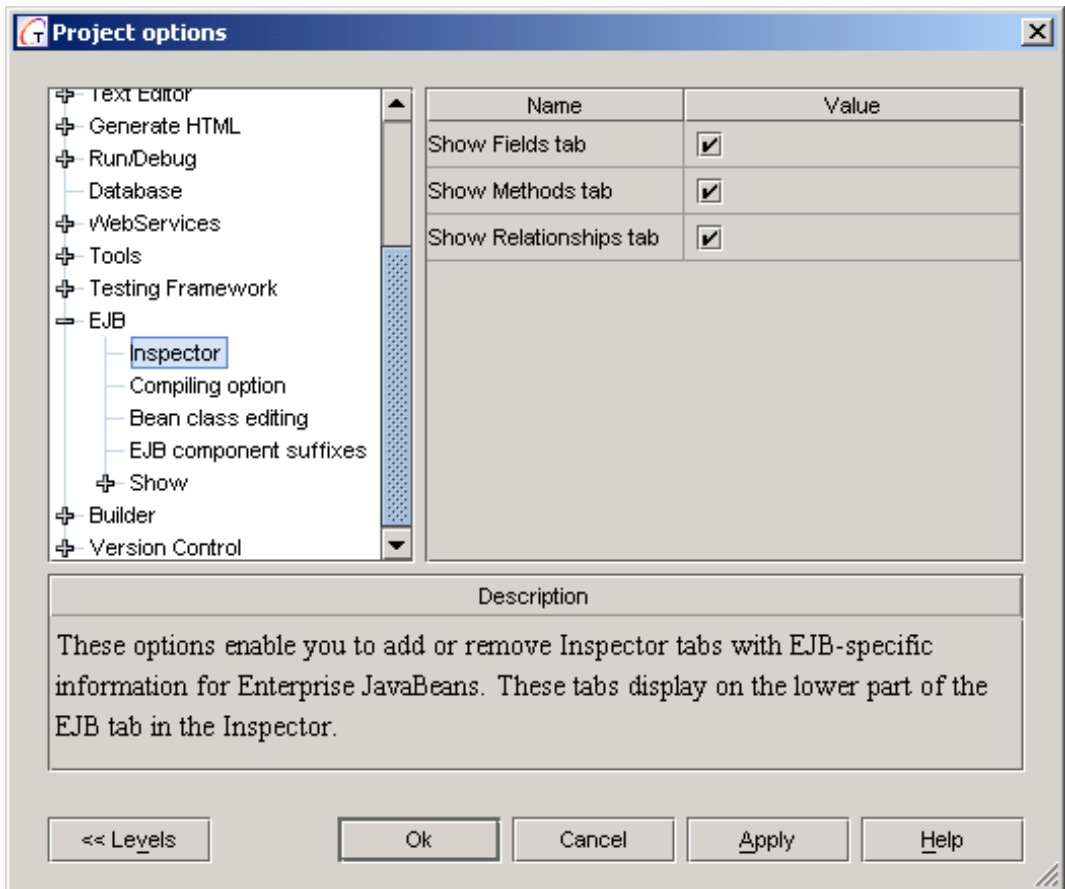
Optionally add the following subtabs to the Inspector:

- Fields
- Methods

- Relationships

To add the tabs, choose **Tools | Options | Default Level or Project Level - EJB - Inspector**.

Figure 137



Select the *Fields*, *Methods*, or *Relationships* checkboxes to show the corresponding subtabs on the EJB page of the EJB Inspector.

Sharing home and remote interfaces

You can optionally share the same set of home and remote interfaces between two or more EJB implementation classes of the same type (session or entity). You may want to use this capability in cases where you need to deploy differing implementations of the same interfaces on different servers.

Creating a second implementation class

In a case of shared interfaces, develop one complete implementation with home and remote interfaces. Create a second implementation class, refactor it into an EJB implementation, and specify the home and remote interfaces from the first bean in the properties of the new one.

To create a second implementation class:

1. Create or open a Class diagram to show the new class.
2. Create a new class in the diagram and name it as desired.

To refactor the class to an EJB implementation:

1. Right-click on the class icon to open the class menu.
2. Point to **Apply EJB Pattern**.
3. Choose **Entity**, **Session**, or **Message-Driven** bean from the submenu.

Now begin developing the implementation, or proceed to specify the home and remote interfaces for the new bean class using the home and remote interfaces from the first implementation.

To reuse the first implementation's home and remote interfaces:

1. In the property Inspector for the new class, select the *xxx EJB* page (where *xxx* is either *Session* or *Entity* depending on the type of EJB).
2. Select **Do not synchronize names**.
3. In the *Remote name* field, click the browse button to open the Select Element dialog.
4. Use the **Model** node of the tree-view to locate and select the remote interface belonging to the first implementation (that is, the remote interface that you want the bean you are working on to share).
5. Click **OK** to accept the selection. Respond *Yes* to the *Change Class* prompt.
6. Follow the same procedure in the *Home name* field, selecting the home interface for the first implementation in the Select Element dialog.

To show the second implementation class in a separate Class diagram:

If the second implementation class is in the same Class diagram as the first, implementation links are automatically drawn from the interfaces to the second implementation class. If the two implementations are in different Class diagrams, you may want to show the second implementation class in the Class diagram for the first implementation.

To show a linked second implementation class:

1. Select the remote interface for the first implementation class.
2. Choose **Add Linked** from the interface right-click menu. Linked elements are shown in the Add Linked tab of the Message pane.
3. Select the second implementation class in the Message pane, right-click and choose **Add**.

A Class icon for the second implementation class is added to the diagram. You can now work on the second implementation class from either Class diagram.

Deleting implementation classes with shared interfaces

Normally, if you delete an EJB implementation class, home and remote interfaces are deleted along with the implementation class, both in source code and in the diagram. In cases where interfaces have been shared with another implementation class, deleting elements from the diagram does not automatically result in deletion of the relevant source code files - only the visual diagram elements are deleted.

You have to explicitly delete implementation classes and home/remote interfaces by selecting them in the Model tab of the Explorer and choosing Delete from the right-click menu.

NOTE: You can find an animated demoguide at www.togethercommunity.com

Entity beans

Before creating an entity EJB, make sure that the appropriate application server (or EJB specification) is selected in Project Properties. To create an entity EJB, select the *Entity EJB* icon on the Class diagram toolbar and click on the diagram pane. This creates a visual component that contains the following:

- EJB implementation class with:
 - Entity context attribute
 - One default field (int type)
 - Default set of method declarations:
 - set & unset context methods
 - ejbActivate & ejbPassivate methods
 - ejbRemove method
 - ejbStore & ejbLoad methods
 - ejbCreate & ejbPostCreate methods
 - findByPrimaryKey method
 - getField & setField methods
- EJB home interface with:
 - ejbCreate method signature
 - findByPrimaryKey method signature
- EJB remote interface
 - getField & setField methods signature
- EJB primary key class with:
 - Default field (int type)
 - primary key, hash code, and equals methods
- EJB local home interface with:
 - create method signature
 - findByPrimaryKey method signature

- EJB local interface with:
 - getField & setField methods signature
- Dependency links

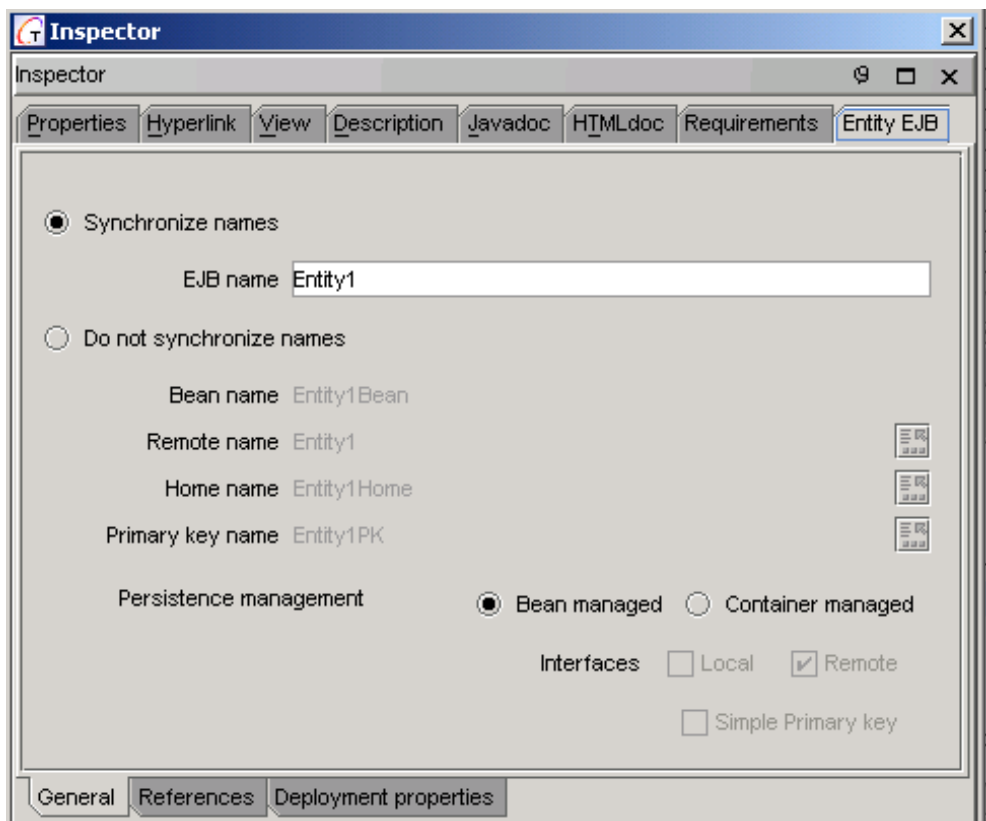
Entity bean Inspector

In the EJB Inspector for the entity bean, select the Entity EJB tab, which has a set of specific tabbed pages.

General subtab

The General tab is the page with general EJB properties. On the General page, choose between bean-managed and container-managed persistence, and between synchronized and non-synchronized names. If this entity bean has a local interface, check *Local* (Interfaces).

Figure 138



Local interfaces are disabled if this EJB does not satisfy the EJB 2.0 Specification.

General properties of the entity EJB

The *Synchronize names* / *Do not synchronize names* radio buttons are set to *Synchronize names* by default.

The edit field *EJB name* is always shown. Other edit fields are enabled only when *Do not synchronize names* is selected.

The edit fields connected with local interfaces (*Local name* and *Local Home name*) and the corresponding *Interface Local* checkbox are enabled only when you are working with EJBs satisfying the EJB 2.0 specification (that is, if you choose Generic 2.0, BEA WebLogic 6.0™, or BEA WebLogic 6.1™ as the current application server).

If you develop your EJB project using the EJB 2.0 specification, select *Remote* and *Local* interfaces. It is required to select either one or both of them (this way the component is not interpreted as a bean). *Remote* is selected by default.

Persistence management displays *Bean managed* by default.

Primary key

There are two ways to specify the primary key class for an entity bean with container-managed persistence:

- Primary key maps to a single field of the EntityBean class.
- Primary key maps to multiple fields of the EntityBean class.

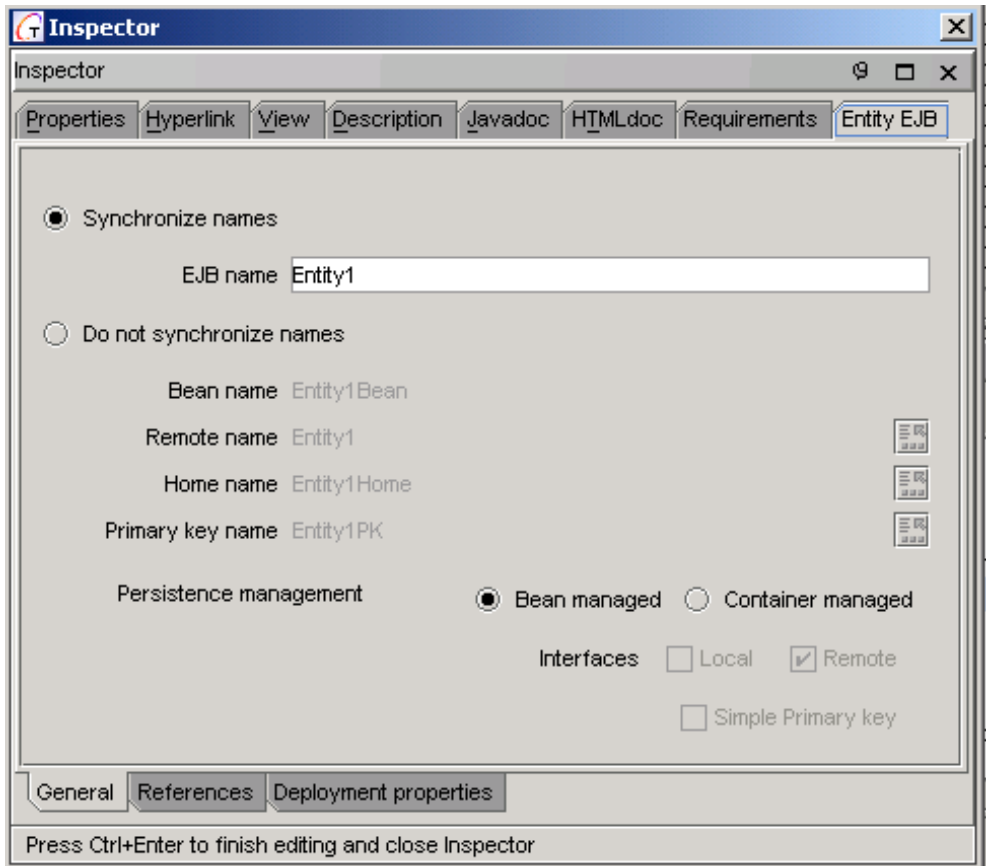
To create a primary key of the first type, check the box *Simple primary key*, which is enabled for container-managed beans. This helps avoid wrapping of the simple type primary key (such as String) into a user-defined class.

The second method is helpful for implementation of compound keys.

Simple primary key is enabled and is checked by default, if there is not any primary key field with the appropriate type in the bean (Integer, String, Long, or some other primary key type).

References subtab

Figure 139



References subtab enables Together users to view existing references and to create new ones. Choose the type of reference using the radio buttons: *EJB*, *EJB Local* (for EJB 2.0 specification only), *Resource*, *Resource-environment* (for EJB 2.0 specification only), *Environment*, *Security role*. The content of the table depends on the selected reference type.

EJB references

Add a new EJB reference or view the list of existing EJB references in the table. This page displays the following properties of EJB references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *EJB reference name*: the specific name to use for look-up.

- *EJB reference*: the type of referenced EJB (session or entity); if you select the existing object using *Remote interface* or *Home interface* paths, *EJB reference* is filled in automatically.
- *Remote interface* field with a path chooser: enables you to select the existing remote interface of the EJB.
- *Home interface* field with a path chooser: enables you to select the existing home interface of the EJB.
- *EJB link*: a parameter in the deployment descriptor.
- *EJB JNDI name*: a parameter in the deployment descriptor.

EJB local references

The *EJB local* radio button is visible only if the EJB satisfies the EJB 2.0 specification. Add a new EJB local reference or view the list of existing EJB local references in the table. This page displays the following properties of EJB local references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *EJB reference name*: the specific name to use for look-up.
- *EJB reference*: the type of the referenced EJB (session or entity); if you select the existing object using *Remote interface* or *Home interface* paths, *EJB reference* is filled in automatically.
- *Local interface* field with a path chooser: enables you to select the existing local interface of the EJB.
- *Local home interface* field with a path chooser: enables you to select the existing local home interface of the EJB.
- *EJB link*: a parameter in the deployment descriptor.

Select the corresponding radio button to choose the type of references, and then click **Add** to create a new reference. To remove a reference, select the corresponding line of the table and then click **Remove**.

Resource references

Add a new EJB resource reference or view the list of existing EJB resource references in the table. This page displays the following properties of EJB resource references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *Resource reference name*: the specific name to use for look-up.
- *Resource reference type*: the type of a resource reference (javax.sql.DataSource / javax.jms.QueueConnectionFactory / javax.jms.TopicConnectionFactory / javax.sql.XADataSource / javax.mail.Session / javax.net.URL).
- *Resource reference authentication*: enables you to choose *Application* or *Container*.
- *Resource JNDI name*: a parameter in the deployment descriptor.

Resource-environment references

The *Resource-environment* radio button is visible only if the EJB satisfies the EJB 2.0 specification. Use this page to reference the container (local) resources.

Add a new EJB resource-environment reference or view the list of existing EJB resource-environment references in the table. This page displays the following properties of EJB resource-environment references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *Resource reference name*: the specific name to use for look-up.
- *Resource reference type*: the type of a resource-environment reference (javax.sql.DataSource / javax.jms.QueueConnectionFactory / javax.jms.TopicConnectionFactory / javax.sql.XADataSource / javax.mail.Session / javax.net.URL).

Environment references

Add a new EJB environment reference or view the list of existing EJB environment references in the table. This page displays the following properties of EJB environment references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *Environment entry name*: the specific name to use for look-up.
- *Environment entry type*: the type of an environment reference (java.lang.Boolean / javax.lang.String / javax.lang.Integer / javax.lang.Double / javax.lang.Byte / javax.lang.Short / javax.lang.Long / javax.lang.Float).
- *Environment entry value*: the concrete value of the entry.

Security role references

Add a new EJB security role reference or view the list of existing EJB environment references in the table. This page displays the following properties of EJB environment references (and the corresponding columns in the table):

- *Name*: the name of the corresponding attribute in the EntityBean class.
- *Role name*: the specific name to use for look-up.
- *Role link*: an attribute in the deployment descriptor.

Deployment properties subtab

On this page, define the main entity EJB properties required for the deployment process and used in the corresponding deployment descriptor.

Figure 140

The Inspector window displays the configuration for an Entity EJB. The 'Entity EJB' tab is selected, showing a table of properties. The 'Transaction attribute' is set to 'NotSupported'. The 'Reentrant' checkbox is unchecked. The 'User/group personal IDs', 'Display name', 'Small icon', and 'Large icon' fields are empty and have associated icons on the right.

Name	Value
Bean name in application	Entity1Bean
JNDI name	Entity1Home
Local JNDI name	Entity1Bean
Pool name	ejbPool
Schema name	
Table name	Entity1
Transaction attribute	NotSupported
Reentrant	☐
User/group personal IDs	
Display name	
Small icon	
Large icon	

General References Deployment properties

Press Ctrl+Enter to finish editing and close Inspector

External access options

- *Bean name in application* has the same default meaning as *Bean name* on the General page.
- *JNDI name* has the same default setting as *Home name* on the General page.
- *Local JNDI name* is enabled only if the EJB satisfies EJB 2.0 specification; this parameter has the same default meaning as *Local Home name* on the General page.

Database options

- *Abstract schema name* is disabled for BMP beans and is enabled for CMP beans.
- *Pool name* is disabled for BMP beans and is enabled for CMP beans.
- *Schema name* is disabled for BMP beans and is enabled for CMP beans.
- *Table name* is disabled for BMP beans and is enabled for CMP beans.

General options

- *Transaction attribute* has the following possible settings: NotSupported / Supports / Mandatory / Required / RequiresNew / Never.
- *Reentrant* should be checked if the current entity EJB allows reentrant calls.
- *User/group personal IDs* field with a multi-string editor: this optional parameter enables you to insert personal identifiers.

Additional options

- *Display name*: an optional parameter for the deployment descriptor.
- *Small icon*: an optional field with a file chooser for selecting a small icon file. Use this optional parameter to select the small icon appearance for the application server's icon element. In this case, you have to specify the location of a small (16x16 pixels) *.gif or *.jpg image used to represent the Web application in the GUI tool.
- *Large icon*: an optional field with a file chooser for selecting a large icon file. Use this optional parameter to select the large icon appearance for the application server's icon element. In this case, you have to specify location of a large (32x32 pixels) *.gif or *.jpg image used to represent the Web Application in the GUI tool.

Fields subtab

By default, the Fields subtab is hidden. Add a field using New | Field on the right-click menu, and change it using the Inspector.

To see all existing fields in the table simultaneously, check the *Fields* box in the Options dialog (see “Optional subtabs” on page 478).

In the case of a CMP entity bean, this page displays the following properties of EJB fields (and the corresponding columns in the table):

- *Type*: the type of the field (String / boolean / byte / char / double / float / int / long / short / java.util.Hashtable / java.util.Vector).
- *Name*: the name of the field in the EJB.
- *DB field name*: the name of the field in the database. This column is hidden for BMP beans.
- *Label*: a parameter for representation in JSP client.
- *Primary key*: checked if this field is used as a primary key.
- *Relationship*: checked if this field is used in a relationship. This column is hidden for BMP beans.

Click **Add** to add a new field. To remove a field, select it and click **Remove**.

Methods subtab

By default, the *Methods* subtab is hidden. Add a method using New on the right-click menu and then choosing the type of method and modify it using the Inspector.

To view all existing methods of a group in the table simultaneously, check the *Methods* box in the Options dialog (see “Optional subtabs”).

Choose the necessary group of methods using the radio buttons. The content of the table on this page depends on the current type of method selected.

The types of methods are:

- Creators
- Finders
- Business methods
- Home methods (should be disabled if not Entity 2.0 specification)
- Selectors (should be disabled if not entity CMP 2.0 bean)

According to the EJB 2.0 specification, all methods, except business and select methods, can be exposed either in a remote interface, or in a local interface. The *Expose in interface* checkbox is used for choosing between a remote or a local interface. Business methods can have both a remote and a local interface.

Select the radio button for choosing the necessary group of methods. Click **Add** to add a new method. To remove a method, select it and click **Remove**.

All changes in the edit fields are synchronized in the source code.

Create methods

If you choose *Creators*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: Method signature in the `EntityBean` class.
- *Exposed in interface*: the type of interface where this method is exposed (enabled only for EJB 2.0 Specification). The possible settings are *Exposed in remote* and *Exposed in local*.

Finder methods

If you choose *Finders*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the `EntityBean` class.
- *Multi*: checked if a finder can return many elements.
- *Finder query*: the corresponding query.

Business methods

If you choose *Business methods*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the `EntityBean` class.
- *Exposed in remote interface*: checked if a method is exposed in a remote interface.
- *Exposed in local interface*: checked if a method is exposed in a local interface.
- *Transaction attribute*: the possible settings are <bean default> NotSupported / Supports / Mandatory / Required / RequiresNew / Never.

Home methods

The *Home methods* radio button is visible only for Entity EJBs satisfying the EJB 2.0 specification (that is, if you selected *Generic 2.0*, *BEA WebLogic 6.0*, or *BEA WebLogic 6.1* as the current application server); otherwise it is not available.

If you choose *Home methods*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the EntityBean class.
- *Exposed in interface*: the type of interface where this method is exposed (enabled only for the EJB 2.0 Specification). The possible settings are *Exposed in remote* and *Exposed in local*.

Select methods

The *Select methods* radio button is visible only for CMP Entity EJBs satisfying the EJB 2.0 specification (that is, if you selected *Generic 2.0*, *BEA WebLogic 6.0*, or *BEA WebLogic 6.1* as the current application server); otherwise it is not available.

If you choose *Select methods*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the EntityBean class.
- *Finder query*: the corresponding query.
- *Exposed in interface*: the type of interface where this method is exposed (enabled only for EJB 2.0 Specification). The possible settings are *Exposed in remote* and *Exposed in local*.

Relationships subtab

Add a relationship using *New | Relationship* on the right-click menu and change it using the Inspector. By default, the *Relationships* subtab is hidden.

To see all existing relationships in the table simultaneously, check the *Relationships* box in the Options dialog (see “Optional subtabs” on page 478).

This subtab is available only for entity CMP EJBs satisfying the EJB 2.0 specification.

- *CMR field name*: the name of the container-managed relationship in the EntityBean class.
- *Relationship name*: a unique name for a relationship.
- *Role name*: an optional name for the relationship role.
- *FK column*: a foreign key name in the database.
- *Key column*: a key field name in the database.
- *Multiplicity*: multiplicity of the role. Possible settings are *One* and *Many*.
- *Relationship destination*: the path to the related EJB element.

Creating a bean-managed entity EJB

This section explains how to create a BMP bean using visual design elements. Use the Object Gallery to also create BMP beans .

To create a bean-managed entity EJB:

1. Decide which EJB specification and application server to use. For demonstration purposes, use the EJB 2.0 specification and BEA WebLogic 6.1 Application Server.
2. Before creating a bean-managed persistence entity bean, create a new project. For example, create a new project with the name **BMP_Entity** in the directory %TG_HOME%/myprojects/BMP_Entity.
3. Set the EJB properties of your project: choose BEA WebLogic Application Server 6.1 (your future EJBs will satisfy the EJB 2.0 specification automatically).
4. Create a package on the default diagram using the package visual component. Double-click to open this package to see a Class diagram. On the Class diagram toolbar, click the **Entity EJB** button, and place it on the diagram. Define the EJB name for this bean.

Open the **EJB Inspector** (right-click menu | **Properties**), open the **Entity EJB** tab, and go to the **General** subtab. The general properties of the BMP bean are already defined correctly by default, and even `int field1` is defined as a primary key.

5. Next, define the necessary fields.

In your database, create the table with necessary fields. The names of the database and the table will be used for connection with the database.

NOTE: Use some means for connection pooling, such as JDBC API.

Create the same fields in the entity BMP EJB.

To add, correct, or remove fields, choose **New | Field** on the right-click menu, use the field's Inspector, or use the optional Field subtab in the EJB Inspector (see "Fields subtab" on page 488).

Now you have the BMP EJB with the same fields as in the database table.

6. Define a simple primary key. Define a field having a serialized type as a simple primary key. Check *Simple primary key* on the *General* tab, and define the selected field as a primary key (check the *Primary key* box for this field on the *Fields* subtab).
7. The skeletons are visible for the general EJB methods `setEntityContext()`, `unsetEntityContext()`, `ejbActivate()`, `ejbPassivate()`, `ejbStore()`, `ejbLoad()`, `ejbRemove()`, `ejbCreate()`, `ejbPostCreate()`, `ejbFindByPrimaryKey()`, and for accessor methods. Notice that the naming convention for the EJB 2.0 specification is satisfied. If you focus on some method in the bean, read the corresponding code in the Editor pane.
8. Add the code to this skeleton in place of the green comments "`// Write your code here.`" Your code should be written according to the selected EJB specification. In the case of a BMP Entity bean, write the content of the general EJB methods manually.
9. It is necessary to import some packages during implementation. Some of the import lines have already been added by Together. Add other lines during the coding process if required.

10. Create the necessary business methods, and edit the code if necessary. To create a method, choose **New | Business Method** on the right-click menu, then insert the name of the method and its properties, and add the code using the Editor. Note that you can first write the code, and the corresponding property will appear in the bean component.

Implement the methods that provide access to the database such as `getConnection()`.

11. If you use your own exceptions, create the corresponding class on the same Class diagram.
12. Create a simple class client that contains calls of the methods on the same Class diagram.
13. Now deploy the BMP entity bean to the current application server (for example, BEA WebLogic 6.1 Application Server) from the Class diagram. Create an EJB Assembler diagram, place the shortcut of the BMP entity bean on it, and invoke the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).
14. Run the client.

Creating a container-managed entity EJB

This section explains how to create a CMP bean using visual design elements. Use the Object Gallery to also create CMP beans (See “Creating EJBs using the Object Gallery”).

To create a container-managed entity EJB:

1. First, decide what EJB specification to use for implementation and select an application server for future deploying. For demonstration purposes, use the EJB 2.0 specification and BEA WebLogic 6.1 Application Server.
2. Before creating a container-managed persistence entity bean, create a new project. For example, create a project with the name `ContainerManaged` in the directory `%TG_HOME%/myprojects/CMP_Entity`.
3. Set the EJB properties of your project: choose BEA WebLogic Application Server 6.1 (this way your future EJBs will satisfy the EJB 2.0 specification automatically).
4. Create a package on the default diagram using the package visual component. Double-click to open this package to see the Class diagram. On the Class diagram toolbar, click the Entity EJB button, and place it on the diagram. Type a name in the *EJB name* field to define the name of this bean. Open the EJB Inspector (right-click menu | **Properties**), open the **Entity EJB** tab, and go to the General subtab. You should define general properties of the CMP bean: change the *Persistence management* property from *Bean-managed* to *Container managed*, and click **Yes** in answer to the question: “All ejbFind methods will be deleted, continue?” Unlike a BMP bean, a CMP bean does not contain implementations of finder methods. The skeletons are visible for the general EJB methods `setEntityContext()`, `unsetEntityContext()`, `ejbActivate()`, `ejbPassivate()`, `ejbStore()`, `ejbLoad()`, `ejbCreate()`, `ejbRemove()`, `ejbPostCreate()`, `findByPrimaryKey()`, and for accessor methods. Notice that the naming convention for the EJB 2.0 specification is satisfied, and skeletons of interfaces are correct for a CMP bean. For example, accessor methods are `public` and `abstract`. If you focus on some method in the bean, you can read the corresponding code in the Editor pane.

5. Next, create the table with necessary fields in your database. The name of the database and the name of the table will be used for connection with the database. To open the Generate DDL Expert, choose **Tools | Generate | DDL** on the main menu. In *Export from*, choose *Enterprise JavaBeans*. Choose the diagram name, and click **Next**. On the next page, check *Run*, and click **Add** to open an additional page. Enter settings in the connection profile. Note: Use some means for connection pooling, such as JDBC API. Test the connection with the database.
6. Before proceeding with deployment, you should create the database pool on the BEA WebLogic 6.1 Application Server, and provide the ODBC data source.
7. Create the same fields in the entity CMP EJB.

To add, correct, or remove fields, choose **New | Field** on the right-click menu, use the field's Inspector, or use the optional Field subtab in the EJB Inspector (see "Fields subtab" on page 488).

Now you have a CMP EJB with the same fields as in the database table.

8. On the Deployment Properties page, set the *Bean name in application*, the *JNDI name*, and database properties: *Abstract schema name*, *Pool name*, *Table name*.
The table name should be the same as the name of the table in the database.
9. The pool For example, if we take the example from J2EE specification, and define the Pool name like `jdbc/TogetherPool`, this pool should be already created on the application server. To define a simple primary key, check *Simple primary key* on the General tab, and define one of the fields having a serialized type as a simple primary key.

Define or correct field properties in the **Fields** subtab.

Define queries for the corresponding finder methods. To define the queries, open the Methods tab and write the necessary SQL queries in the *Finder query* field. For these queries, Together generates the lines in the deployment descriptor.

10. Add the code to this skeleton in place of the green comments "`// Write your code here.`" Your code should be written according to the selected EJB specification.
It is necessary to import some packages during implementation. Some of the import lines have already been done by Together. Add other lines during the coding process if required.
11. Create the necessary business methods, and edit the code if necessary. To create a method, choose **New | Business Method** on the right-click menu, then insert the name of the method and its properties, and add the code using the Editor. Note that you can first write the code, and the corresponding property will appear in the bean component.
12. Now deploy the CMP bean to the current application server (such as BEA WebLogic 6.1 Application Server) from the Class diagram. You can also create an EJB Assembler diagram, place the shortcut of the CMP entity bean on it, and invoke the J2EE Deployment Expert (see Chapter 34, "Deploying J2EE" on page 584).
13. Run the Client.

Session beans

Before creating a session EJB, make sure that the appropriate application server (or EJB specification) is selected in Project Properties. To create a session EJB, select the *Session EJB* icon on the Class diagram toolbar and click on the diagram pane. This creates a visual component that contains the following:

- EJB implementation class with:
 - Session context attribute
 - `setSessionContext` method
 - `Activate & Passivate` methods
 - `Remove` method
 - `ejbCreate` method
- EJB home interface with:
 - `Create` method, possibly with parameters
- EJB remote interface
 - Method signatures of additional methods intended for client use
- EJB primary key class with:
 - `Default` field (int type)
 - `primary` key, `hash code`, and `equals` methods
- EJB local home interface with:
 - `create` method signature
- EJB local interface with:
 - Method signatures of additional methods intended for client use
- Dependency links

Session bean Inspector

In the EJB Inspector for the session bean, select the Session EJB tab, which has a set of specific tabbed pages.

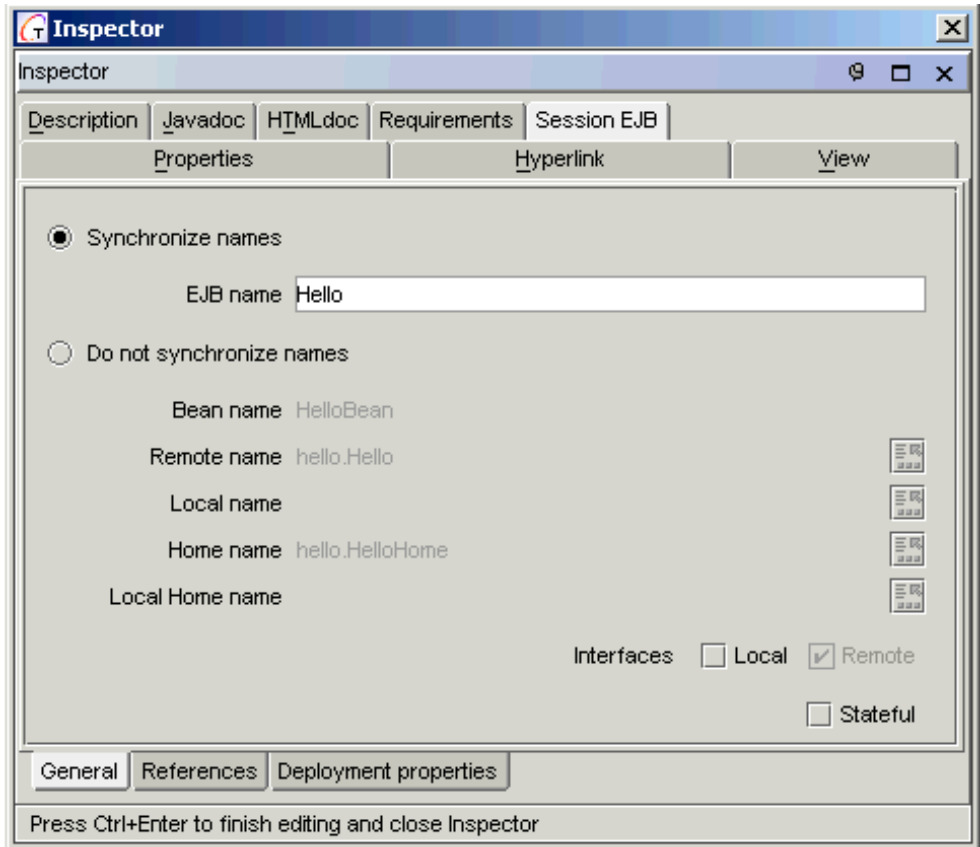
There are mandatory pages: General, References, and Deployment Properties.

You can also add the optional Methods tabbed page.

General subtab

The General tab is the page with general EJB properties. On the General page choose between stateless and stateful, and between synchronized and not synchronized names. If this session bean has a local interface, check *Local* (Interfaces).

Figure 141



Local interfaces are disabled if this session EJB does not satisfy the EJB 2.0 Specification.

General properties of the session EJB

The *Synchronize names* / *Do not synchronize names* radio buttons are set to *Synchronize names* by default.

The edit field *EJB name* is always shown. Other edit fields are enabled only when *Do not synchronize names* is selected.

The edit fields connected with local interfaces (*Local name* and *Local Home name*) and the corresponding Interface *Local* checkbox are enabled only when you are working with EJBs satisfying the EJB 2.0 specification (that is, if you choose Generic 2.0, BEA WebLogic 6.0, or BEA WebLogic 6.1 as the current application server).

To develop your EJB project using the EJB 2.0 specification, select *Remote* and *Local* interfaces. It is required to select either one or both of them (this way the component is not interpreted as a bean). *Remote* is selected by default.

Stateless or stateful session bean

Use the *Stateful* checkbox to define the type of session bean. Check the box if it is stateful.

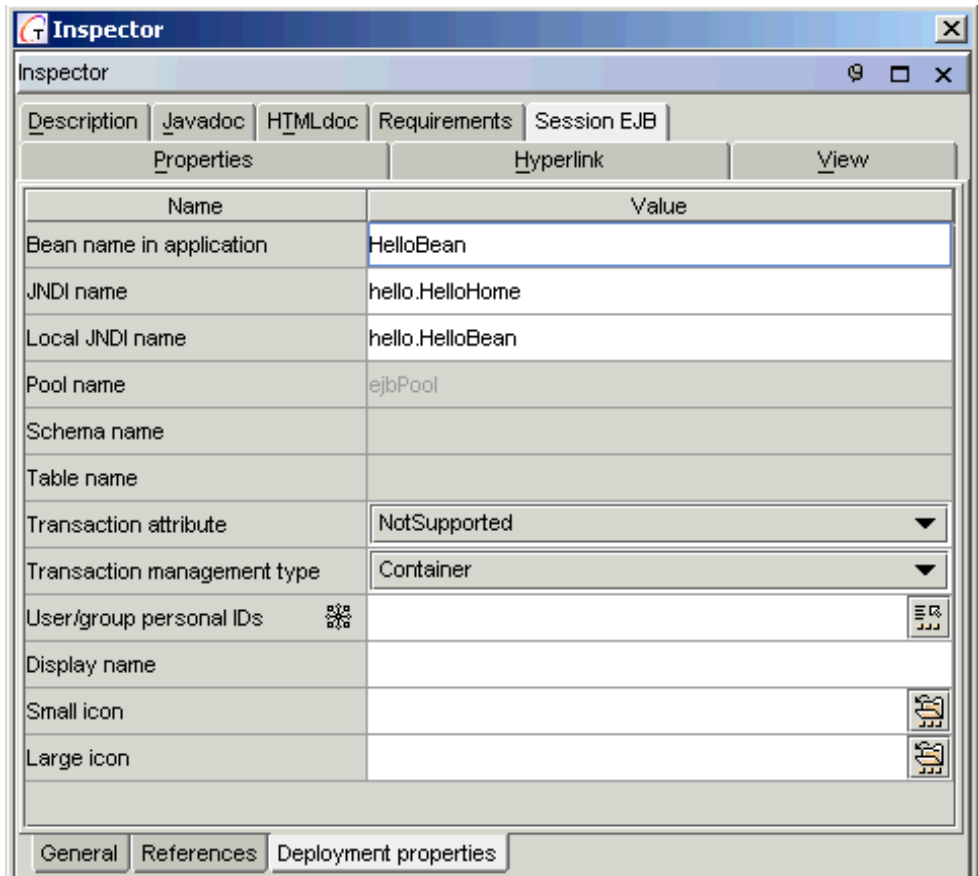
References subtab

This page enables you to view existing references and to create new ones. You can see the section “References subtab” for entity EJBs. This page is the same for a session bean.

- *Name*: the name of the corresponding attribute in the SessionBean class.
- The meanings of other parameters are the same as for an entity bean.

Deployment properties subtab

Figure 142



External access options:

- *Bean name in application* has the same default meaning as *Bean name* on the General page.
- *JNDI name* has the same default setting as *Home name* on the General page.
- *Local JNDI name*

General options:

- *Transaction attribute* has the following possible settings: NotSupported / Supports / Mandatory / Required / RequiresNew / Never .
- *Transaction management type* (Container or Bean)
- *User/Group personal IDs* field with a multi-string editor: this optional parameter enables you to insert personal identifiers.

Additional options:

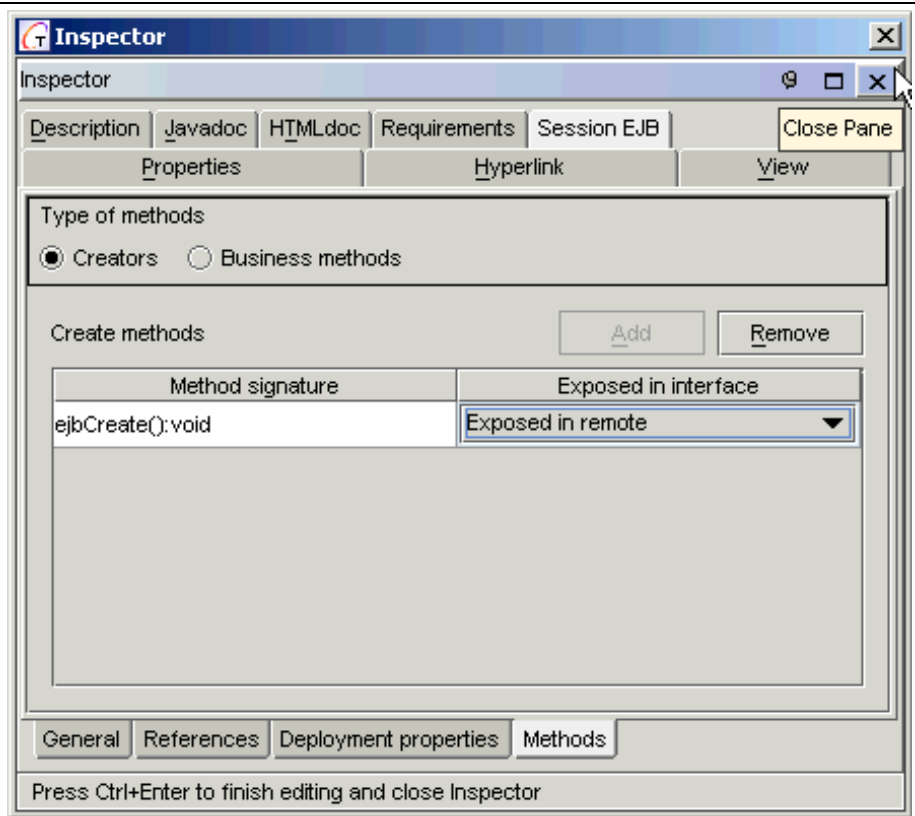
- *Display name*: an optional parameter for the deployment descriptor.
- *Small icon*: an optional field with a file chooser for selecting a small icon file.
- *Large icon*: an optional field with a file chooser for selecting a large icon file.

The different groups should be implemented as different tables.

Methods subtab

An optional tabbed page. To view this page, choose **Tools | Options | Default Level or Project Level - EJB - Inspector**.

Figure 143



Add a method using New on the right-click menu, choose the type of method, and change it using the Inspector. By default, the *Methods* subtab is hidden.

To see all existing methods of a group in the table simultaneously, check the *Methods* box in the Options dialog (see “Optional subtabs” on page 478).

Choose the necessary group of methods using the radio buttons. The contents of the table on this page depends on the current type of method selected.

The types of methods are:

- Creators
- Business methods

Select the radio button for the necessary group of methods. Click **Add** to add a new method. To remove a method, select it and click **Remove**.

All changes in the edit fields are synchronized in the source code.

Create methods

If you choose *Creators*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the EntityBean class.
- *Exposed in interface*: the type of interface where this method is exposed (enabled only for EJB 2.0 Specification). The possible settings are *Exposed in remote* and *Exposed in local*.

Business methods

If you choose *Business methods*, the table contains the following parameters (and the corresponding columns):

- *Method signature*: method signature in the EntityBean class.
- *Exposed in remote interface*: checked if a method is exposed in a remote interface.
- *Exposed in local interface*: checked if a method is exposed in a local interface.
- *Transaction attribute*: the possible settings are <bean default> / NotSupported / Supports / Mandatory / Required / RequiresNew / Never.

Creating a stateless session EJB

This section explains how to create a stateless session bean using visual design elements. Use the Object Gallery to also create session beans (“Creating EJBs using the Object Gallery” on page 473).

To create a stateless session EJB:

1. Decide which EJB specification and application server to use. For demonstration purposes, use the EJB 2.0 specification and BEA WebLogic 6.1 Application Server.
2. Before creating a stateless session bean, create a new project. For example, create a project with the name `Stateless_Session` in the directory `%TG_HOME%/myprojects/Stateless_Session`.
3. Set the EJB properties for your project: choose BEA WebLogic Application Server 6.1 (this way your future EJBs will satisfy the EJB 2.0 specification automatically).
4. Create a package on the default diagram using the package visual component. Double-click to open this package to see a Class diagram. On the Class diagram toolbar, click the Session EJB button, and place it on the diagram. Define the EJB name for this bean.

Open the EJB Inspector (right-click menu | **Properties**), open the **Session EJB** tab, and go to the **General** subtab. Uncheck *Stateful* to define a session bean as a stateless session bean.

5. The skeletons are visible for the general session EJB methods `setSessionContext()`, `ejbActivate()`, `ejbPassivate()`, `ejbCreate()`, `ejbRemove()`. Notice that the naming convention for EJB 2.0 specification is satisfied. If you focus on some method in the bean, read the corresponding code in the Editor pane. Add your code to implement the `ejbCreate()` method in place of the green comments “// Write your code here.” The `ejbCreate()` method of a stateless session bean does not have any variables.
6. Add the business methods to the session bean, write your code, and edit the code if necessary. To create a method, choose **New | Business Method** on the right-click menu, then insert the name of the method and its properties, and add the code using

the Editor. Note that you can first write the code, and the corresponding property will appear in the bean component.

7. Check that the home interface contains the `create()` method signature, and the remote interface contains method signatures of the business methods.
8. It is necessary to import some packages during implementation. Some of the import lines have already been done by Together. Add other lines during the coding process if required.
9. Create a client on the same Class diagram such as a simple class that contains calls of the methods.
10. Now deploy the stateless session bean to the current application server (such as BEA WebLogic 6.1 Application Server) from the Class diagram. Create an EJB Assembler diagram, place the shortcut of the stateless session bean on it, and invoke the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).
11. Run the Client.

Creating a stateful session EJB

This section explains how to create a stateless session bean using visual design elements. Use the Object Gallery to also create session beans (See “Creating EJBs using the Object Gallery” on page 473).

To create a stateful session EJB:

1. Decide which EJB specification and application server to use. For demonstration purposes, use the EJB 2.0 specification, and BEA WebLogic 6.1 Application Server.
2. Before creating a stateless session bean, create a new project. For example, create a project with the name `Stateful_Session` in the directory `%TG_HOME%/myprojects/Stateful_Session`.
3. Set the EJB properties for your project: choose BEA WebLogic Application Server 6.1 (this way your future EJBs will satisfy the EJB 2.0 specification automatically).
4. Create a package on the default diagram using the package visual component. Double-click to open this package to see a Class diagram. On the Class diagram toolbar, click the Session EJB button, and place it on the diagram. Then define the EJB name for this bean. Open the EJB Inspector (right-click menu | **Properties**), open the **Session EJB** tab, and go to the **General** subtab. Make sure that *Stateful* is checked (it is checked by default).
5. The remaining steps are almost the same as for a stateless session bean.

The difference is connected with the property of a stateful session bean to retain state on behalf of an individual client. For a stateful bean, you need to do the following:

 1. Define a serializable member variable that obeys the rules for conversation state.
 2. Define `ejbcreate(<parameter>)` method with an input parameter.
6. Create a client that can both passivate and activate the EJB during the same session.
7. Use the same procedure for deploying as for a stateless session bean (see Chapter 34, “Deploying J2EE” on page 584).

Using a session EJB as a Web service

If you created a session bean on the diagram, use it as a Web service in the BEA WebLogic 6.1 Application Server. Find all necessary information in “BEA WebLogic 6.1” on page 629.

Message-driven beans

Before creating a message-driven EJB, make sure that the application server supporting the EJB 2.0 specification or Generic 2.0 is selected in Project Properties. To create a message-driven EJB, select the *Message-Driven EJB* icon on the Class diagram toolbar and click on the diagram pane. This creates a visual component that contains the following:

EJB implementation class, with:

- MessageDriven bean context attribute
- setMessageDrivenContext method
- ejbCreate method
- ejbRemove
- ejbActivate and ejbPassivate methods
- onMessage method

NOTE: Message-driven beans are only used with the EJB 2.0 specification.

Message-driven bean Inspector

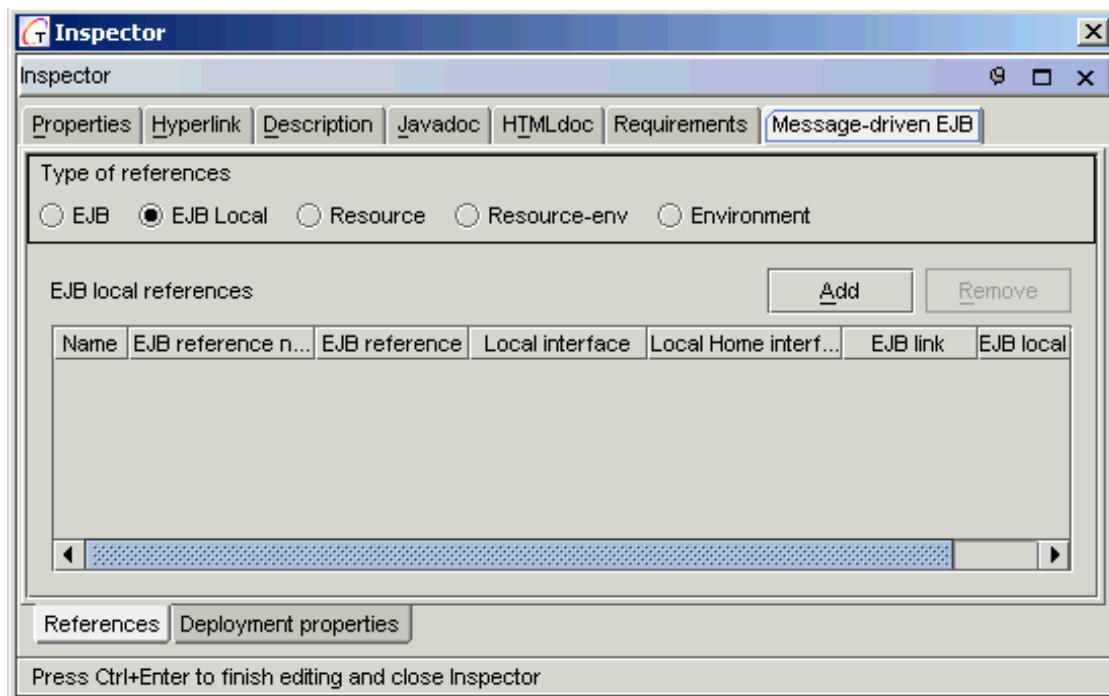
In the EJB Inspector for the message-driven bean, select the Message-driven EJB tab, which has a set of specific tabbed pages.

There are only two mandatory pages: References and Deployment properties.

References subtab

The References page is almost the same as for other EJBs, but there are only five types of references: EJB reference, EJB local reference, EJB resource reference, EJB resource-environment reference, and EJB environment reference.

Figure 144



The meanings of these references are also the same as for entity and session beans (see “References subtab” on page 484).

Deployment properties subtab

Figure 145

Name	Value
Bean name in application	MessageTraderBean
JNDI name to bind to	Message.MessageTraderBean
Transaction attribute	NotSupported
Transaction management type	Container
User/group personal IDs	
Acknowledge mode	Auto-acknowledge
Destination type	javax.jms.Queue
Destination JNDI name	
Subscription durability	Durable
Message selector	
Security identity	none
User caller identity	
Role name	
Display name	
Small icon	
Large icon	

External access options:

- *Bean name in application* has the same default meaning as *Bean name* on the General page.
- *JNDI name* has the same default setting as *Home name* on the General page.

General options:

- *Transaction attribute* has the following possible settings: NotSupported / Supports / Mandatory / Required / RequiresNew / Never.
- *Transaction management type* (Container or Bean)
- *User/Group personal IDs* field with a multi-string editor: this optional parameter enables you to insert personal identifiers.
- *Acknowledge node* is a deployment parameter that can have the following meanings: Auto-acknowledge, Dups-ok-acknowledge.
- *Destination type* is a deployment parameter that can have the following meanings: javax.jms.Queue, javax.jms.Topic.

Additional options:

- *Destination JNDI name*.
- *Subscription durability* is a deployment parameter that can have the following meanings: Durable, NonDurable.
- *Message selector*.
- *Security identity* is a deployment parameter that can have the following meanings: none, user-caller-identity, run-as-specified-identity.
- *User caller identity*
- *Role name*
- *Display name* - an optional parameter for the deployment descriptor.
- *Small icon* - an optional field with a file chooser for selecting a small icon file.
- *Large icon* - an optional field with a file chooser for selecting a large icon file.

Creating EJB shortcuts

By the time you are ready to assemble one or more EJBs into an application, you should have existing EJBs in your Together project. To use the EJB Assembler diagram to specify security roles, method permissions, and so on for deployment, place the relevant EJBs and other design elements on the EJB Assembler diagram by creating one or more shortcuts to the completed EJBs.

An EJB shortcut

A shortcut is just a visual representation of an element that resides somewhere else in the project.

To create an EJB shortcut:

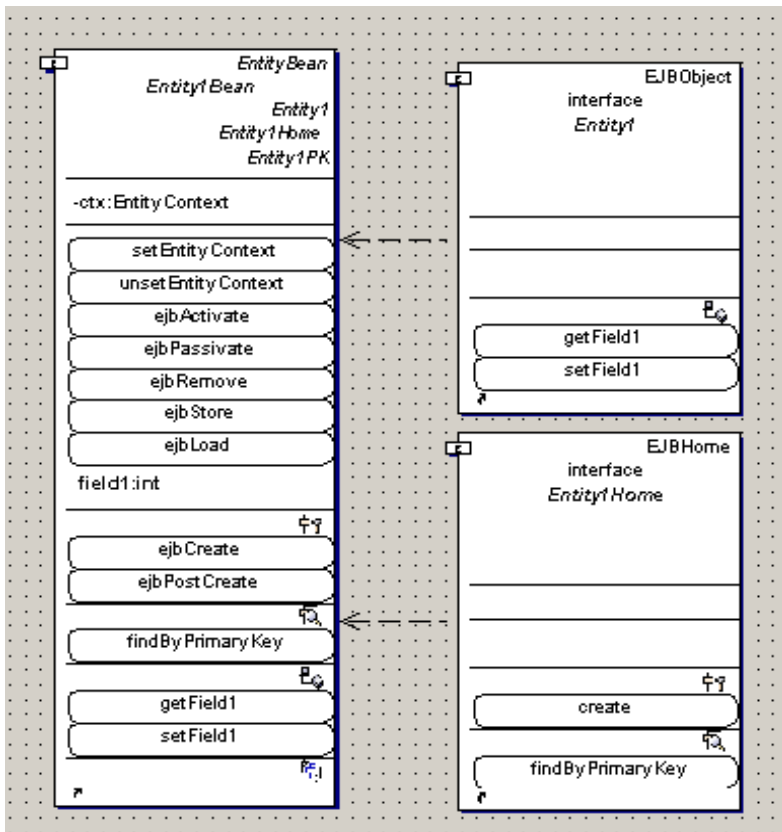
1. Create a new EJBAsembler diagram, or open an existing one in the project that contains the EJBs for the application (**File** | **New** | **Diagram** icon, then select **Together** tab and **EJB Assembler**).
2. To include EJBs that are not part of the current project, specify the paths to them in the Search/Classpath tab of the Project Properties dialog (**Project** | **Properties**).

3. Right-click on the background of the open EJB Assembler diagram and choose **New | Shortcuts**. Alternatively, use the EJB Shortcuts design element from the toolbar.
4. In the *Add Shortcuts* dialog, expand the *Model* node and locate the EJB classes and interfaces from your project that you want to display in the diagram. Select them in the tree view and click the **Add** button.
5. In the *Add Shortcuts* dialog, expand the *Search/Classpath* node and locate the EJB classes and interfaces from outside your project (if any) that you want to display in the diagram. Select them in the tree view and click the **Add** button.
6. Click **OK** to display the selected EJB shortcuts on the diagram.

How EJBs are displayed in diagrams

An EJB element is displayed in essentially the same kind of a visual container as in the Class diagram. The various methods show up as elliptical objects within the Class framework. These objects exhibit highlighting when they are link sources or targets.

Figure 146



Showing and hiding EJB elements

When creating EJB shortcuts in the EJB Assembler diagram, select home or remote interfaces, or primary key classes in the selection dialog. If they do not appear in the diagram, the *View Management* option settings are hiding them. These elements are hidden by default. Change the settings to see them.

To preserve the settings at the project level, change them for the specific diagram.

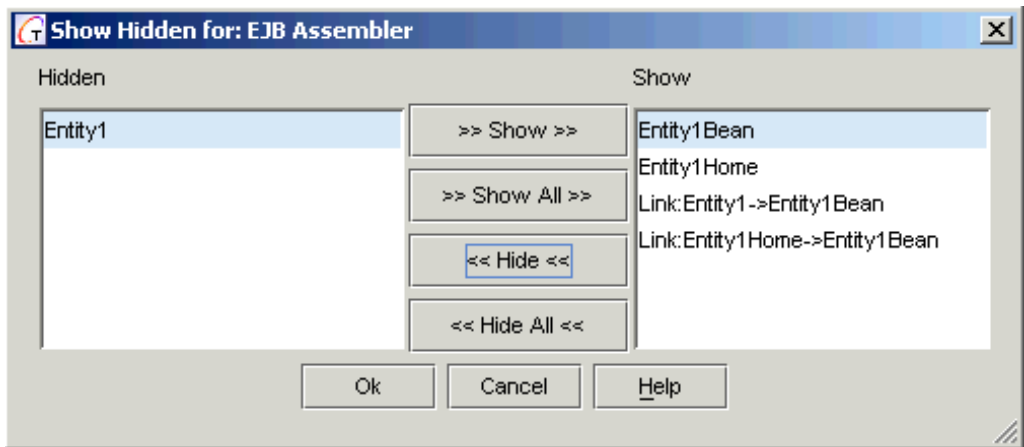
1. Choose **Tools | Options | Diagram Level** on the main menu.
2. Expand *View Management - Show - EJB*.
3. Check the elements you want to display in the current diagram.

Hide individual elements or restore them in the diagram using the Show Hidden command from the right-click menu.

To show or hide interfaces:

1. Right-click the diagram background and choose **Show Hidden**.
2. Select EJBs and interfaces you want to see in the EJB Assembler diagram using the file selector with Add and Remove buttons.

Figure 147



Verification and Correction of EJBs

Verification overview

Generated entity, session, and message-driven EJBs must comply with certain requirements, which include general conformance to all EJB specifications and specific requirements of EJB 1.0, 1.1, and 2.0. Describing the interfaces to be implemented or

methods to be overridden is beyond the scope of this document. Refer to the documentation at www.sun.com.

EJBs are verified according to the selected specification (EJB 1.0, 1.1, or 2.0). Subsequently, it is possible to correct errors that are encountered. To make sure that certain methods or properties exist in an EJB, or to obtain their values, it is necessary to scan the entire hierarchy of objects that comprise the EJB. The members are added to a hash table.

In order to support the OOP approach, there is the possibility to develop classes for customized verification and correction using the Open API.

Together allows users to verify Bean components for various errors:

1. Conformity to the common requirements for all specifications: the correctness of inheritance, home and remote interfaces, testing the primary key type, and so on.
2. Conformity to the specific requirements of the EJB 1.0 Specification.
3. Conformity to the specific requirements of the EJB 1.1 Specification.
4. Conformity to the specific requirements of the EJB 2.0 Specification, including local interfaces and message-driven beans.

Verification runs in compliance with the chosen EJB specification (1.0, 1.1 or 2.0). It is possible to subsequently correct errors that were discovered.

Syntactic rules for verification are defined in the configuration file under `%TOGETHER_HOME%\modules\togethersoft\modules\ejb`, in the folder for the EJB specification that is used. This makes it possible to easily adopt new requirements.

Create customized syntactic rules by editing the appropriate config files for the entity or session beans. However, you are fully responsible for the results of your changes.

Using verification and correction

To make use of this feature, choose the **Verify EJB** command on the bean's right-click menu. The encountered errors are corrected. The process adds the Verification tab to the Messages pane. Two kinds of messages display the results: messages of the first type inform you of encountered errors, and messages of the second type inform you that the errors were fixed.

The verification and correction process can be used during deployment of EJB modules (see Chapter 34, “Deploying J2EE” on page 584).

How an EJB Assembler diagram relates to the J2EE Deployment Expert

Together provides an “expert” dialog that simplifies the process of deploying EJB modules.

Run the J2EE Deployment Expert against either a Class diagram or an EJB

Assembler diagram: main menu | Deploy | J2EE Deployment Expert.

- For “fast track” deployment with default values, or deployment prototyping, use the J2EE Deployment Expert from an appropriate Class diagram.

- For “full featured assembly information” with control over security and permissions, use the J2EE Deployment Expert from an EJB Assembler diagram.
For information on using the J2EE Deployment Expert, see Chapter 34, “Deploying J2EE” on page 584.

Tips for the EJB Assembler diagram

- Work with a container using the special design element *Container Transaction*.
- Separate EJB properties using the special design element *EJB Properties*.
- Use the special design elements for EJB references to facilitate working with them.
- Facilitate working with environment values and resources using special design elements.
For information, see Chapter 28, “Working with Container-Transaction Elements and References” on page 510.
- Create a Web service using an EJB Assembler diagram.
For information, see Chapter 35, “Designing and Developing Web Services” on page 621.
- Provide security support using the special *Principal*, *Security Role* and *Method Permission* design elements.
For information, see Chapter 33, “J2EE Platform Security Support” on page 574.
- Archive several EJB modules and Web applications in one EAR and deploy it to the current application server.
For information, see Chapter 30, “Designing and Developing Enterprise Applications” on page 558.

CHAPTER 28

WORKING WITH CONTAINER-TRANSACTION ELEMENTS AND REFERENCES

This chapter includes the following topics:

- “What is a container-transaction element?” on page 510
- “Creating container-transaction elements” on page 511
- “Deploying container-transactions” on page 513
- “EJB References Support” on page 513

What is a container-transaction element?

The EJB 2.0 specification provides the following definition of a transaction attribute: “...Every client method invocation on an enterprise Bean object is interposed by the container. The interposition allows for delegating the transaction management responsibilities to the container. The declarative transaction management is controlled by a transaction attribute associated with each enterprise Bean’s home container. The container provider’s tools can be used to set and change the values of transaction attributes. Enterprise JavaBeans define the following values for a transaction attribute:

TX_NOT_SUPPORTED

TX_BEAN_MANAGED

TX_REQUIRED

TX_SUPPORTS

TX_REQUIRES_NEW

TX_MANDATORY

An application server uses the container-transaction elements to define the transaction attributes for the methods of EJBs home and component interfaces and for the `onMessage` methods of message-driven EJBs. Each container-transaction element consists of a list of one or more `method` elements and the `trans-attribute` element. The container-transaction element specifies that all the listed methods are assigned the specified trans-action attribute value. All the methods specified in a single container-transaction element must be methods of the same EJB.

The transaction attribute is specified in the enterprise Bean's deployment descriptor.”

See the description of possible transaction attribute variants in the *Sun Microsystems Enterprise JavaBeans 2.0* specification (page 100).

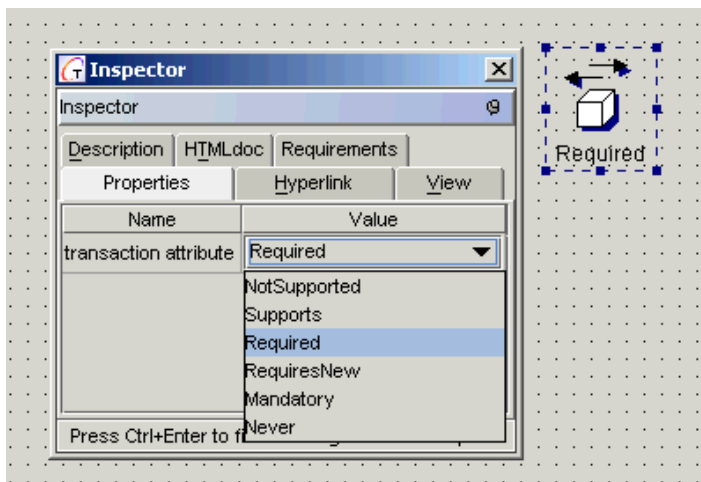
Together makes it possible to define a transaction attribute using the Container Transaction visual component.

Creating container-transaction elements

To create container-transactions:

1. Create a new EJB Assembler diagram or open an existing one.
2. Create a shortcut to the EJB that you want to use (see “Creating EJB shortcuts” on page 504).
3. Select the Container Transaction button on the Class diagram toolbar and click on the diagram pane. This creates a visual component (see Figure 148).

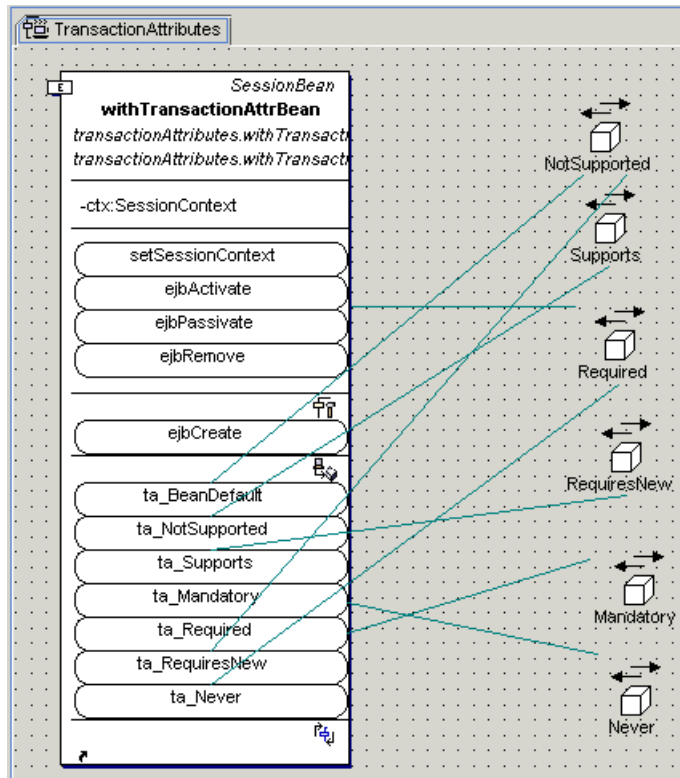
Figure 148



4. *Transaction attribute* is set to *Required* by default. To change this setting, use the Inspector (right-click menu | Properties).
5. Create other container-transaction elements using the Container Transaction design element.

- Now define the appropriate methods for the EJB, and draw the links between these methods and the required Container Transaction elements.

Figure 149



- Define the transaction type (Supported, Not Supported, Mandatory, Never, and so on) in the Inspector for a Container Transaction element (right-click menu | Properties).
- Draw a link from some method in an EJB implementation class to a Container Transaction element. A Container Transaction element can be linked to an EJB icon as a whole, and/or to sub-icons representing methods (see 6.).
- The corresponding transaction type uses default for all methods if a Transaction Attribute links to an EJB icon.
- The corresponding transaction type uses the linked methods, overriding the default value, if a Transaction Attribute links to methods.

Deploying container-transactions

Run the J2EE Deployment Expert against an EJB Assembler diagram. With the diagram open, choose **Deploy | J2EE Deployment Expert** from the main menu.

During deployment, all necessary descriptions connected with container-transaction elements and the corresponding methods are generated and added to the deployment descriptor. Container-transaction elements are added to the JAR of the current EJB module (that is, the EJB Assembler diagram).

EJB References Support

The concept of *EJB references* is very useful for connecting EJBs.

- Regard one EJB as a client with respect to another EJB, meaning that one EJB refers to other EJBs. Place the source EJB and destination EJB in the same JAR file or in different JAR files in the same J2EE module.
- Every EJB reference has a set of its own properties. EJB reference properties can be defined manually, or by visually linking them to the necessary EJB. In the latter case, all EJB properties are transmitted to this EJB reference.
- Two EJBs (one referring to another) cannot be linked directly. For this purpose, a special visual component should be used, called the EJB Reference component. This component refers to the appropriate EJB and is used to access this EJB. In this case, the first EJB gets properties of the second EJB via the intermediate EJB Reference component.
- Using separate visual EJB Reference elements, it is possible to redirect links among EJBs.

Creating and using an EJB Properties element

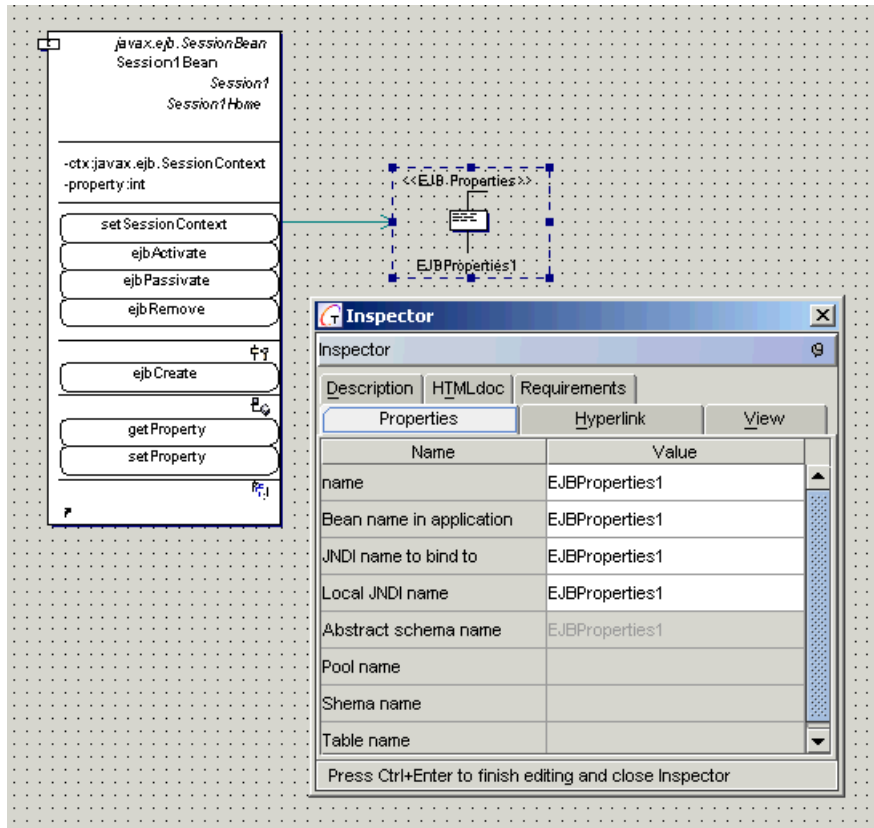
To create an EJB Properties element:

1. Create a new EJB Assembler diagram or open an existing one.
2. Create a shortcut to an EJB on this diagram, or focus on an existing EJB on the diagram.
3. Click on the EJB Properties button on the diagram toolbar and place it on the diagram.
4. Define the name of this element and its other properties.
5. Connect the EJB shortcut to the EJB Properties design element, and use this element for referencing this EJB at a later time.

For example, place a shortcut to this element on another EJB Assembler or Web Application diagram. By doing so, you include the corresponding EJB in the other EJB module or Web application.

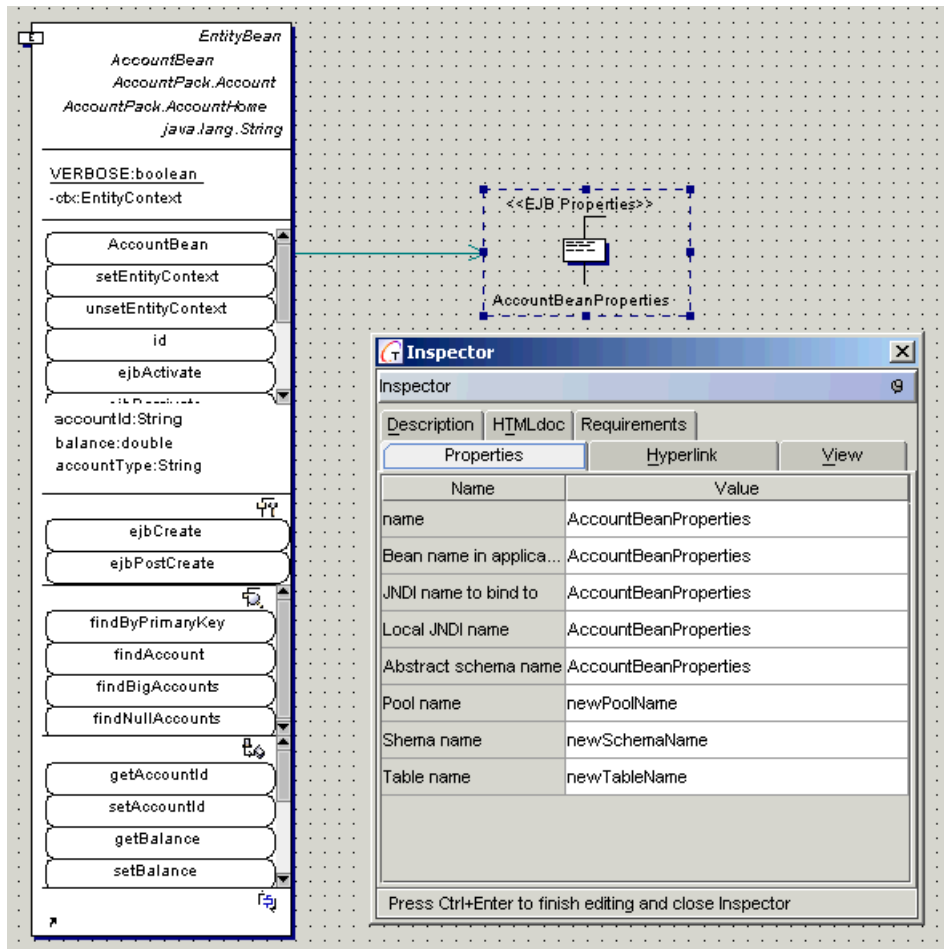
Using the EJB Properties design element, change the EJB name, its name in the application, JNDI name, and local JNDI name. These changes make your application more flexible.

Figure 150



For CMP EJBs, redefine parameters representing the connection with a database. Change the database pool name and the name of the table corresponding to this CMP bean (see Figure 151). This is convenient when using your bean in another application and with another database because it is not necessary to change properties inside the bean. For this purpose, use an additional EJB Properties design element.

Figure 151



Creating EJB references

Together supports the following types of references:

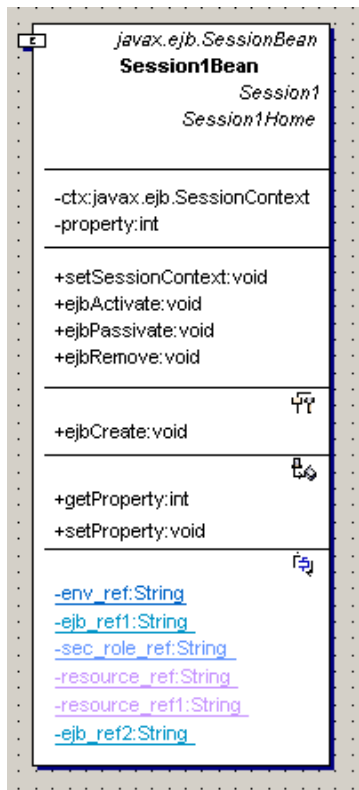
- EJB reference
- EJB local reference
- EJB resource reference
- EJB resource-environment reference
- EJB environment reference
- EJB security reference (do not define for message-driven EJBs)

NOTE: EJB local reference and EJB resource-environment reference are defined for EJBs satisfying the EJB 2.0 specification.

To create EJB references:

1. Create a new EJB Assembler diagram or open an existing one.
2. Create an EJB shortcut on this diagram, or focus on an existing EJB shortcut on the diagram.
3. Right-click on the EJB shortcut and choose **New** from the right-click menu. Choose the necessary reference type from the submenu. The corresponding reference is created inside the current EJB design element in the list of EJB references. References of different types have different colors.

Figure 152



4. All references are `String` type by default. Change the name and the type of a reference as you create it (type the new name and the new type in the corresponding line).
5. Use the reference Inspector to change properties of references.

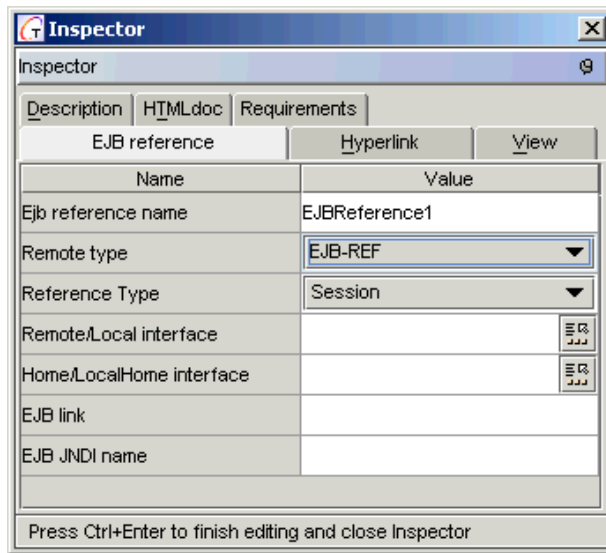
NOTE: To add or remove references to an EJB, alternatively use the References subtab in the EJB Inspector (“References subtab” on page 484).

The EJB References Inspector

To open the References Inspector:

1. Right-click on the Reference design element on the diagram.
2. Choose **Properties** from the right-click menu, and view the Inspector for references.

Figure 153



3. Insert or change the values of these properties.
 4. Choose “EJB-REF” as the remote type if you are using a remote interface.
- Alternatively, use the References subtab on the EJB tab of the bean element Inspector.

Creating EJB local references

The EJB local reference element defines a local reference from one bean to another. Provide this type of reference if local interfaces are used to call from one bean to another. There is no special design element for EJB local references. Use the EJB References design element.

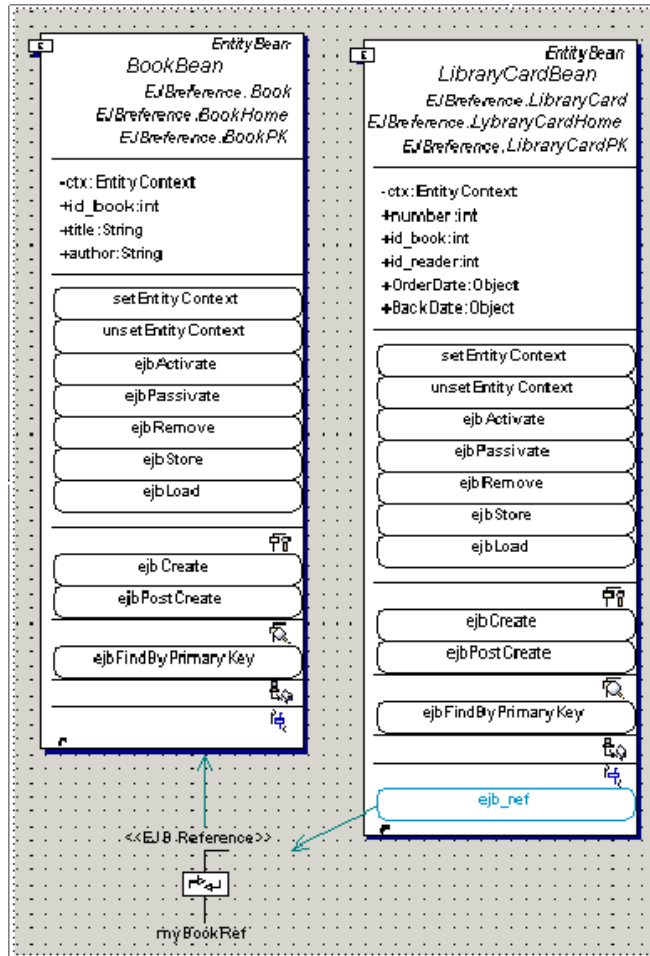
To create an EJB local reference:

1. Place the EJB Reference design element on the current diagram (for instructions, see “Creating EJB references” on page 515).
2. Choose “EJB-LOCAL-REF” as the remote type when using a local interface.

Working with EJB references

- As an example, consider two EJBs: EJB1 and EJB2. EJB Reference from EJB1 to EJB2 is defined in the Class diagram. Then the shortcut to EJB1 is added to an EJB Assembler diagram. We can use an intermediate visual EJB Reference element to redirect the link from EJB1 to another EJB, called EJB3. So in this example, if the J2EE Deployment Expert is started with the Class diagram current, there will be a reference from EJB1 to EJB2. But if the J2EE Deployment Expert is started with the EJB Assembler diagram current, there will be a reference from EJB1 to EJB3.
- It is possible to visually draw EJB references without having EJBs in the project at all.
- Develop a Web application and, instead of importing shortcuts of actual EJB components to the diagram, create an EJB Reference element, specify properties (home, remote interfaces, and others) in the Inspector, and draw a link from the Servlet to the EJB Reference. This information will be stored in the deployment descriptor.
- Using visual design elements to define an EJB Reference with its own properties expands your possibilities for using EJBs. This element is more flexibly used as a connection unit for establishing references between EJBs. This way, you use the same EJB element in different applications without changes, but its properties are modified through the corresponding EJB Reference element.
- EJBs are not allowed to refer directly to each other. One EJB can refer to another EJB only via a visual EJB Reference element. Define an EJB reference (ejb_ref), add the EJB Reference icon (bookBean) to the EJB Assembler diagram, and draw the links.

Figure 154



Sample: How to provide linking between two EJBs

1. Create a new EJB Assembler diagram or open an existing one.
2. Assume that the entity EJBs `BookBean` and `LibraryCardBean` are already created on a Class diagram. Create EJB shortcuts on the current EJB Assembler diagram.
3. To provide `LibraryCardBean` referencing to `BookBean`, right-click on `LibraryCardBean` and choose **New | EJB reference** in the right-click menu. A new `ejb_ref` reference appears inside the EJB shortcut (and also inside `LibraryCardBean` on the Class diagram).
4. Create an EJB Reference element `myBookRef` on the current EJB Assembler diagram.
5. Draw a link from `ejb_ref` (`LibraryCardBean`) to the `myBookRef` design element.

6. Draw a link from myBookRef to BookBean .

Creating EJB resource references

Define a resource reference as an EJB's attribute or as a separate design element (*Resource*). This element has all the properties of the referenced resource (for example, a database object).

To define a resource reference, you should define:

- The resource reference name.
- The resource reference type (*res-type*; in other words, which interface, remote or local, is used to work with this resource).
- The JNDI name for this resource (*res-ref-name* in Together).
- The creation mode of the resource manager and resource authentication (application or container-managed).

Gain access to the corresponding table by creating a reference to a database resource .

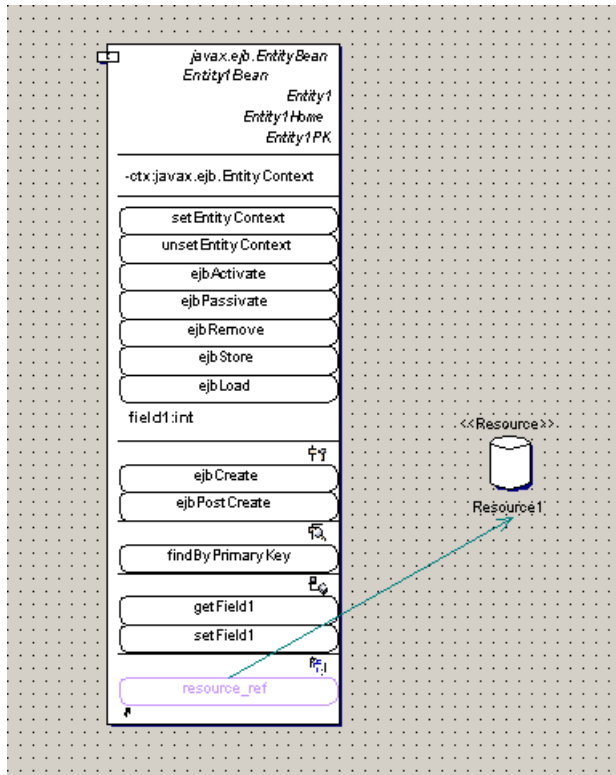
Link a resource reference to an EJB via a resource reference in the EJB implementation class.

Change the resource's properties using the special design element without making any changes to EJB properties.

To create an EJB resource reference:

1. Create a new EJB Assembler diagram or open an existing one.
2. Create an EJB shortcut on the current EJB Assembler diagram (for instructions, see “Creating EJB shortcuts” on page 504).
3. Right-click on the bean and choose **New | EJB Resource reference** from the right-click menu to create an EJB resource reference resource_ref inside the bean.
4. If you are working with a remote interface, choose the remote resource reference type.
5. Create an EJB Resource Reference element Resource1 on the current EJB Assembler diagram.
6. Draw a link from resource_ref to the Resource1 design element .
7. Open the resource Inspector to define or change resource properties.

Figure 155



Creating resource-environment references

To create a resource-environment reference, follow the steps in “Creating EJB resource references” on page 520.

The only difference is in step 4. If you are working with a local interface, choose the `local resource reference` type.

EJB Security Reference Support

Working with security references is one of the mechanisms for J2EE security support.

Creating EJB security references is almost the same as other EJB references. For more information, see Chapter 33, “J2EE Platform Security Support” on page 574.

Creating EJB environment references

Environment References are actually static constants that cannot be changed after an EJB’s deployment. Each environment reference has its type, value, and name. In

Together, define the environment reference as an EJB's attribute or as a separate design element - Environment. This element has its own properties.

An Environment Reference element can be linked with an EJB via an environment reference in the EJB implementation class. Change Environment properties using the special design element without making any changes to the EJB's properties.

To create an EJB environment reference:

1. Create a new EJB Assembler diagram or open an existing one.
2. Create an EJB shortcut on the current EJB Assembler diagram (for instructions, see "Creating EJB shortcuts" on page 504).
3. Right click and choose **New | EJB Environment reference** from the right-click menu to create an EJB environment reference `env_ref` inside the bean.
4. If you are working with a remote interface, choose the `remote resource reference` type.
5. Create an EJB Resource Reference element `Resource1` on the current EJB Assembler diagram.
6. Use the Inspector to define properties of environment references.

For example, assume that we have environment variables of the following types:

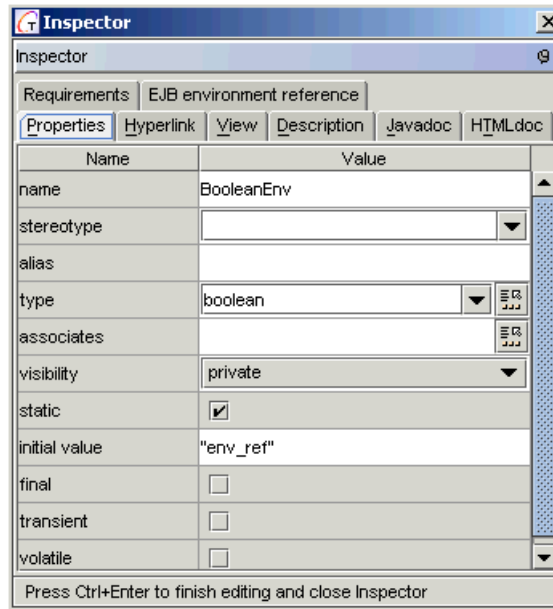
- `boolean BooleanEnv`
- `string StringEnv`
- `integer IntegerEnv`
- `double DoubleEnv`

The properties of the Environment references are defined in the Inspector.

For example, we can define the properties of `BooleanEnv` environment reference as shown in Figure 156.

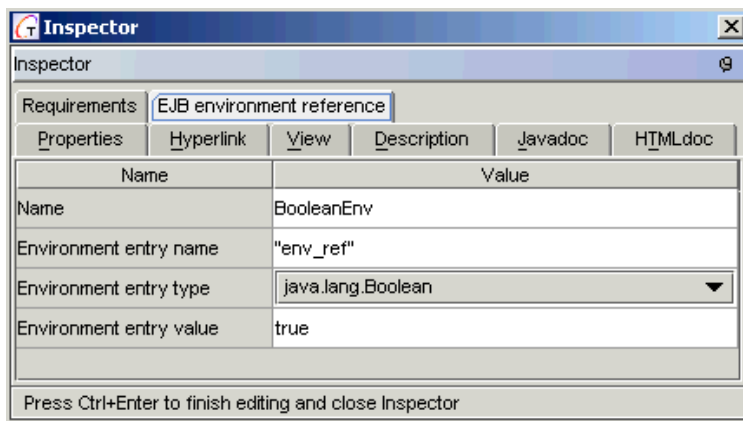
NOTE: The initial value is written in quotes.

Figure 156



On the *EJB Environment Reference* tab of the Inspector, specify the properties of this environment reference.

Figure 157



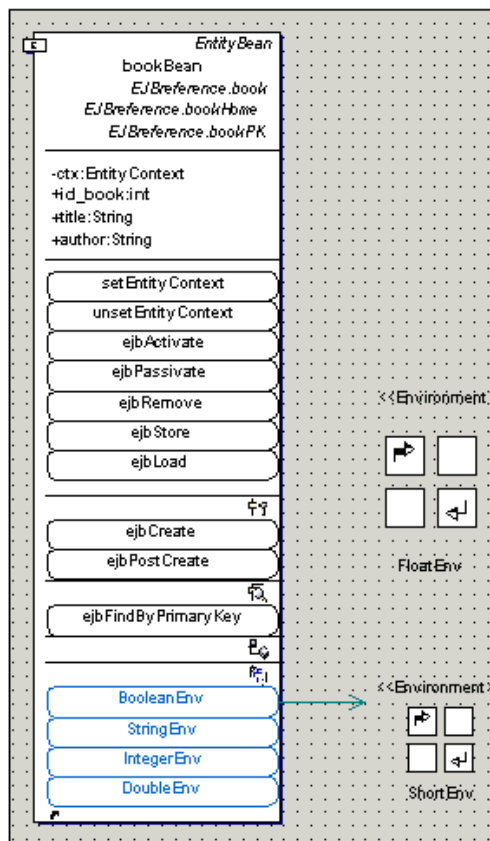
Customize a specific environment entry value using, for example, true for BooleanEnv. When environment references and their properties are completely defined, the corresponding records appear in the EJB's list of attributes.

To change the type of the environment variable:

1. Add a new Environment Reference element `ShortEnv` on the current EJB Assembler diagram.
2. Define its properties as described above. To link `BooleanEnv` with `ShortEnv`, draw an Assembly link from this reference to the `ShortEnv` element.
3. If the properties of `ShortEnv` Environment Reference design element and `BooleanEnv` environment reference are different, and the J2EE Deployment Expert is started with the EJB Assembler diagram current, the properties of `ShortEnv` are preferred.

In our case, the `boolean` type of the environment variable will be redefined as the `short` type.

Figure 158



Look at the two icons on the diagram (see Figure 158): `FloatEnv` and `ShortEnv`. The `FloatEnv` element is not linked to the EJB environment reference and will be deployed as a separate environment variable. Draw a link from the `BooleanEnv` environment reference to the `ShortEnv` visual element.

In this case `BooleanEnv` becomes a `short` type in the `*.xml` file and the name `ShortEnv` is used as a “lookup” name.

- Draw a link from an EJB Reference in an EJB implementation class to an EJB Reference icon.
- Draw a link from an EJB Reference icon to an EJB icon.
- Draw a link from an Environment Reference in an EJB implementation class to an Environment icon.
- Draw a link from a Resource Reference in an EJB implementation class to a Resource Reference icon.

Deploying EJB references

Run the J2EE Deployment Expert against an EJB Assembler diagram or Web Application diagram. Choose **Deploy | J2EE Deployment Expert** on the main menu.

During deploying, all necessary descriptions connected with EJB references are generated and added to the deployment descriptor (`*.xml` file). EJB Reference elements are added to the JAR of the current EJB module (that is, the EJB Assembler diagram).

CHAPTER 29

DESIGNING AND DEVELOPING WEB APPLICATIONS

This chapter includes the following topics:

- “Visual assembly of Web applications for deployment” on page 526
- “Working with a Web Application diagram” on page 535
- “Creating EJB shortcuts” on page 536
- “Creating Container Transactions, EJB Properties, and References” on page 536
- “Developing and designing applets” on page 537
- “Developing and designing servlets” on page 540
- “Developing and designing JSPs” on page 544
- “Designing and developing Tag libraries” on page 548
- “Creating and mapping filters” on page 551
- “Developing and designing listeners” on page 552
- “How a Web Application diagram relates to the J2EE Deployment Expert” on page 556
- “Tips for Web Application diagrams” on page 556

Visual assembly of Web applications for deployment

A Web Application diagram enables you to visually create and build a Web Application archive (WAR), where all JSPs, servlets, and other classes are stored.

If a Web application is already implemented as a set of servlets, JSPs, filters, and other Web files (for example, pictures), it is possible to generate the deployment descriptor and assemble Web components to the archive file with the *.war extension.

On the Together Web Application diagram, you can model the way JSPs, servlets, filters, and other web files are assembled into an application. With the help of a Web Application diagram you can combine all JSPs, servlets, and other Web files in a WAR file and deploy it to an application server.

Creating a new Web Application diagram

To create a new Web Application diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *General* from the categories on the left, and select the *Diagram* icon from the templates on the right. Click **Next** to select the type of diagram.
3. Activate the Together tab, select the *EJB Assembler diagram* icon, and click **Finish**.

Alternatively, click the **New Diagram** button on the diagram toolbar, go to the Together tab, and select Web Application.

You should see the new Web Application diagram with the default name *Web Application*. You can change the default name using the Inspector (right-click menu | **Properties**). Develop the content of the diagram using visual design elements from the toolbar or using the Object Gallery.

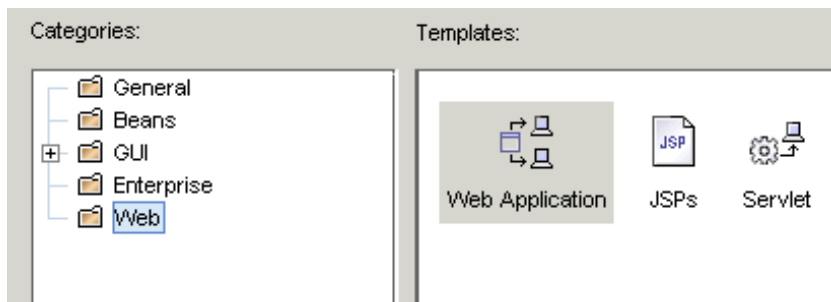
Creating a new Web application diagram using the Object Gallery

You can use the Object Gallery to create a new Web application (Web Application diagram).

To create a Web Application:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Web* from the categories on the left.

Figure 159



3. Select the *Web Application* icon in the templates on the right. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the Web application. The Expert automatically creates the Web Application diagram. To combine several servlets, JSPs, taglibs, Web files in the same module for deployment, go to Web Application in the Object Gallery and follow the recommendations, filling in the most important parameters. This procedure is particularly helpful if you have only limited experience in the field of J2EE technology.

4. Fill in the **New Web Application Module** page in the expert. Set the name of the new Web Application diagram and its location, the name of this diagram in the deployment descriptor, and add the main elements (such as servlets, JavaServer pages, taglibs, or Web files) you want to include in the module. See online help for details.

Figure 160

New Web Application

Web application name: Web Application

Display name: Web Application

Web application location: <default>

Components Included in the Web Application

Servlets JavaServer pages Taglibs Web Files

Add from Project...

Remove

Web application description:

< Previous Next > Finish Cancel Help

5. Click **Finish**. This creates the Web Application diagram that represents your Web application.

Visual design elements

All Web Application visual components are represented in the toolbar on the left side of the diagram.

Table 63 Design elements of a Web Application diagram

Icon	Description
	Zoom: Magnifies the diagram. Access the Zoom tool on the right-click menu (or NumPad+). To zoom out, use the Zoom Out command on the right-click menu (or NumPad-).
	EJB Shortcuts: Opens a selection manager dialog box. Add or remove Web shortcuts (visual representations of servlets, filters, etc.) on the Web Application diagram using the selection manager.
	Security Role: Defines the security role that stands for one of the recommended security roles for the EJB's clients. A Security Role design element in an EJB Assembler diagram presents a simplified view of the EJB application's security to the application deployer. This is used in the J2EE Deployment Expert.
	Principal: Creates a visual design component that represents a User or a Group of Users separated from their <i>Security Role</i> . Note: It is possible to change the stereotype using in-place editing.
	Security constraint: Creates a visual design component for annotating the intended protection of the Web contents. A security constraint includes Web resource collection, authorization constraints, and user data constraints.
	Web resource collection: Creates a visual design component for the set of resources to be protected. A <i>Web Collection</i> is a set of URL patterns or HTTP methods that describe resources to be protected.
	EJB Reference: Creates an element with its own properties. In the visual model, it represents a reference to the corresponding EJB. Define a remote type for EJB reference: EJB, or EJB local (for EJB 2.0 specification).
	Environment Reference: Creates a visual design component with the properties of some static constant, which cannot be changed after EJB deployment.
	Resource Reference: Creates a visual design component that has all properties of the referenced resource. Define the type of reference: resource or resource-environment (for EJB 2.0 specification).
	JSP: Creates a visual design component that represents JSPs and Servlets.

Table 63 Design elements of a Web Application diagram (continued)

	JSP Collection: Creates visual design components that represent a set of JSPs and servlets. Click this button on the toolbar, then use the selection manager for choosing the necessary JSPs.
	Web Files: Creates a visual design component that represents Web files (such as images).
	Filter mapping: Creates a visual design component that enables mapping servlets to filters. Link a servlet to this component and then map it to a filter. Refer to the Servlets 2.3 specification for details.
	Error page: Creates a visual design component for the error page returned if the user authentication fails. Configure an application server to display custom web pages or other HTTP resources in response to particular HTTP errors or Java exceptions. In this case the user-defined pages appear instead of the standard error pages of an application server. If the Error page property is defined, the corresponding error-page element of the deployment descriptor (web.xml) appears. This optional element maps error code or exception type with the resource path in the Web application.
	TagLib: Creates a visual component used to describe the JSP tag library. This tag library should be previously created.
	Web Link: Drawn between diagram elements to show their relationships in the application.
	Note: Creates a visual Note element.
	Note Link: Creates a link between the note element and another visual component to show the note's relationship with this element.

Key elements and properties

Similar to the EJB Assembler diagram, some of the key elements cannot be added to the diagram using the toolbar buttons. To show EJBs in the Web Application diagram, create a shortcut to each EJB. To include Home and Remote interfaces of an EJB, add the corresponding shortcuts to this EJB in the Web Application diagram.

Notation

The Web Application diagram is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

Once you create this diagram, open the J2EE Deployment Expert to generate the deployment descriptor and deploy the application to the selected server. If you have a separate element (for example, some reference in a Web Application diagram), the appropriate record appears in the generated deployment descriptor. Home and remote interfaces are generated for EJBs.

Properties of a Web Application diagram

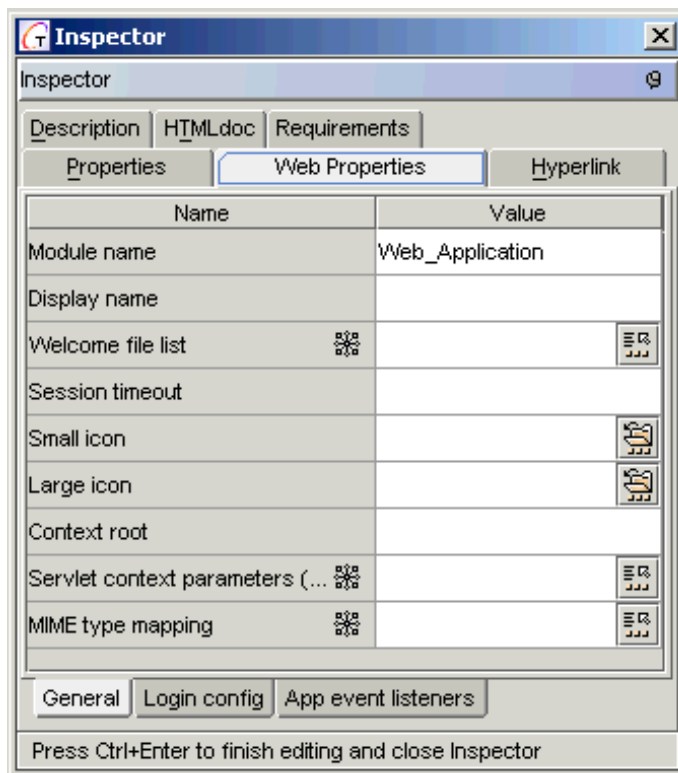
A Web Application diagram has certain Web properties that are available on the *Web Properties* tab of the diagram's Object Inspector. There are three subtabs listed at the bottom of the Object Inspector page:

- *General*
- *Login config*
- *App event listeners*

General subtab

Figure 161 contains general Web application properties, used mainly in a deployment descriptor.

Figure 161



NOTE: Specifying the properties, keep in mind that the application server should comply with the Sun Microsystems specification for EJB 1.1 and higher. Refer to <http://java.sun.com/products/ejb/docs.html> for details.

Module name

A name of an archive *.war file that is created during the deployment process.

Display name

An optional parameter for the deployment descriptor.

Welcome File List

This property enables you to define welcome pages in the deployment descriptor of a Web Application (web.xml). The Welcome File List is an optional element that contains an ordered list of welcome file elements. When the URL refers to a directory name, the application server goes to the first file in this list. If this file is not found, the application server tries the next one. In other words, the Welcome File List is a list of files that can be used as a starting page. This process enables users to type a URL without having to type a specific file name.

Welcome pages are defined on the Web Application level. If your server hosts multiple Web applications, define welcome pages separately for each Web application. If the welcome pages are not defined, the application server looks for the following files in the following order:

1. index.html
2. index.htm
3. index.jsp

Enter the lists of welcome pages in the appropriate field of the Inspector, using a comma as the delimiter.

Additional options

- *Display name* - an optional parameter for the deployment descriptor.
- *Small icon* - an optional field with the file chooser for selecting a small icon file. Use this optional parameter to select the small icon appearance for the BEA WebLogic Icon Element. In this case you must specify the location of a small (16x16 pixels) *.gif or *.jpg image used to represent the Web application in the GUI tool.
- *Large icon* - an optional field with a file chooser for selecting a large icon file. Use this optional parameter to select the large icon appearance for the BEA WebLogic Icon Element. In this case you have to specify the location of a large (32x32 pixels) *.gif or *.jpg image used to represent the Web Application in the GUI tool.

Session time-out

This is an optional element defining one of the session parameters for a Web application: the time in seconds that an application server waits before timing out a session. Minimum value is 1, default is 3600, and maximum value is integer MAX_VALUE. This parameter enables you to override the session time-out element in web.xml.

Context Root

An optional parameter of the deployment descriptor that defines context root for a Web application. The path name must begin with a slash (/) and is interpreted as relative to the current context root.

Servlet Context Parameters

Use this optional parameter to define a context-param element that declares context initialization parameters of the Web application servlet. The following methods are used to access these parameters:

```
javax.servlet.ServletContext.getInitParameter()  
javax.servlet.ServletContext.getInitParameterNames()
```

Mime type mapping

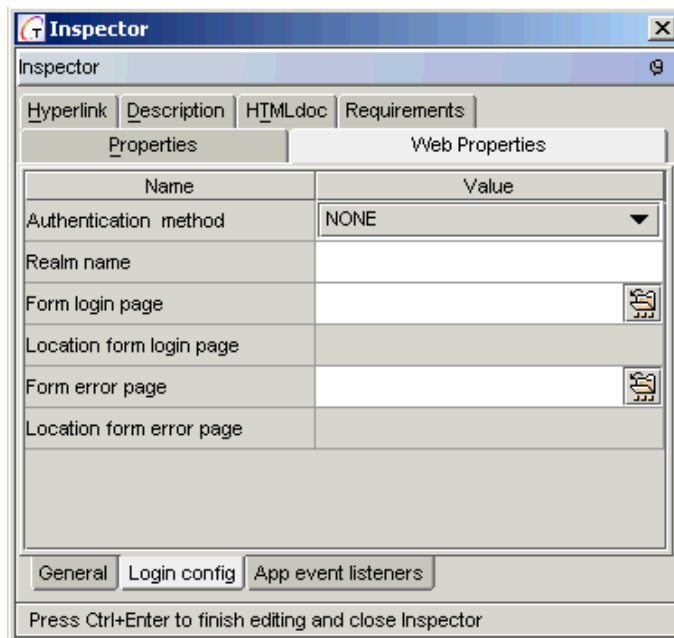
This parameter, which defines a mapping between an extension and a mime type, is an element of the deployment descriptor. Fill in this field using an extended editor.

Login Config subtab

This page contains additional Web properties used for security purposes. For instructions on how to provide security, see Chapter 33, “J2EE Platform Security Support”.

This section explains the purpose of the parameters represented on this page.

Figure 162



Authentication method

A deployment parameter defining an authentication method that should be used.

Realm name

A deployment parameter that specifies the realm name to use in HTTP.

Form login page

A deployment parameter that defines the form-login-page element of the deployment descriptor: the full path and the name of a JSP file that will be used as a login page. If the Location form login page is filled in, this parameter is disabled.

Location form login page

A deployment parameter that defines the form-login-page element of the deployment descriptor: the location in the Web application of the login page. The path defined in the Location form login page begins with a leading slash (/) and is interpreted relative to this root. If the Form login page is chosen, this field is disabled.

Form error page

A deployment parameter that defines the form-login-page element of the deployment descriptor: the full path and the name of a JSP file that will be used as a login page. If the Location form login page is filled in, this parameter is disabled.

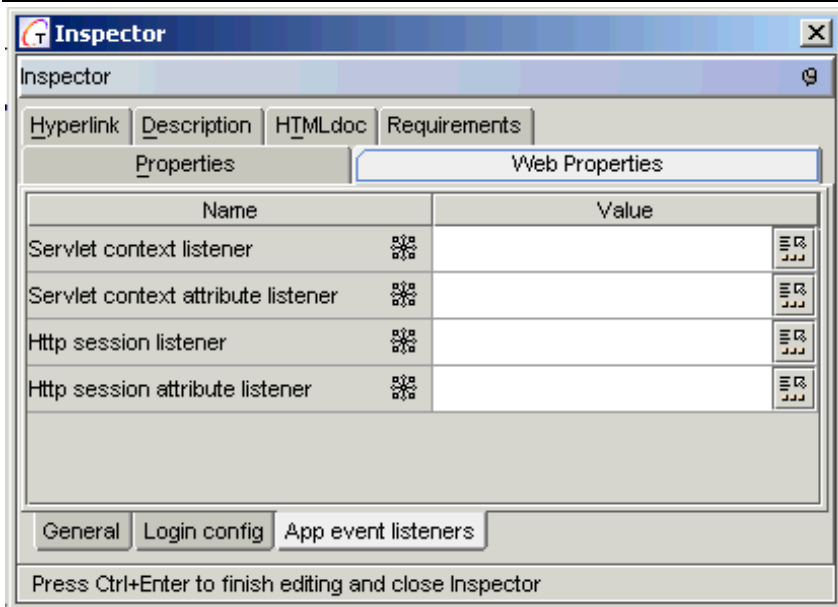
Location form error page

A deployment parameter that defines the form-login-page element of the deployment descriptor: the location in the Web application of the login page. The path defined in the Location form login page begins with a leading slash (/) and is interpreted relative to this root. If the Form login page is chosen, this field is disabled.

App event listeners subtab

Develop listeners for servlets.

Figure 163



Each line on this page corresponds to one of the listeners' types. These files are optional. To use one or more listeners of this type, insert the corresponding full paths, separated by a semicolon. Use the extended editor for choosing the files.

Working with a Web Application diagram

In addition to the basic technique of drawing diagrams, there are several fundamental functions of the Web Application diagrams:

- “Creating EJB shortcuts” on page 536
- “Creating Container Transactions, EJB Properties, and References” on page 536
- “Developing and designing applets” on page 537
- “Developing and designing servlets” on page 540
- “Developing and designing servlets” on page 540
- “Developing and designing JSPs” on page 544
- “Creating and mapping filters” on page 551
- “Developing and designing listeners” on page 552

For information on how to work with EJB properties, containers, and references, see Chapter 28, “Working with Container-Transaction Elements and References” on page 510.

You can read about security support of Web applications in Chapter 33, “J2EE Platform Security Support” on page 574.

When a Web Application diagram is completed, you can invoke the J2EE Expert for deploying all elements placed on it to a single WAR archive file. For more details, see Chapter 34, “Deploying J2EE” on page 584.

Creating EJB shortcuts

EJB shortcuts placed on a Web Application diagram (see “Creating EJB shortcuts” on page 504), can be viewed on the diagram. However, during deployment EJBs are included in the WAR file like ordinary classes. Use EJB modules (EJB Assembler diagrams) for deployment of EJBs.

If servlets or other elements of your Web application use EJBs (elements from an EJB Assembler diagram), it is necessary to place shortcuts of the corresponding EJB interfaces on the Web Application diagram.

To create a shortcut of an EJB interface:

1. Open an existing Web Application diagram.
2. Choose **New | Shortcut** on the right-click menu. Alternatively, you can use the Shortcut button on the toolbar.
3. In the file chooser, use the Add and Remove buttons to select the necessary interfaces, and click **OK** to close the extended editor.
4. You can view the new shortcuts on the Web Application diagram.

Creating Container Transactions, EJB Properties, and References

If servlets or other elements of your Web application use EJB properties, container transaction elements, or references, it is necessary to place the corresponding design elements (or their shortcuts) on the Web Application diagram.

To create a shortcut of an existing EJB property, container transaction element, or reference:

1. Open an existing Web Application diagram.
2. Choose **New | Shortcut** on the right-click menu. Alternatively, you can use the Shortcut button on the toolbar.
3. Using the file chooser, Add and Remove the necessary EJB properties, container transaction elements, or references, and click **OK** to close the extended editor.
4. See new shortcuts on the Web Application diagram.

To create EJB properties, container transaction elements, or references:

1. Open an existing Web Application diagram.
2. Choose **New** on the right-click menu and choose the name of the added design element.

3. View the new shortcut on the Web Application diagram. Change the properties of this new design element using the Inspector.

For more information on how to work with EJB properties, container transaction elements, or references, see Chapter 28, “Working with Container-Transaction Elements and References” on page 510.

Developing and designing applets

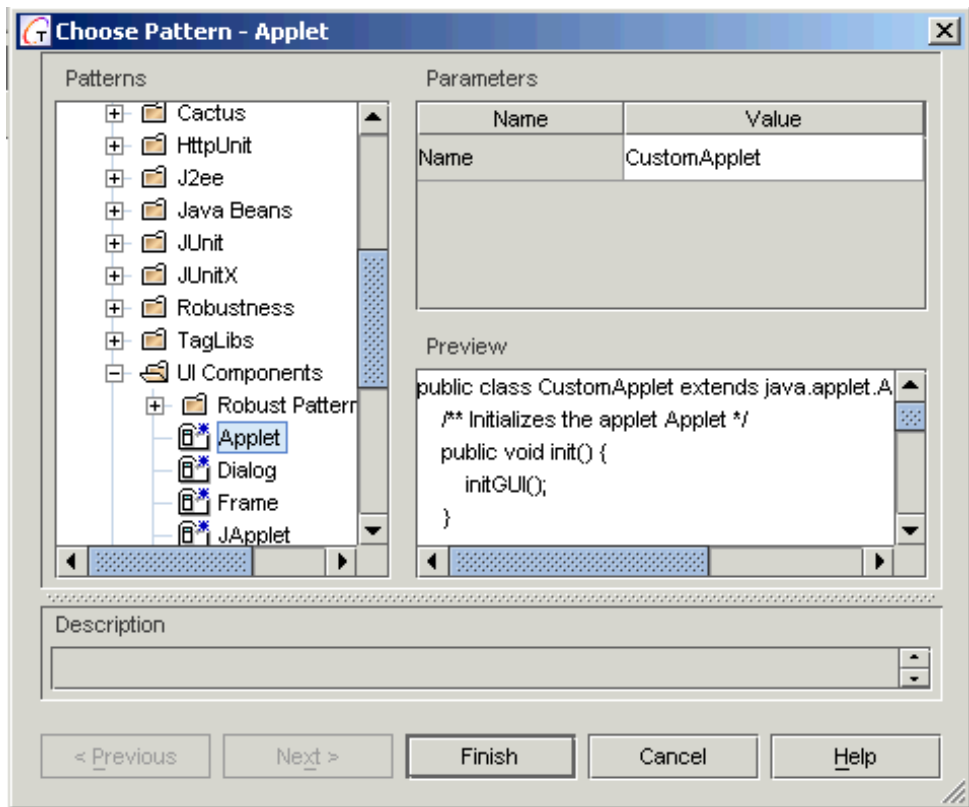
Creating an applet

An applet is not a part of J2EE technology, but shortcuts for these elements can also be used in Web Application diagrams.

To create an applet:

1. Create a project for a listener. Choose **New Project Expert** on the File menu, and insert a project name and specify the required path for your project (for example, c:\jdk1.3.1\demo\applets\GraphicsTest). Alternatively, you can open an existing project.
2. When ready, click **Finish** to complete and close the New Project Expert dialog.
3. Open a Class diagram. If there are *.java files in the directory (for example, c:\jdk1.3.1\demo\applets\GraphicsTest), you can see the corresponding applets, other elements, and links on the current Class diagram.
4. To create a new applet, choose the **Class by Pattern** command from the diagram toolbar or right-click menu.

Figure 164



5. In the Choose Pattern dialog, select the Applet pattern (*UI Components - Applet*). It contains the default class name and the standard set of methods: `init()`, `start()`, `stop()`, `destroy()`, `getAppletInfo()`, `initGUI()`.
6. Enter an applet class name and click **Finish**.
7. Edit the source code in the editor to add the required functionality. Add the code to this skeleton in place of the green comments “// Write your code here.”
8. Correct the properties of your applet using the Inspector (choose **Properties** on the applet’s right-click menu).

Having created an applet, run and debug it.

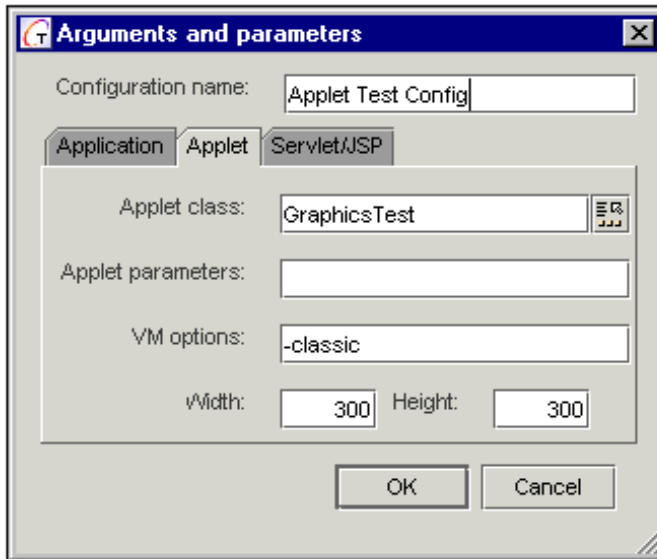
Running and debugging an applet

To debug an applet:

1. Make sure that `%TG_HOME%/lib/openapi.jar` library is added in the Search / Classpath page of the Project properties.

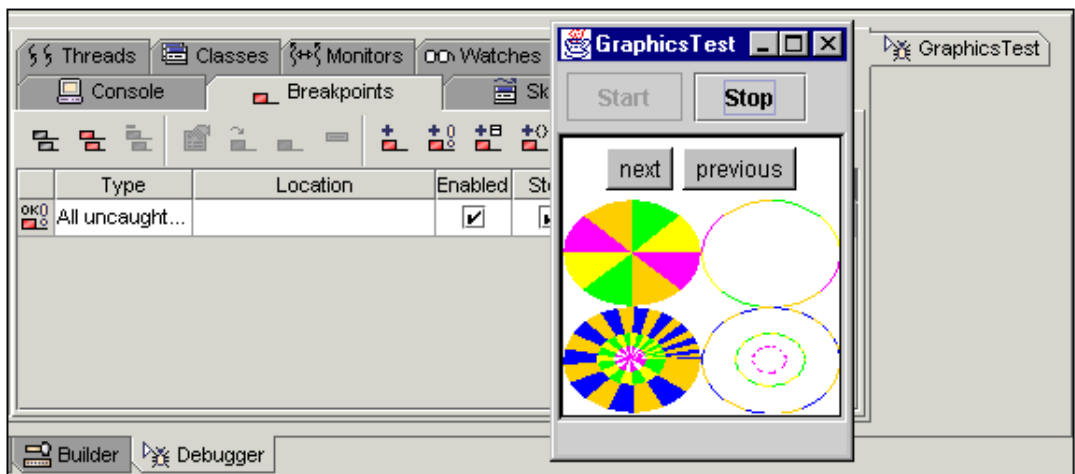
2. Choose **Run** (or **Debug**) on the Run menu. The Arguments and Parameters dialog contains the Applet tab. Specify the name of the main class, parameters passed to the applet, and VM options (if any), and the dimensions of the applet frame. Parameters are entered in the format "name1"="value1" "name2"="value2" and so on.

Figure 165



3. Click **OK** when ready. The Debugger (or Runner) pane opens with the applet frame:

Figure 166



Deploying an applet

To deploy an applet to an application server:

1. Create a Web Application diagram or open an existing one.
2. Click the **Add Shortcut** button on the toolbar (or choose **New | Shortcuts** on the right-click menu) to open the file chooser dialog.
3. Place the shortcut of the necessary applet on the Web Application diagram.
4. Add other classes if necessary.
5. Invoke the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).

Developing and designing servlets

Servlets are used to implement client-server interactions. Normally, this is a http-based interaction between an Internet browser and a Web server. Any applications capable of creating http-requests may also act as a client. A Java applet or Java application are examples.

A servlet is a Java class on the server side that implements a standard interface for handling http requests. It should be noted that there may be multiple classes on the server side to handle http requests. However, the servlet only gets direct requests from the client.

The number of servlets allocated on the server is not determined. A single servlet can cater for one or for multiple types of requests. It is possible to create different configurations: allocate one servlet on the server to handle all incoming requests, or allocate numerous servlets for each request type. The solution is determined by the specific task and the developer's preferences.

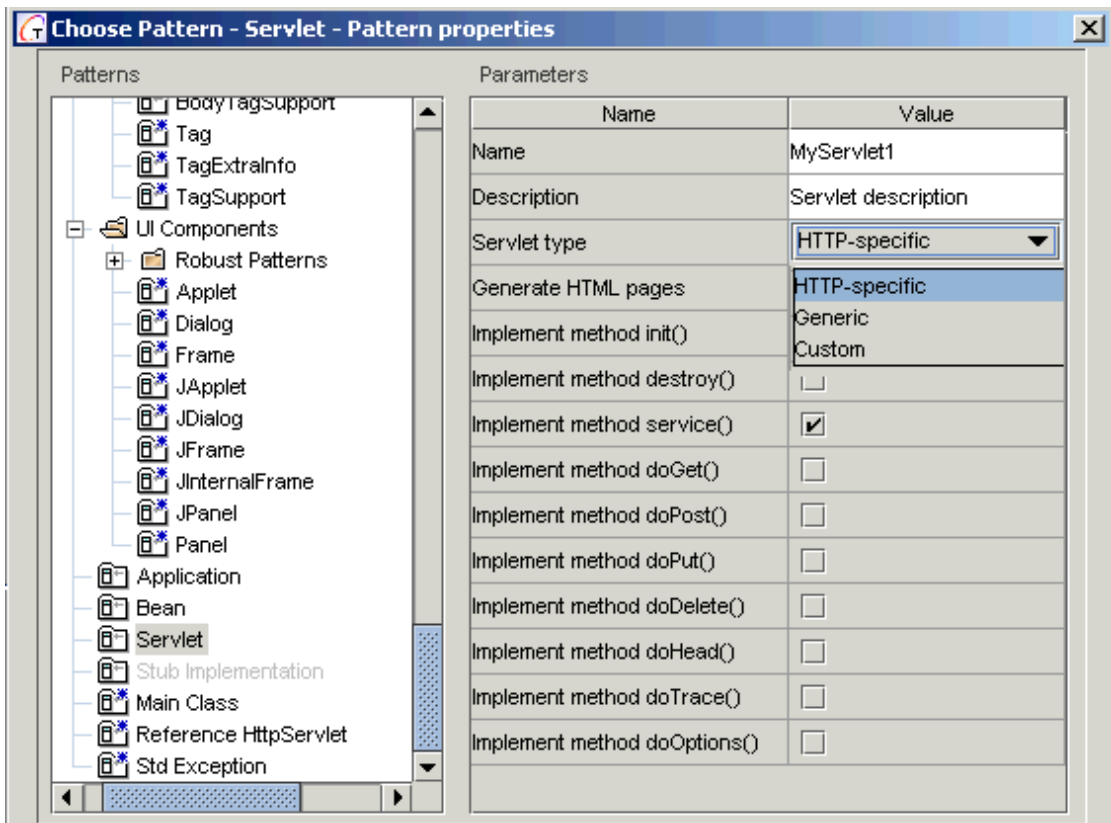
Creating a servlet

Together provides a convenient way to create servlets using patterns.

To create a servlet:

1. Create a project for a listener. Choose **New Project Expert** on the File menu, and insert a project name and specify the required path for your project. Alternatively, you can open an existing project.
2. When ready, click **Finish** to complete and close the New Project Expert dialog.
3. Open a Class diagram. Choose the **Class by Pattern** command from the diagram toolbar or right-click menu. This opens the Choose Pattern dialog.

Figure 167



4. Define the servlet type. There are three servlet patterns available in Together: the http-specific servlet pattern that generates a skeleton of a basic servlet that extends `HttpServlet`; the generic servlet pattern that extends `GenericServlet` class and services any request-response protocol; and the Custom Servlet that directly implements the Servlet interface. This enables the user to create servlets in accordance with the specific task.
5. Define other parameters in the Choose Pattern dialog. Select the methods to be generated in the body of the class. This step alleviates tedious coding. You can also choose to generate the bodies of methods. Checking *Generate HTML pages* adds code to the request-processing methods (`service`, `doGet`, `doPost`, `doPut`, `doDelete`), that return an empty HTML page.
6. If a necessary method is not selected when creating a servlet by pattern, add it using the Inspector (Choose **Properties** on the servlet's right-click menu). You can also add and correct the code.

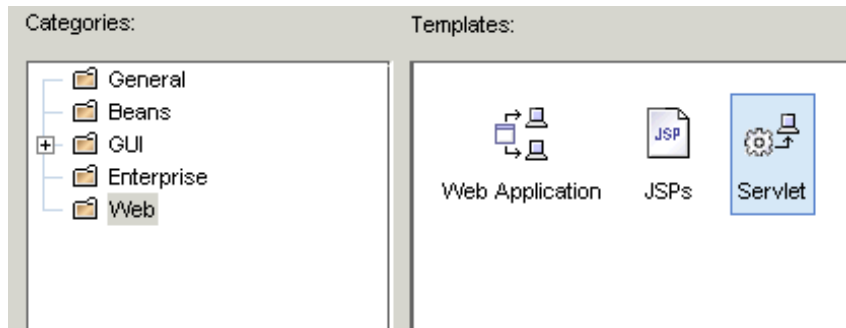
Creating a new servlet using the Object Gallery

Use the Object Gallery to create a new servlet.

To create a servlet:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Web* in the categories on the left.

Figure 168



3. Select the *Servlet* icon in the templates on the right. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the servlet. The Servlet Expert automatically creates the Web Application diagram. This procedure is particularly helpful if you have only limited experience in the field of servlets' developing.

4. Fill in the **New Servlet** page in the expert.

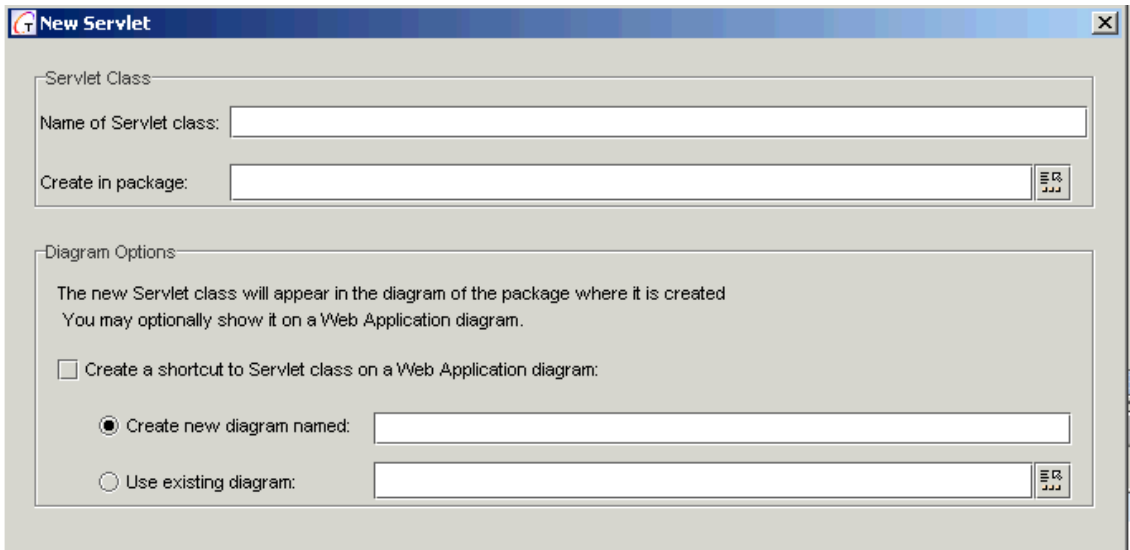
Servlet Class pane. Set the name of the new servlet and its location (the full path).

Diagram options pane. Include the servlet in a Web application. There are two ways to do this:

- create the servlet on the new Web Application diagram that is created automatically,
- create the servlet on the existing Web Application diagram.

If you choose the first variant, define the name of this diagram. Otherwise, type the name of the existing diagram in the corresponding field, or use the file chooser for this.

Figure 169



5. Click **Finish**. This creates the new Web Application diagram (or opens the existing Web Application diagram), and creates your servlet on it.

Running and debugging a servlet

Jakarta Tomcat, the Reference Implementation for the Java Servlet 2.2 (or higher) Technology, is used to debug servlets. This is how it's done:

1. Make sure that %TG_HOME%\bundled\tomcat\lib\servlet.jar is added to the Search/Classpath of your project.
2. Select a servlet for debugging on the Class diagram and set breakpoints as required.
3. Choose **Debug** on the Run menu. Select the Servlet/JSP tab and enter the name of the servlet, or choose one using the browse button.
4. Click **OK** and observe the results in the Debugger pane and in the browser window.

Deploying a servlet

To deploy a servlet to an application server:

1. Create a Web Application diagram or open an existing one.
2. Click the **Add Shortcut** button on the toolbar (or choose **New | Shortcuts** on the right-click menu) to open the file chooser dialog.
3. Place the shortcut of the necessary servlet on the Web Application diagram.
4. Add other classes if necessary.

5. Invoke the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).

Developing and designing JSPs

JavaServer Pages (JSPs) represent HTML pages with JavaScript. JSPs improve the enterprise deployment with a Web-based presentation layer. At runtime, JSPs compile into servlets. Both servlets and JSPs are network components suited for request/response interaction with a client over HTTP.

The main difference between a servlet and a JSP file is that a JSP is more oriented to look-and-feel issues, and JSP code is not pure Java code, as in the case of a servlet.

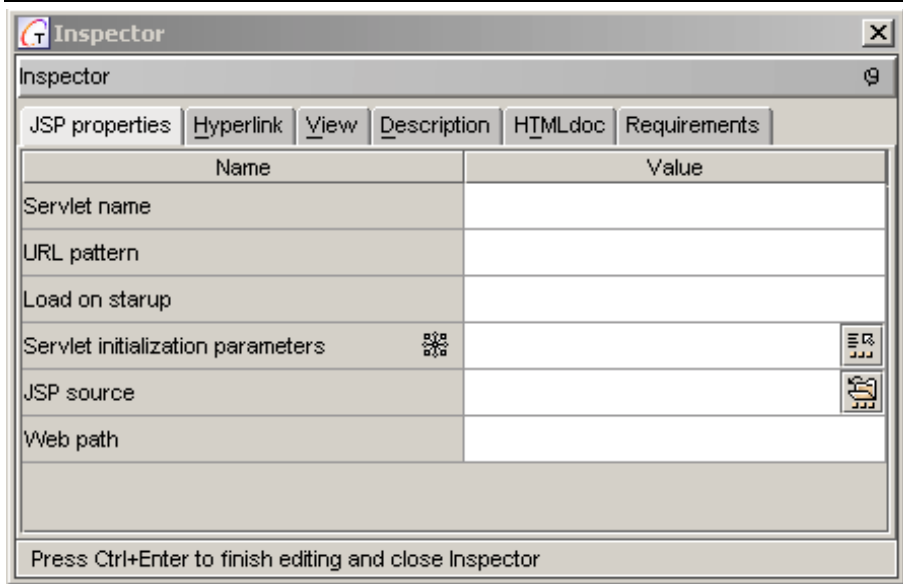
Creating a JSP

A new JSP can be created on a Web Application diagram.

To create a JSP:

1. Create a new Web Application diagram or open an existing one.
2. To create a single JavaServer Page on the Web Application diagram, use the toolbar button or choose **New | JSP** on the diagram’s right-click menu.
3. Having created a visual component, observe its properties in the Inspector (right-click on the JSP and choose **Properties**).

Figure 170



4. Insert a JSP name and fill in the editing fields.

5. Click the file chooser button in the *JSP Source* field, and select the desired *.jsp file that will be loaded into the created visual component.
6. Having created a JSP, edit it as required, add necessary code, run and debug.

Creating a JSP collection

To load several JSPs in the Web Application, create a collection of JSPs.

To create a collection of JSPs:

1. Create a new Web Application diagram or open an existing one.
2. To create a collection of JavaServer Pages on the Web Application diagram, use the toolbar button, or the **New | JSP Collection** command on the diagram's right-click menu.
3. Select the JSPs files needed using the file chooser. Click **OK** when ready. All selected JSPs appear on the Web Application diagram.
4. Having created visual components, observe the properties of each component in the Inspector (choose **Properties** on the JSP's right-click menu).

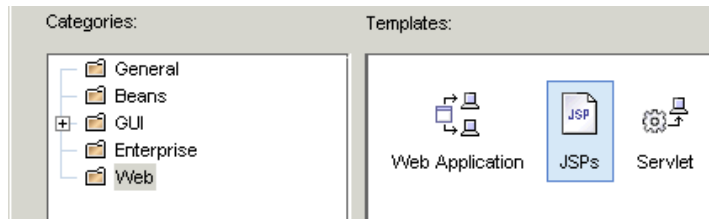
Creating new JSPs using the Object Gallery

Also use the Object Gallery to create new JSP(s).

To create JSP(s):

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Web* in the categories on the left.

Figure 171



3. Select the *JSPs* icon in the templates on the right. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the servlet. You do not have to understand how to use the Web Application diagram, because it can be created automatically. This procedure is particularly helpful if you have only limited experience in the field of JSPs' developing.

4. Fill in the **JavaServer pages** page in the expert.

Creation scenario pane. You can choose one way of doing this:

- importing the existing JSP(s) using the file chooser;

- creating the new JSP.

If you choose the second variant, define the name of this JSP and its location.

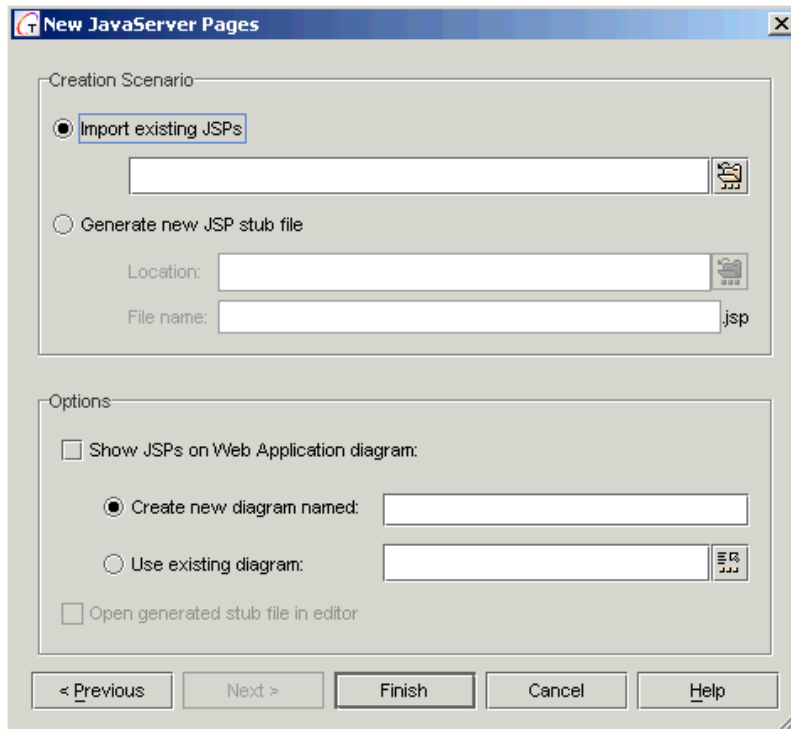
Options pane. Your JSP(s) should be included in some Web application. You can choose one way of doing this:

- creating the JSP(s) on the new Web Application diagram that is created automatically,

- creating the JSP(s) on the existing Web Application diagram.

If you choose the first variant, define the name of this diagram. Otherwise, type the name of the existing diagram in the corresponding field, or use the file chooser for this.

Figure 172



5. Click **Finish**. This creates the new Web Application diagram (or opens the existing Web Application diagram), and creates your JSP(s) on it.

Debugging JSPs

Together provides various ways to debug JSPs. The most basic way to run a JSP in debug mode requires appropriate settings in the Run Configuration dialog.

In theory, after the settings are made, it is possible to run or debug a JSP. However, in most cases, JSPs are intended for data presentation only; consequently, they need to be invoked by a servlet or an HTML page. This type of debugging is mainly suitable for servlets, rather than JSPs.

Together provides JSP debugging using Jakarta Tomcat 3.2, the official Reference Implementation for the Java Servlet 2.2 and JavaServer Pages 1.1 Technologies.

One type of JSP debugging takes place when an EJB is deployed to a certain application server. The client can be created as a JSP that allows to invoke EJB methods. For IBM WebSphere 3.5, and BEA WebLogic 5.1, 6.0, and 6.1, it is possible to create a JSP client in the debug mode. If this option is selected, the page *Simple JSP client generation* provides the fields for starting the debugger session under Tomcat.

The most comprehensive method of debugging JSPs suggests integration with Web Application diagram deployment and use of the generated files and configurations as a background for launching the debugger session under Tomcat.

Debugging JSPs under Tomcat

To debug JSPs under Tomcat:

1. On the current Web Application diagram, which contains JSP visual components, select the desired JSP.
2. Open the selected JSP in the Editor pane.
3. Set breakpoints in the desired lines.
4. Open the J2EE Deployment Expert.
5. Choose Generic 1.1 server and select the following tasks on the first page of the Expert:
 - Compile classes from the selected diagram
 - Generate WAR Deployment Descriptor
 - Run Web Application under Tomcat in debug mode
6. After entering all necessary fields, on the page *Run Web Application under Tomcat in debug mode* specify the root folder of the Web application.

NOTE: The libraries of the appropriate application server where the client will search for the `INITIAL_CONTEXT_FACTORY` class should be added to the project classpath if a JSP can address to an EJB. Additionally, the JSP file should contain lines for the proper handling of the `InitialContext` object.

It is vitally important to properly specify the root catalogue. The *.jsp file being debugged should be the actual root element. Otherwise, the debugger session will start but with an incorrect component.

Deploying JSPs

To deploy one or more JSPs to an application server:

1. Create a Web Application diagram, or open an existing one.
2. Create JSPs on this diagram.

3. Add other classes if necessary.
4. Open the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).

Designing and developing Tag libraries

This section provides the following information:

- Overview of the tag library concept
- Elements and properties of the Taglib diagram
- How to create tag libraries using the Taglib diagram
- How to make use of the custom tags in the Web applications.

Taglibs

JSPs enables programmers to implement dynamic content generation by including Java source code directly in the web pages and includes a set of HTML-like tags for interacting with Java objects residing on the server.

Tag libraries are composed of a set of custom tags. These custom tags help separate presentation from implementation. Web designers can change the layout without having to modify the underlying logic. Custom tags also help developers avoid embedding scripting code within the JSP page as well as encourage reuse and ease maintainability.

The tag handlers must implement either of the two interfaces **Tag** or **BodyTag**, or extend one of the classes, **TagSupport** or **BodyTagSupport**. There is an additional handler that extends **TagExtraInfo** class. To learn more about the tags and tag handlers and their implemented methods, refer <http://www.javasoft.com/products/jsp/tutorial/TagLibraries.pdf>.

Create custom tag handlers in the Class diagram using Class by Pattern icon and in Taglib diagram.

Taglib diagram

Main elements

TagLib diagram allows the easy creation of handlers and declares the tags in a tag library descriptor.

TagLib diagram contains the following design elements:

Table 64 Main elements of the Taglib diagram

Icon	Description
	Creates a handler for a simple tag without a body. The handler class implements Tag interface.

Table 64 Main elements of the Taglib diagram (continued)

Icon	Description
	Creates a handler for tag with attributes. The handler extends TagSupport class.
	Creates a handler for a tag with a body that contains tags, scripting elements, HTML text etc. between start and end tags. The handler class implements BodyTag interface.
	Creates a handler for a body tag with attributes. The handler class extends BodyTagSupport.
	Creates a visual design element for a class that extends TagExtraInfo and provides information to the JSP container about the scripting variable.
	Notes and note links

Properties of the diagram

Inspector of a TagLib diagram contains specific page TagLib Properties and defines the tag library.

- **TLD File Name:** This field allows users to specify the fully qualified name of the tag library descriptor or select it using the File Chooser button. The descriptor generated by Generate TLD command of the diagram right-click menu resides in this location.
- **Short name:** A simple default name used by a JSP page authoring tool to create names with a mnemonic value. For example, a short name may be used as the preferred prefix value in taglib directives and/or to create prefixes for IDs.
- **TagLib Version:** Version of the tag library.
- **JSP Version:** The JSP specification version used by the tag library.
- **TagLib URI:** The URI that uniquely identifies the tag library.

Properties of the elements

- **Tag Name:** Enter tag name.
- **Body Content:** Select the required content from the drop-down list.
- **TEI Class:** This field allows users to specify the class of additional information, associated with the tag handler.

Creating Tag libraries

Together provides a convenient method of creating tag libraries using the Taglib diagram. The basic procedure is:

1. Create the TagLib diagram
2. Generate the Tag Library Descriptor
3. In the Web Application diagram, add the required tag handlers.

These steps are described below.

To create a TagLib diagram:

1. Create the TagLib diagram, applying usual technique.
2. Populate the diagram with the necessary elements.
3. For each element invoke its Properties Inspector:
4. Fill in the inspector fields: tag name, implemented TagExtraInfo class, and tag body content (if any).
5. Enter additional information in the *Description* tab of the object inspector.

After creating the diagram, proceed with generating the tag library descriptor.

To generate the Tag Library Descriptor

1. Right-click on the diagram pane to invoke the diagram Inspector.
2. Specify the fully qualified class name of the tag library descriptor, or locate it using the File Chooser button. Fill in the other fields as required.
3. Right-click on the diagram pane to invoke the diagram right-click menu.
4. Select *Generate TLD* command.

Thus, the tag library descriptor file is written to the location specified in *TLD file name* field of the diagram inspector.

Using tag libraries

Tag handlers are referred to from the Web Application diagram.

To add a tag handler to the WAD:

1. Choose *TagLib* button on the Web Application diagram toolbar
2. Click on the diagram pane. This invokes *Select TagLib diagram* dialog in the form of [Selection manager](#).
3. Expand the desired node and select the taglib to be added.

Subsequently, working with JSP applications, specify which tag library to employ using the `tagLib` directive before any custom tag (refer to the on page 548, see tutorial for details).

Tag Library Helper

To insert the tags from your library:

1. Right click on the [JSP editor](#)
2. select **Tag Library Helper** command from the **Tools** node. The helper displays contents of the tag library according to the current library descriptor.
3. Choose the required tag

4. Click **Insert** to add the tag to the desired line in your JSP code.

Creating and mapping filters

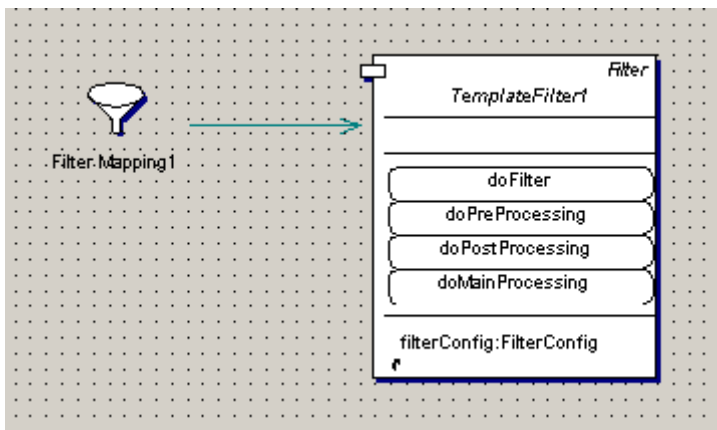
The objective of the *Filter Mapping* design component is to provide a filter mapping element for the deployment descriptor.

To create a filter or a collection of filters in a Web Application diagram:

1. Right-click on the Filter Mapping element and choose **Properties** to open its Properties Inspector.
2. Select the Mapped Filters tab.
3. Click **Add** to create a filter.
4. Add as many filters as required and press **CTRL+Enter** to apply changes and close the Inspector.

This generates a filter class that implements the `javax.servlet.Filter` interface. The Web Application diagram displays the shortcut of the filter element.

Figure 173



At the same time the corresponding elements are included in the default Class diagram. This class is attached to the Filter Mapping element by a Web link. The property Inspector of the filter class contains the *Filter Properties* tab, where you can specify the following properties:

- *Filter name*: used to map the filter to a servlet or URL.
- *Display name*: visible name of the filter.
- *Icon*: fully-qualified path to an image file used as the filter's icon.
- *Initialization parameters*: initialization parameters for a filter.

All filters of a Filter Mapping element are listed in its property Inspector.

NOTE: The Remove button in the filter mapping Inspector only deletes the link between a Filter Mapping and a filter. The filter class is still preserved on the diagram.

Use the *URL Pattern Mapping* tab of the filter mapping Inspector to define groups of servlets and static resources in the Web application to which the filter is applied. The rules of the path mapping are outlined in the Java Servlet Specification version 2.3, SRV 11.1, *Use of URL Paths*.

Each filter processes the current request, the current response, and the FilterChain containing filters to be executed.

Deploying filters

During deployment, filters are archived to a *.war file and deployed to the current application server. For each filter element the corresponding lines are generated automatically in the deployment descriptor.

For deployment, invoke the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).

NOTE: Filter mapping is defined only for application servers supporting EJB 2.0 specification.

Developing and designing listeners

Java Servlet API v. 2.3n supports event notifications for the state changes in the servlet context and http session objects. This gives you the ability to manage resources and state at the logical level of the Web application.

Application event listeners are Java classes that implement one or more of the servlet event listener interfaces.

Servlet event listeners support event notifications for state changes in the ServletContext and HttpSession objects. There may be multiple listener classes listening to each event type.

Event types

According to the Java Servlet 2.3 specification, the following event types are supported in Together:

- 1. Servlet Context Events. Lifecycle** - the servlet context has just been created and is available to service its first request or the servlet context is about to be shut down (`javax.servlet.ServletContextListener`).
- 2. Servlet Context Events. Changes to attributes**- attributes on the servlet context have been added, removed, or replaced (`javax.servlet.ServletContextAttributesListener`).
- 3. HttpSession Events. Lifecycle** - an HttpSession has been created, invalidated, or timed out (`javax.servlet.http.HttpSessionListener`).

4. HttpSession Events. Changes to attributes - attributes have been added, removed, or replaced on an HttpSession (javax.servlet.http.HttpSessionattributesListener).

- javax.servlet.ServletContextListener has the following predefined methods:
 - contextDestroyed(ServletConextEvent)
 - contextInitilized(ServletConextEvent)
- javax.servlet.ServletContextattributesListener has the following predefined methods:
 - attributeAdded(ServletContextAttributeEvent)
 - attributeRemoved(ServletContextAttributeEvent)
 - attributeReplaced(ServletContextAttributeEvent)
- javax.servlet.http.HttpSessionListener has the following predefined methods:
 - sessionCreated(HttpSessionEvent)
 - sessionDestroyed(HttpSessionEvent)
- javax.servlet.http.HttpSessionattributesListener has the following predefined methods:
 - attributeAdded(HttpSessionBindingEvent)
 - attributeRemoved(HttpSessionBindingEvent)
 - attributeReplaced(HttpSessionBindingEvent)

When an event takes place (the Web server shuts down or the application is removed from the Web server), the listener class is notified and then necessary actions in the system can be provided.

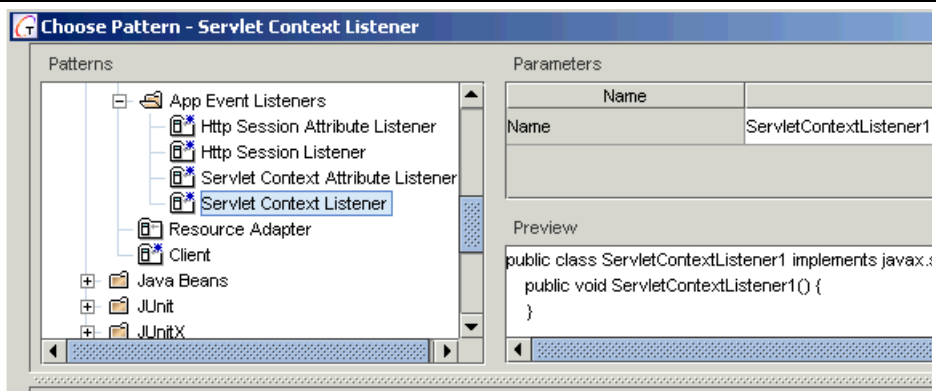
The number of listeners of each type is not limited.

Creating a listener

To create a listener:

1. Create a project for a listener. Choose **New Project Expert** on the File menu, and insert a project name and specify the required path for your project. Alternatively, you can open an existing project.
2. When ready, click **Finish** to complete and close the New Project Expert dialog.
3. Open a Class diagram. Choose the **Class by Pattern** command from the diagram toolbar or right-click menu. This opens the Choose Pattern dialog.
4. Using one of the four listener patterns, create a listener and define its name.

Figure 174

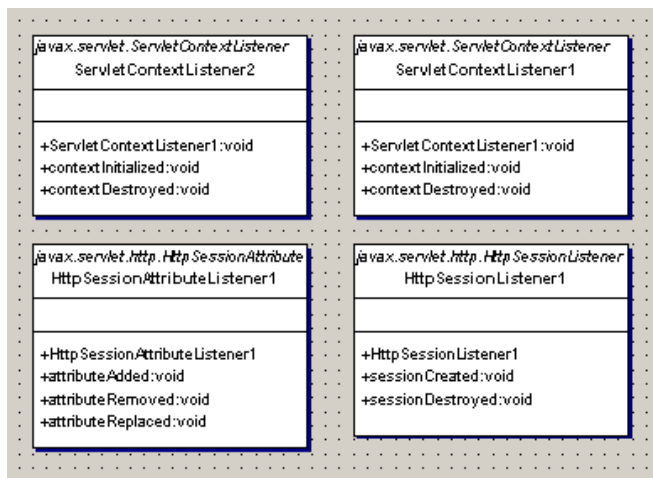


5. Click **Finish**; view the new listener on the Class diagram.
Repeat these actions to create several listeners.

Deploying listeners

Suppose you have already created the following listeners on the Class diagram:

Figure 175



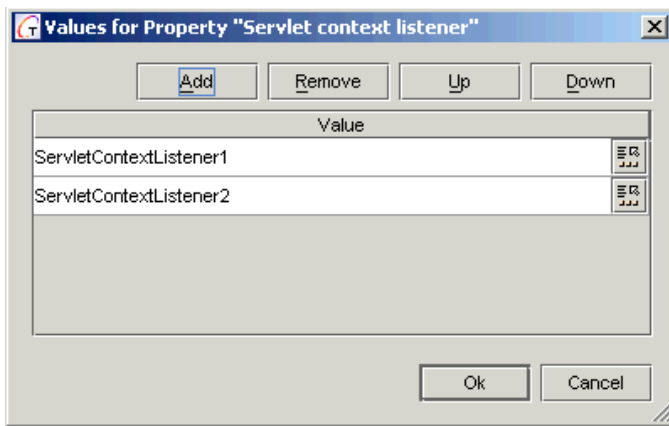
We have two listeners of the `ServletContextListener` type, one listener of the `HttpSessionAttributeListener` type, and one listener of the `HttpSessionListener` type.

To deploy listeners to an application server:

1. Create a Web Application diagram or open an existing one.

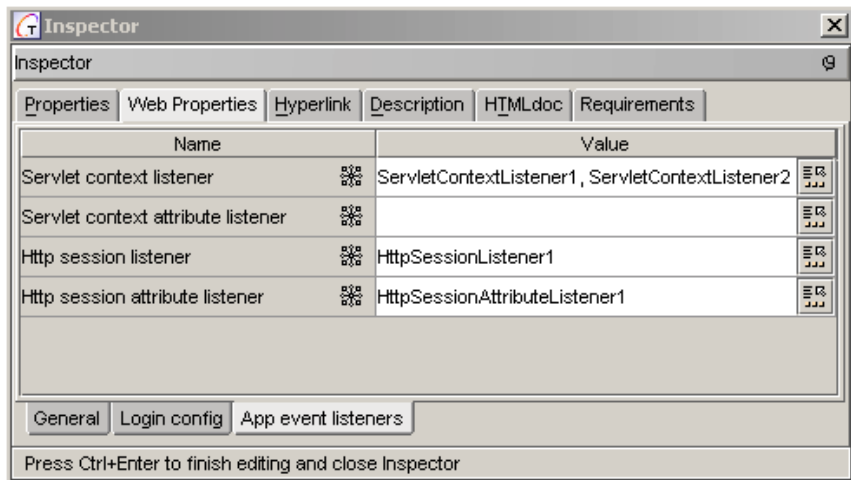
2. Open the Web Application diagram's Inspector, select the *Web Properties* tab, and go to the *APP event listeners* subtab. The next dialog opens (). There are four edit fields with buttons for the multi-string extended editor for each listener's type.
3. To include two listeners of the `ServletContextListener`, type in your Web application, click the file chooser on the editor button. In the dialog that opens, you can add, remove, and sort elements.
4. Click the **Add** button to add the first listener of this type. Choose the `ServletContextListener1` listener using the file chooser.
5. Click the **Add** button a second time to add the second listener of the same type. Choose the `ServletContextListener2` listener using the file chooser.
6. Use the **Add** and **Remove** buttons to add or remove listeners of this type.

Figure 176



7. The listeners will be processed in the order in which they are listed. Use the **Up** and **Down** buttons to change the order of the listeners in the list and to sort them.
8. In the same way, add the listeners of other types. After the process, the following list of listeners exist: See figure 171.

Figure 177



9. Open the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE” on page 584).

NOTE: Listeners are defined only for application servers supporting the EJB 2.0 specification.

How a Web Application diagram relates to the J2EE Deployment Expert

Together provides the J2EE Deployment Expert that simplifies the process of deploying servlets, JSPs, Web files, filters, application event listeners, and other files. Run the J2EE Deployment Expert against a Web Application diagram from the main menu: Deploy | J2EE Deployment Expert.

For “fast track” deployment with default values or deployment prototyping, use the J2EE Deployment Expert from an appropriate Class diagram. For “full-featured assembly information” with control over security and permissions, use the J2EE Deployment Expert from the Web Application diagram.

The J2EE Deployment Expert is used the same way as the EJB Assembler diagram. When deployment is performed from a Web Application diagram, Together takes files with the directory structure from the Web Files source folder.

Tips for Web Application diagrams

- The Properties Inspector of a JSP element (right-click menu | Properties) enables users to define the component's type (JSP or servlet) and servlet init parameters.
- Access the JSP Editor from a Web Application diagram.

- To define Web files, use a separate visual Web Files element placed on a Web Application diagram.
- Use the special design reference elements to facilitate working with EJB references, resources and environment variables.

For information, see Chapter 28, “Working with Container-Transaction Elements and References” on page 510.

- Provide security for your application using the special design elements: *Security Role*, *Principal*, *Security Constraint*, *Web Resource Collection*.
- Archive several Web applications and EJB modules in one EAR and deploy it to the current application server.

For information, see Chapter 30, “Designing and Developing Enterprise Applications” on page 558.

CHAPTER 30

DESIGNING AND DEVELOPING ENTERPRISE APPLICATIONS

This chapter discusses the following topics:

- “Visual assembly of Enterprise Applications for deployment” on page 558
- “Creating and drawing Enterprise Application diagrams” on page 559
- “Visual design elements” on page 560
- “How an Enterprise Application diagram relates to the J2EE Deployment Expert” on page 562

Visual assembly of Enterprise Applications for deployment

An Enterprise Application diagram is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

To support J2EE specifications, a tool is required that collects elements for an EAR file in a single diagram. This is the Enterprise Application diagram, which combines Web applications, EJB modules, classes, and connectors. In this way, an Enterprise Application diagram is used to visually assemble an enterprise application for deployment.

We assume that the developer who wants to create an Enterprise Application diagram already has one or more EJB Assembler, Web Application, and Resource Adapter diagrams.

An Enterprise Application diagram can contain shortcuts to EJB Assembler, Web Application, Resource Adapter, and Class diagrams. In addition, existing WAR, JAR, and RAR archive files can be included in an Enterprise Application diagram. This diagram provides all necessary elements for generating the deployment descriptor and creating an *.ear archive file.

NOTE: If the target application server does not support *.ear generation (which is the case for BEA WebLogic 5.1 and IBM WebSphere 3.5), multiple *.jar files are generated instead. BEA WebLogic 6.0 and 6.1 can deploy *.ear files, so a single *.ear file is generated when deployment is performed against the Enterprise Application diagram.

The Enterprise Application diagram also supports Security Roles. See Chapter 33, “J2EE Platform Security Support” for more information.

Thus, an Enterprise Application diagram is a visual equivalent of the enterprise application described by the EJB 2.0 Specification. According to the specification, the assembler assembles EJB modules, Web applications, and connectors into a single deployment unit (if the current application server supports EARs). It provides application assembly information to the application deployer, which is represented in Together by the J2EE Deployment Expert.

Creating and drawing Enterprise Application diagrams

There are two ways to create an Enterprise Application diagram:

- Create a new Enterprise Application diagram (using the Object Gallery or the diagram toolbar button) and then add enterprise components to it.
- Create a complete Enterprise Application using the Object Gallery. If you already have existing components you want to combine in a single enterprise application, this creates an Enterprise Application diagram that is ready for deployment.

Creating a new Enterprise Application diagram

To create a new Enterprise Application diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *General* in the categories on the left, and select the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Activate the Together tab, select the *Enterprise Application* icon, and click **Finish**.

Alternatively, you can click the New Diagram button on the diagram toolbar, go to the Together tab, and select Enterprise Application.

You should see the new Enterprise Application diagram with the default name *Enterprise Application*. Change this name using the diagram’s Inspector (right-click menu | **Properties**).

Develop the content of the diagram using visual design elements from the toolbar.

Creating an enterprise application using the Object Gallery

You can use the Object Gallery to create a new enterprise application for existing components if you have several EJB modules and Web applications that you want to combine in the same module for deploying.

To create an enterprise application:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Enterprise* in the categories on the left.
3. Select the *Enterprise Application* icon in the templates on the right. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the enterprise application. You can follow its recommendations, filling in the most important parameters, inserting names, or choosing paths. You do not need to work extensively with the Enterprise Application diagram - it is automatically created and populated with shortcuts to the components that you specify.

Visual design elements

All enterprise application visual components are represented in the toolbar on the left side of the diagram.

An *Enterprise Application* diagram contains the following design elements:

Table 65 Enterprise Application diagram design elements

Icon	Description
	Zoom: Magnifies the diagram. This is the same as using Zoom In on the right-click menu (or NumPad+). To zoom out, use the Zoom Out command on the right-click menu (or NumPad-).
	Module: Creates a visual shortcut to an existing EJB Assembler, Web Application, or Class diagram.
	Archived Module: Creates a visual shortcut to an existing *.jar file.
	Security Role: Defines the <i>security role</i> that stands for one of the recommended security roles for the EJB's clients. The <i>Security Role</i> element in an Enterprise Application diagram presents a simplified view of EJB applications security to the application deployer (such as the J2EE Deployment Expert).
	Note: Creates a visual Note element.
	Note Link: Creates a link between the note element and another visual component to show the note's relationship with this element.

The Enterprise Application diagram is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

Creating diagram shortcuts

By the time you are ready to assemble one or more diagrams into an Enterprise Application diagram, you should have existing Web Application, EJB Assembler, or Resource Adapter diagrams ready in your Together project. You add these diagrams to the Enterprise Application diagram as shortcuts. A diagram shortcut is just a visual representation of an element that resides somewhere else in the project.

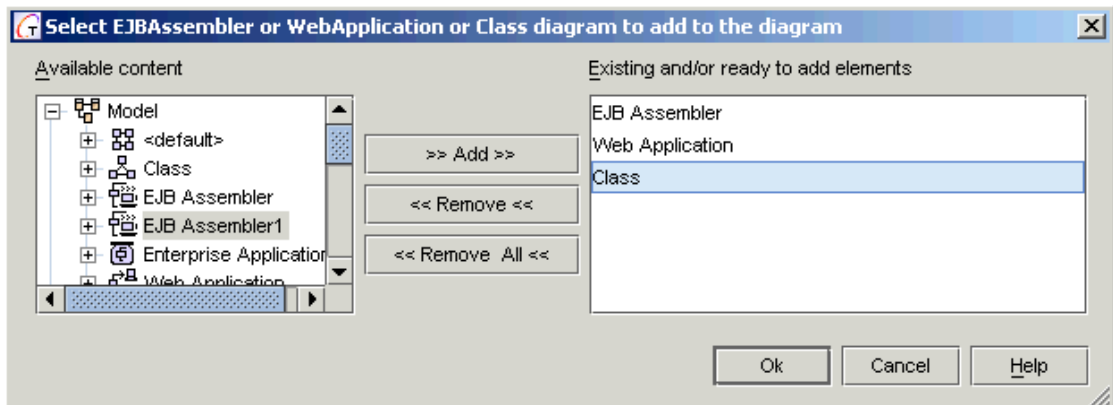
To create a shortcut to a diagram:

1. Create a new Enterprise Application diagram or open an existing one that contains Web Application and EJB Assembler diagrams.
2. To include diagrams that are not part of the current project, specify their paths in the *Search/Classpath* tab of the Project Properties dialog (Project | Project Properties).
3. Right-click on the background of the open Enterprise Application diagram and choose **New | Module**. Alternatively, you can click the Module button on the diagram toolbar and place the element on the diagram pane.

This invokes a selection manager dialog where you can choose available EJB Assembler, Web Application, Resource Adapter, or Class diagrams to be added to the current Enterprise Application diagram.

4. In the *Add Module* dialog, expand the *Model* node and locate the EJB classes and interfaces from your project that you want to display in the diagram. Select them in the tree view and click the **Add** button.
5. Still in the *Add Module* dialog, expand the *Search/Classpath* node and locate the EJB modules, Web applications or classes from outside your project (if any) to display in the diagram as shown in Figure 178. Select them in the tree view and click the **Add** button.

Figure 178

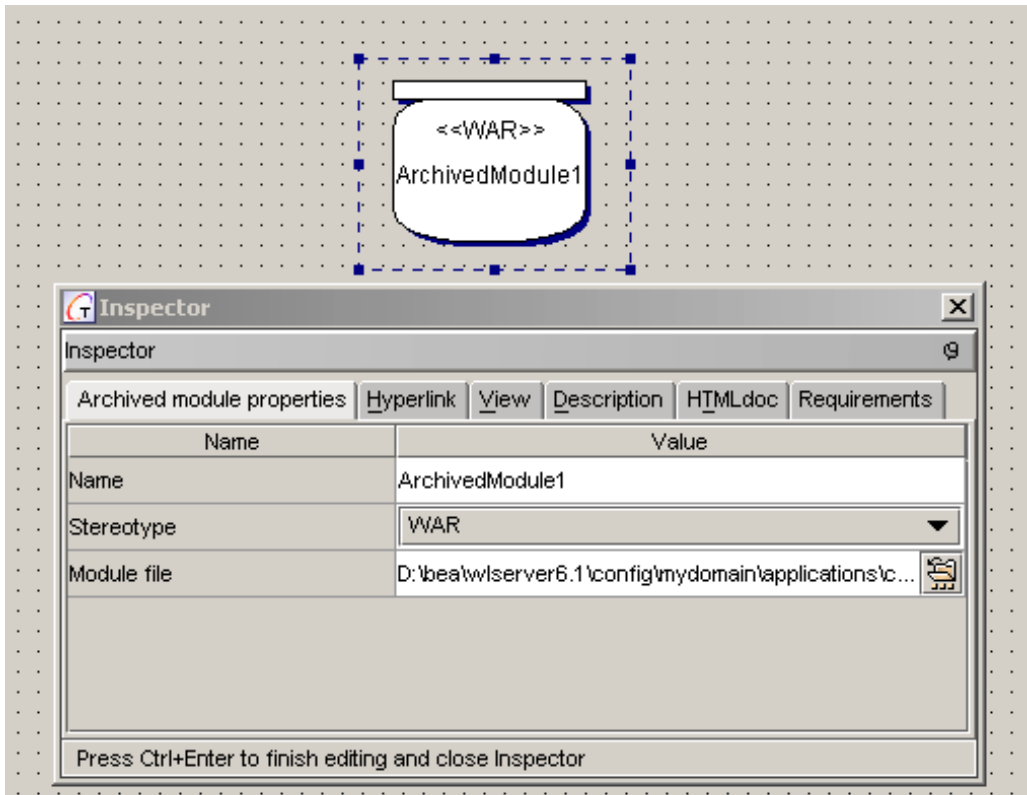


6. To include diagrams that are not part of the current project, specify their paths in the *Search/Classpath* tab of the Project Properties dialog (Project | Project Properties).

Importing existing archive files

1. To add existing JAR or WAR archive modules, choose the **Archived Module** design element from the toolbar and place this element on the diagram pane.
2. Open the Inspector to define the name of this element and its other properties. Choose *Stereotype* (WAR or JAR), and use the file chooser to find the full path to the necessary WAR, JAR, or RAR file.

Figure 179



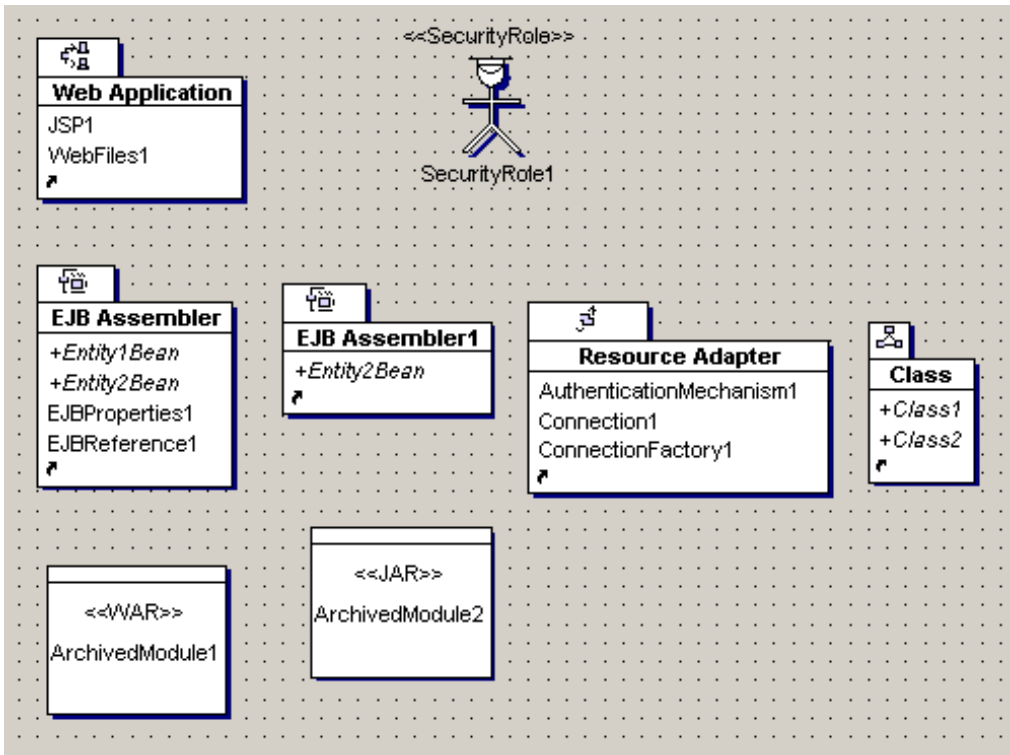
How an Enterprise Application diagram relates to the J2EE Deployment Expert

Together's deployment expert simplifies the process of deploying EJBs to various application servers. Together provides an "expert" dialog that simplifies the process of deploying enterprise applications.

To deploy an enterprise application:

1. Place all necessary shortcuts to EJB Assembler or Web Application diagrams and other design elements on the Enterprise Application diagram.
2. Define a security role for this diagram, if necessary.
3. Diagrams included in an Enterprise Application diagram are displayed in a kind of a visual container that includes all internal elements, as shown below.

Figure 180



4. To run the J2EE Deployment Expert against an Enterprise Application diagram, choose **Deploy | J2EE Deployment Expert** on the main menu.

CHAPTER 31

DESIGNING AND DEVELOPING APPLICATION CLIENTS

This chapter includes the following topics:

- “Overview of Application Client diagram” on page 564
- “Creating an Application Client” on page 565
- “Using an Application Client diagram” on page 567

Overview of Application Client diagram

Application clients are packaged in JAR format files with the `*.jar` extension, and they include a corresponding deployment descriptor describing enterprise beans and external resources referenced by the application. Access to resources is configured at deployment time, when the deployment descriptor is generated.

The tool used to deploy an application client and the mechanism used to install the application client are not specified in the J2EE specification. The way in which an end user invokes an application client is not specified either.

Together implements the Application Client diagram to improve e-commerce support. An Application Client diagram expands the possibilities of the standard UML and provides the ability to visually model client applications.

You can compile all elements placed on the Application Client diagram from this diagram.

When the deployment process of an application is initiated, a JAR file generates with a set of compiled classes and a deployment descriptor. The Client Application diagram contains all the necessary elements needed for deployment so that the deployment descriptor can be generated and, together with the client, added to the resulting JAR archive.

Creating an Application Client

If you have already created one or more Class diagrams and constructed a complete set of Java classes used in the application client, you can create a Client Application diagram and place shortcuts to classes on this diagram. You can also create one or more references to additional files, references to EJBs, and so on.

Creating a new Application Client diagram

To create a new Client Application diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *General* in the categories on the left, and select the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Activate the Together tab, select the *Application Client* icon, and click **Finish**.

Alternatively, click the New Diagram button on the diagram toolbar, and continue from step 3.

You should view the new Application Client diagram with the default name *Application Client*. To change the default name, use the Inspector (right-click menu | **Properties**).

Develop the content of the diagram using visual design elements from the toolbar.

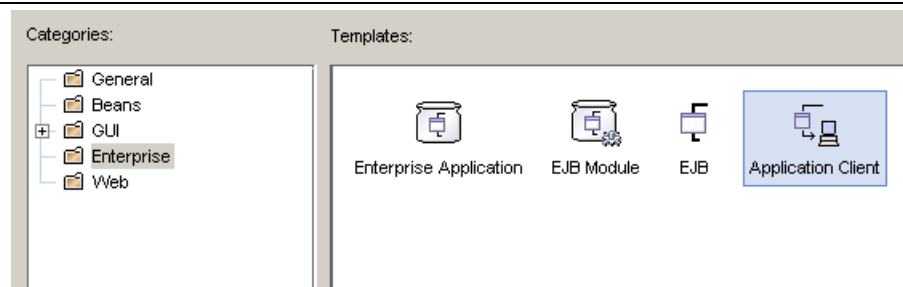
Creating a new application client using the Object Gallery

Use the Object Gallery to create a new application client (Application Client diagram).

To create a Web Application:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *Enterprise* in the categories on the left.

Figure 181

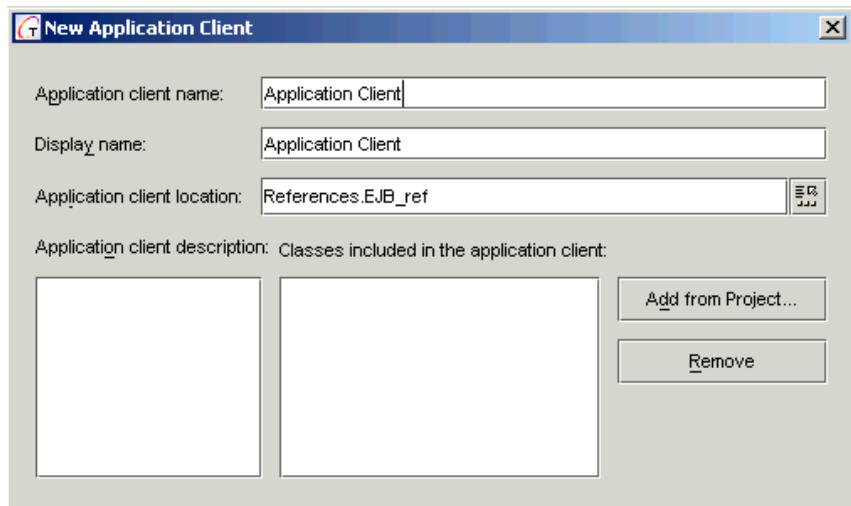


3. Select the *Application Client* icon in the templates on the right. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the application client. The Application Client diagram automatically creates the application client. Follow the recommendations in the Object Gallery, filling in the most important parameters. This procedure is particularly helpful if you have limited experience in the field of J2EE technology.

4. Fill in the **New Application Client** page in the expert. Set the name of the new Application Client diagram and its location. Add the existing elements you want to include in this diagram. See online help for details.

Figure 182



5. Click **Finish**. This creates the Application Client diagram that represents your application client.

Visual design elements

All EJB Application Client visual components are represented in the toolbar on the left side of the diagram. The design elements are listed in Table 66.

An Application Client diagram contains only shortcuts (it cannot contain classes themselves). These classes already exist on the Class diagram.

Table 66 Application Client diagram design elements

Icon	Description
	Zoom: Magnifies the diagram. Access the Zoom tool on the right-click menu (or NumPad+). To zoom out, use the Zoom Out command on the right-click menu (or NumPad-).

Table 66 Application Client diagram design elements (continued)

Icon	Description
	Client Classes shortcuts: Opens a selection manager dialog box, where you add or remove shortcuts of client classes on your Application Client diagram.
	EJB Reference: Creates an element with its own properties. In the visual model, it represents a reference to the corresponding EJB. Define a remote type for EJB reference: EJB, or EJB local (for the EJB 2.0 specification).
	Environment Reference: Creates a visual design component with the properties of some static constant, which cannot be changed after EJB deployment.
	Resource Reference: Creates a visual design component that has all properties of the referenced resource. Define the type of reference: resource, or resource-environment (for the EJB 2.0 specification).
	Note: Creates a visual Note element.
	Note Link: Creates a link between the note element and another visual component to show the note's relationship with this element.

Creating “one-click” application clients

To create a “one-click” application client:

1. Create or open a Together Project, and create or navigate to the package where you want to create an application client.
2. Create or open a Class diagram in the target package.
3. Create client classes on the Class diagram.
4. Create an Application Client diagram, and name it.
5. Click the **Client Classes Shortcuts** button and place it on the Application Client diagram. Use the chooser to select the classes to add to the diagram as shortcuts.
6. Add other necessary design elements (such as environment and resource references) on the diagram.

Using an Application Client diagram

You can use all of the elements placed on an Application Client diagram, like a separate module with the diagram's name:

- Compile classes from each client module separately.
- Generate a client XML deployment descriptor for each client module.
- Package compiled classes and a deployment descriptor into a JAR for each client module.

NOTE: Generating the deployment descriptor and packaging the application client and its deployment descriptor into a JAR file are possible only from the Enterprise Application diagram or EJB Assembler diagram.

To start the deployment process:

1. Create or open an Enterprise Application or EJB Assembler diagram.
2. Click the **Shortcuts** button and place it on the current diagram. Use the file chooser to select the necessary application client.
3. Place other necessary design elements on the current diagram.
4. Choose **Deploy | J2EE Deployment Expert** on the main menu to invoke the J2EE Deployment Expert and to start the deployment process (see “Deploying J2EE” on page 584).

CHAPTER 32

DESIGNING AND DEVELOPING RESOURCE ADAPTERS

This chapter includes the following topics:

- “Integration Problems and Connector Architecture” on page 569
- “Creating a Resource Adapter diagram” on page 570
- “Using a Resource Adapter diagram” on page 573

Integration Problems and Connector Architecture

It is often necessary to join a new Java application with existing enterprise applications or external applications. For example, sometimes data from one system is transmitted to another.

The facilities of the Enterprise Application Integration (EAI) solve this problem for both enterprise applications and “business-to-business” applications. EAI includes different technologies for integrating different enterprise systems. Java EAI should collaborate with other EAIs.

J2EE 1.3 includes a connector architecture that can be used specifically for integration purposes.

J2EE connectors define the standard Connector Service Provider Interface (Connector SPI) that should be implemented by a provider of other Enterprise Information Systems (EISs). An EIS provider also provides resource adapters for connecting with control systems of a J2EE resource adapters container (server) implementing Connector SPI EIS. This way, J2EE containers can control EIS service tools.

The Connector SPI defines three types of interfaces:

- Resource managing interface

A resource managing interface provides the control for pooling connections with EISs, and then receiving connections by the user’s request to the current J2EE server.

- Transactions managing interface

A transactions managing interface provides a control for executing the distributed transactions (several at the same time) to the current J2EE server.

- Security managing interface

A security managing interface provides security accessing components from the J2EE container to the EIS.

EJBs get resource references to resources of a specific provider through JNDI, and find the necessary resources using the `InitialContext.lookup()` method and `<res-ref-name>`. So, a container can provide for controlling resources of EJBs.

Creating a Resource Adapter diagram

If you already have the necessary `ConnectionFactory` and `ManagedConnectionFactory` classes, you are ready to create a resource adapter. As in the case of EJB shortcuts, you can use shortcuts to these classes on a Resource Adapter diagram.

To create a new Resource Adapter diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, select *General* in the categories on the left, and select the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Activate the Together tab, select the *Resource Adapter* icon, and click **Finish**.

Alternatively, click the **New Diagram** button on the diagram toolbar, go to the Together tab, and select Resource Adapter.

You should see the new Resource Adapter diagram with the default name *Resource Adapter*. To change the default name use the Inspector (right-click menu | **Properties**). Develop the content of the diagram using visual design elements from the toolbar.

NOTE: To add a shortcut of an existing connection class, choose **New | Shortcuts** from the right-click-menu.

Visual design elements

All EJB Resource Adapter visual components are represented in the toolbar on the left side of the diagram.

A Resource Adapter diagram contains only shortcuts to connection implementation classes and their interfaces (it does not contain classes themselves).

A Resource Adapter diagram contains the following design elements:

Table 67 Resource Adapter diagram design elements

Icon	Description
	Zoom: Magnifies the diagram. Access the Zoom tool on the right-click menu (or NumPad+). To zoom out, use the Zoom Out command on the right-click menu (or NumPad-).

Table 67 Resource Adapter diagram design elements

Icon	Description
	Config Property: Contains a declaration of a single configuration property for a ManagedConnectionFactory instance (an option description, a name, a type and an optional value for a configuration property). This component corresponds to the <config-property> attribute in the deployment descriptor.
	Authentication Mechanism: Specifies a type of an authentication mechanism. Any additional security mechanisms are outside of the scope of the connector architecture. This component corresponds to the <authentication-mechanism> attribute in the deployment descriptor.
	Security Permission: Specifies a security permission based on the Security policy file syntax. It defines what permissions (which types of system resource accesses) are allowed by code from specified code sources. Refer to the corresponding URL of the security specification. This component corresponds to the <security-permission> attribute in the deployment descriptor.
	Connection: A visual design component that can be linked with a connection interface and with a connection implementation class.
	Connection Factory: A visual design component that can be linked with a connection factory interface and with a connection factory implementation class.
	Resource Adapter Link: Provides linking for Connection/Connection Factory elements with the corresponding interfaces and implementation classes.
	Note: Creates a visual Note element.
	Note Link: Creates a link between the note element and another visual component to show the note's relationship with this element.

Creating a Resource Adapter using a pattern

To create a resource adapter diagram using a pattern:

1. Create or open a Together Project, and create or navigate to the package where you want to create a resource adapter.
2. Create or open a Class diagram in the target package.

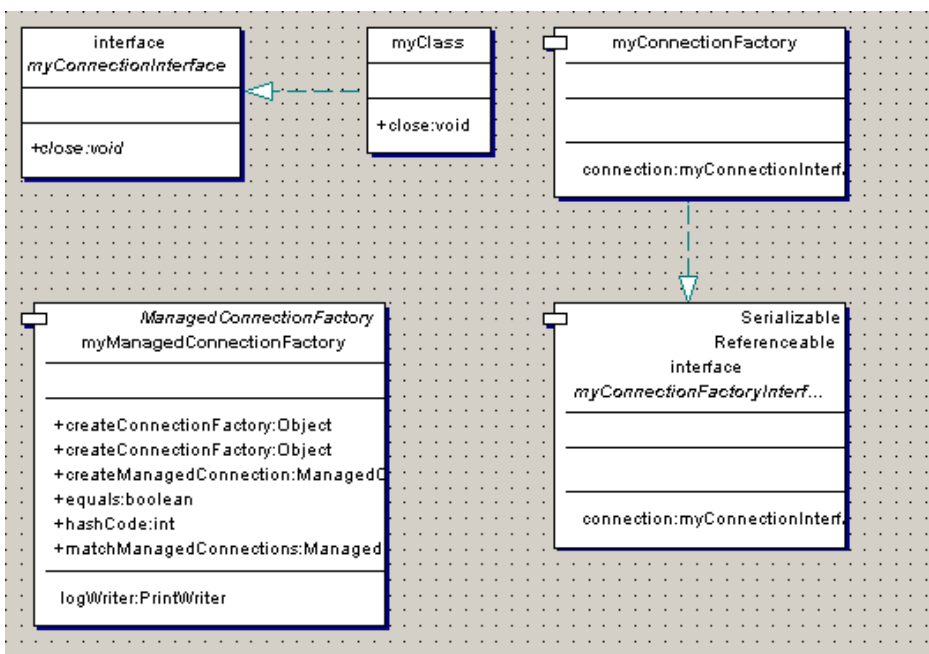
3. Click on the **Class by Pattern** button and place it on the Class diagram. Use the file chooser to select the pattern from the tree: *Patterns - J2EE - Resource Adapter*.
4. Fill in the lines on the right side of the window (insert the name of your future Resource Adapter diagram and other necessary names).

Figure 183

Parameters	
Name	Value
Resource Adapter Diagram name	Resource Adapter
Managed Connection Factory class name	myManagedConnectionFactory 
Connection Factory interface name	myConnectionFactoryInterface 
Connection Factory class name	myConnectionFactory 
Connection interface name	myConnectionInterface 
Connection class name	myClass 

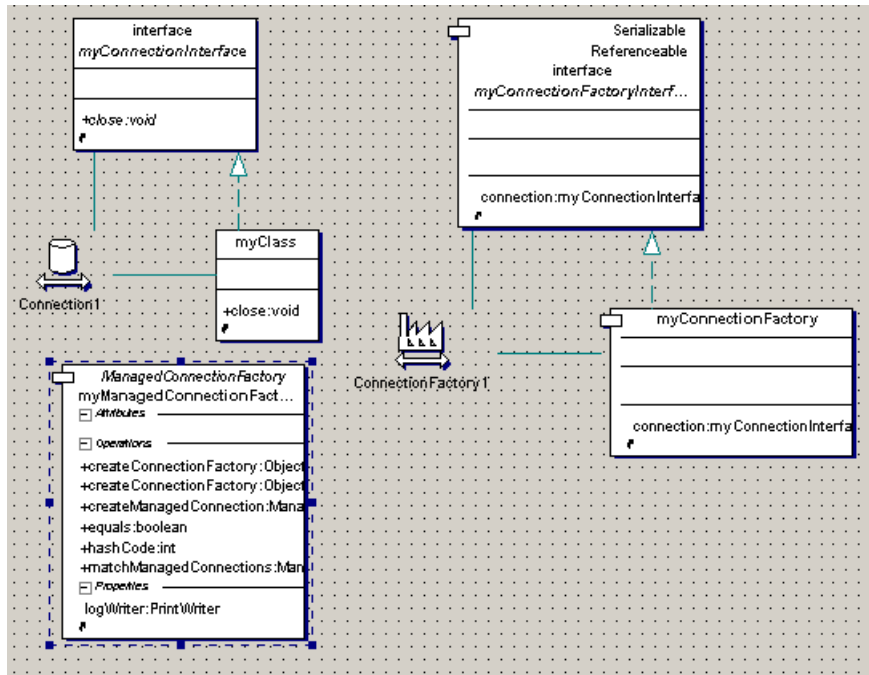
5. Click **Finish**. The defined classes and their interfaces appear on the current Class diagram.

Figure 184



- At the same time, an empty Resource Adapter is created with the name that you inserted in step 4, or the default name *Resource Adapter* if you did not type anything in this field. Open this diagram to see the result.

Figure 185



- Add other design elements on the diagram and change their properties, if necessary.

Using a Resource Adapter diagram

Use all elements placed on a Resource Adapter diagram together like a separate module with the diagram's name:

- Generate an XMLs deployment descriptor for every resource adapter diagram.
- Package compiled classes and a deployment descriptor into RARs for each Resource Adapter diagram.

To start the deployment process:

Choose **Deploy | J2EE Deployment Expert** on the main menu to open the J2EE Deployment Expert and start the deployment process.

J2EE PLATFORM SECURITY SUPPORT

This chapter includes the following topics:

- “Security roles” on page 574
- “Security References support” on page 574
- “Process overview” on page 575
- “Security support in EJB modules” on page 576
- “Security support in Web applications” on page 579
- “Security support in enterprise applications” on page 583

Security roles

Users have a variety of access rights to the application or elements of the application. The Security Role design element is used on almost all J2EE diagrams. Placing this element on the diagram and connecting it to the Principal element or to another design element defines security conditions for this principal.

The security support methods are different for the different types of J2EE diagrams.

A Security Role design element defines a security role that represents one of the recommended security roles for the EJB's clients.

A Principal and a Security Role are defined as separate design elements in Together. A Principal is an Actor element separated from its security features. A Principal can be linked to a Security Role.

Security References support

To provide controlled access to an EJB-method, the declarative authorization specified in the deployment descriptor is used. The necessary EJB method is associated with a Method-Permission element of the deployment descriptor, which contains a list of methods that can be accessed by users with a certain security role.

The deployer maps a security role to a user group in the operational environment, or maps a security role to a principal name in the security policy domain. In the latter case, the principal name of the calling principal is retrieved from its security attributes.

If the principal (or group) has a security role that allows access to this EJB method, the security system allows the principal to call and execute this EJB method. The same technique is used for the protection of Web resources (see “Authentication and authorization in Web applications” on page 580).

An EJB security role reference that is defined as one of the EJB attributes can be used for linking a Security Role with a Servlet or JSP. A Security Role can be linked with an EJB via an EJB security role reference in the EJB implementation class.

If a JSP (or servlet) defines any Security Role, it must be linked to the corresponding Security Role element.

Process overview

If you have a business-logic implemented with EJBs, create an EJB Assembler diagram and add the EJB shortcuts. Edit the EJB Assembler diagram to define resources, environment variables, security roles, links between elements, and so on. Use the J2EE Deployment Expert for generating the deployment descriptor and deploying a JAR archive.

If you have a user interface implemented via JSPs and servlets, create a Web Application diagram and add the shortcuts to JSPs, servlets, and EJBs. Create an Enterprise Application diagram and add shortcuts to the necessary Web Application and EJB Assembler diagrams.

- Edit the Enterprise Application diagram and use the J2EE Deployment Expert to generate the deployment descriptor and deploy the EAR archive.
- The Inspector for the Security Role element (right-click menu | Properties) enables you to define the name of the element.
- Create a link from one or more Method Permissions to the security role. This defines method permissions for this security role.
- Create a link from the Security Role to one or more Principals. This defines the User or the Group for this security role.
- If an EJB defines any EJB security role references (shown in the last compartment of the EJB icon in the diagram), all of them must be linked to Security Role elements.
- Each Security Role Reference must have one and only one link to some Security Role element.
- One Security Role element can be associated with any number of EJB security role references or with none at all.

Create a link from a method in an EJB implementation class to an EJB Container Transaction element or to a Method Permission element.

Security support in EJB modules

Each EJB module is represented by an EJB Assembler diagram in Together. A Security Role element in an EJB Assembler diagram presents a simplified view of the EJB applications security to the J2EE Deployment Expert.

The J2EE specification states that “... the J2EE authorization model is based on the concept of security roles. A security role is a logical grouping of users that is defined by an Application Component Provider or Assembler. It is then mapped by a Deployer to security identities (e.g., principals, groups, etc.) in the operational environment. A security role can be used either with declarative security or with programmatic security.”

Together implements two variants of the Security Role definition for an EJB Assembler diagram:

- Declarative Security Role
- Programmatic Security Role

According to the J2EE specification, declarative security is realized in a form external to the application: “The deployment descriptor is the primary vehicle for declarative security in the J2EE platform. A deployment descriptor is a contract between an Application Component Provider and a Deployer or Application Assembler. In the context of J2EE security, it can be used by an application programmer to represent an application’s security-related environmental requirements. Groups of components are associated with a deployment descriptor. The application’s logical security requirements are mapped by a Deployer to a representation of the security policy that is specific to the environment at deployment time. A Deployer uses a deployment tool to process the deployment descriptor. At runtime, the container uses the security policy that was derived from the deployment descriptor and configured by the Deployer to enforce authorization.”

For this purpose, Together provides a special visual element called Method Permission.

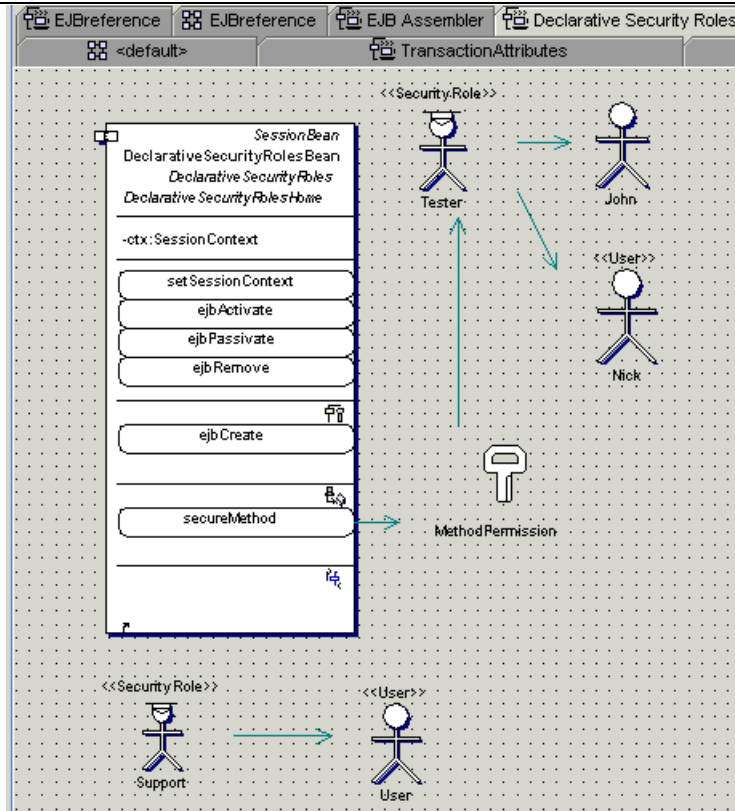
Declarative security role

To provide declarative security:

1. Create an EJB Assembler diagram, or open an existing one.
2. Place shortcuts for design elements necessary to your project (EJBs, classes, and others) on this diagram.
3. Define a method to be secured in some EJB element (secureMethod).
4. Place a Method Permission design element on the current EJB Assembler diagram.
5. Place a Security Role design element on the current EJB Assembler diagram. Set the name for this security role (this name will be used in the deployment descriptor).
6. Place a Principal design element on the current EJB Assembler diagram, and set its properties.
7. Link the design element. For example, link session EJB `DeclarativeSecurityRolesBean` with `Security Role1`. Then connect this Security Role element with the Principal element `User1`. As a result, the access of `User1` to the servlet `MyServlet1` is defined by `Security Role1`.

8. Place a Security Role design element on the current diagram and link it with some principal element without any connection to a concrete method. You can see this variant on 8., where the Support security role is linked with the User principal. In this case, lines about the Support security role of the User will be added to the deployment descriptor.

Figure 186



Programmatic security role

Security-aware applications uses programmatic security . According to the J2EE Specification, “Programmatic security consists of two methods of the `EJBContext` interface and two methods of the `HttpServletRequest` interface:

```
isCallerInRole (EJBContext) getCallerPrincipal (EJBContext) isUserInRole
(HttpServletRequest)
getUserPrincipal (HttpServletRequest)
```

These methods allow components to make business-logic decisions based on the security role of the caller or remote user. They also allow the component to determine the principal name of the caller or remote user to use as a database key, for example.”

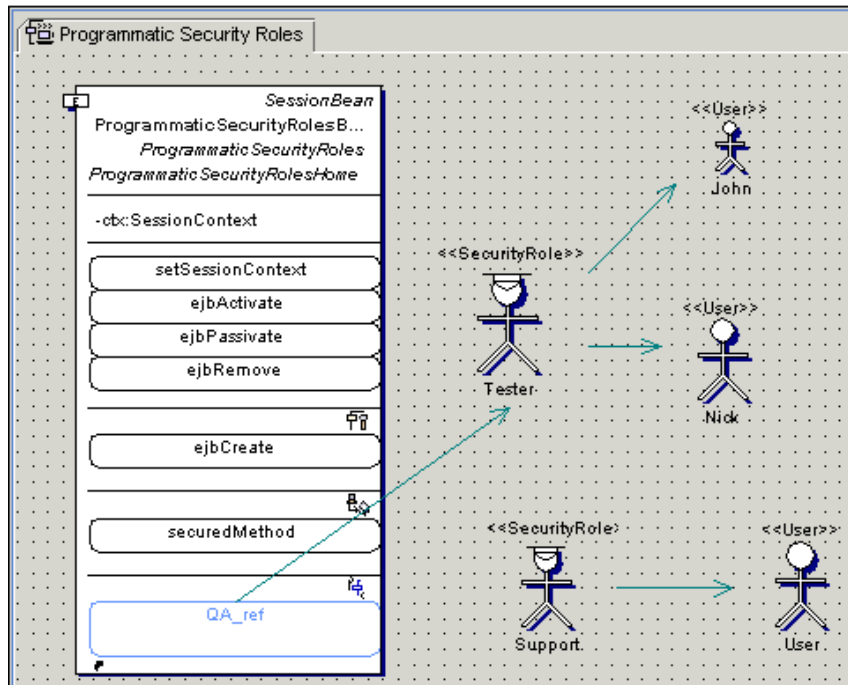
To provide programmatic security:

1. Create an EJB Assembler diagram or open an existing one.
2. Place shortcuts for design elements necessary to your project (EJBs, classes, and others) on this diagram. For example, see `ProgrammaticSecurityRoleBean` in .
3. Add an EJB security reference in the EJB. For example, see `QA_ref` in .
4. Place a Security Role design element on the current EJB Assembler diagram. Set the name for this security role (for example, `Tester`). This name will be used in the deployment descriptor.
5. Place one or more Principal design elements on the current EJB Assembler diagram and set their properties.
6. Link the EJB security role element. For example, link `QA_ref` with the security role `Tester` . Then connect this Security Role element with the Principal elements `John` and `Nick`. As a result, the access of these users to the `ProgrammaticSecurityRoleBean` is defined by the `Tester` security role.
7. Place a Security Role design element on the current diagram and link it to some principal element without any connection to a concrete method. You can see this variant where the `Support` security role is linked to the `User` principal.

In this case, you have to add Security Role icons to the EJB Assembler diagram (`support` and `tester` in the example) and link them to the corresponding principals (`John` and `Nick`).

If an EJB has a security role reference, define EJB security reference (`QA_ref`) as an attribute of EJB. This security reference can be linked to the Security Role icon (`tester` in our case). Then the J2EE Deployment Expert will generate information about properties of `tester-reference` in the deployment descriptor.

Figure 187



Use the Inspector for a Security Role element (right-click menu | Properties) to define the element name.

You can create a link from a Security Role to one Principal.

Security support in Web applications

There are several visual design elements that support the security of your project:

- Security Role
- Security Constraint
- Web Resource Collection

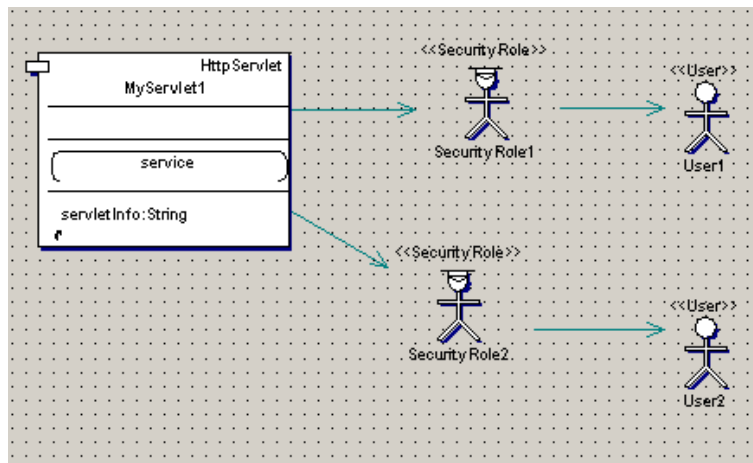
Using a Security Role design element for supporting security

On a Web Application diagram, a Security Role is a visual design component for a security role, meaning one of the recommended security roles for Web Applications or EJB clients. A Security Role element on a Web Application diagram presents a simplified view of the JSP's (or Servlet's, or EJB's) security to the J2EE Deployment Expert.

To provide a security role for the user:

1. Create a Web Application diagram, or open an existing one.
2. Place the design elements necessary to your project (shortcuts of servlets, JSPs, Web files, and others) on this diagram.
3. Place a Security Role design element on the current Web Application diagram. Set the name for this security role (this name will be used in the deployment descriptor).
4. Place a Principal design element on the current Web Application diagram, and set its properties.
5. Link the shortcut of the design element. For example, link Servlet `MyServlet1` to `Security Role1`. Then connect this Security Role element to the Principal element `User1`. As a result, access of `User1` to the servlet `MyServlet1` is defined by `Security Role1`.

Figure 188



Authentication and authorization in Web applications

Users often want to log in to the system, register to access a protected area, and log out from an application server.

For example, to provide the following scenario:

When a user tries to access a protected area, the login page prompts for the user name and password. If one of them or both are wrong, the user sees the login error page. Otherwise, the user sees the login page.

To get authentication and authorization in a Web application:

1. Grant write permissions to the `realm` file. To do this, you have to modify the server `.policy` file that is placed in the `%J2EE_HOME/lib/security/` directory. Find this file in the `%TG_HOME/bundled/j2ee/lib/security/` directory. Open this file for editing and add the following line to the section where default permissions are granted to all domains:

```

    permission java.io.FilePermission
    "${com.sun.enterprise.home}${file.separator}repository${file.separator}
    r}-" , "read,write,delete";

```

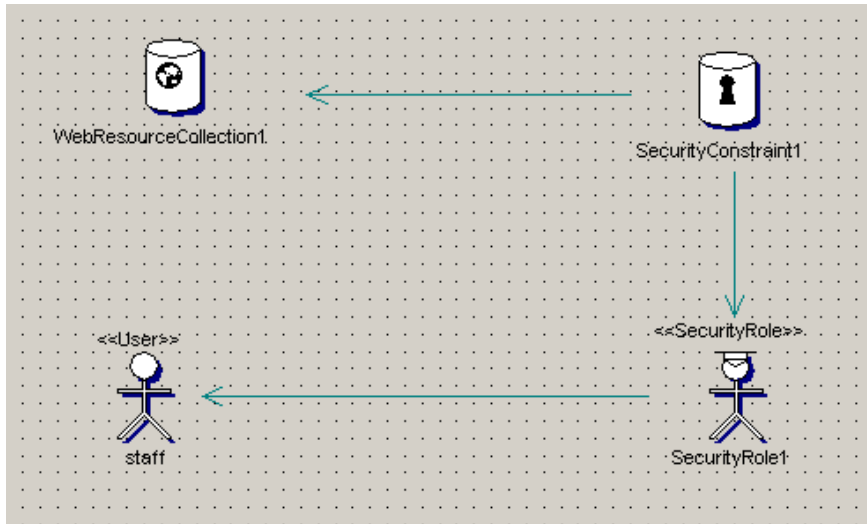
NOTE: This line will allow full access to your granted directory for all Web applications.

2. Create a new project and specify its location. Add `j2ee.jar` to the **Search/Classpath** tab. (The `j2ee.jar` file is located in `%TG_HOME/bundled/j2ee/lib/.`)
3. Prepare the utility java classes and necessary JavaServer pages that you are going to use for authentication.

For example:

1. Create the `Webfiles` package, and include in this package the JSP that you are going to define as protected, the default `index.html` file to show first, additional images, and so on.
2. Create the `AuthenticationAdapter` class - a utility java class for adding new users to the `realm` file.
3. Create the servlet `AddUser` that adds users using the `AuthenticationAdapter` class.
4. Create the `GetImage` servlet to show the image corresponding to the login user type (one image for a member, and a different image for a guest).
4. Create a new Web Application diagram.
5. Define the properties of the Web Application diagram using the **Web Properties** tab, **General** subtab in the Inspector. *Module name* and *Display name* are the parameters of the deployment descriptor.
6. Choose a path in *Welcome file list* to show the default file instead of showing the content of directories. Naturally, an `index.html` file should be prepared for this in the `webfiles` package. Use the file chooser to provide `index.html` as a welcome file.
7. Define the user authentication options on the **Login Config** tab of the **Web Properties** tab in the Inspector:
 1. Choose `FORM` as *Authentication method*, and set *Realm name* to default.
 2. In *Form login page*, choose the path to the prepared login JSP `login.jsp`.
 3. In *Form error page*, choose the path to the JSP file that should be shown in the case of errors (`login_error.jsp` should be prepared first).
 4. Set *Location from login* to `/login.jsp`.
 5. Set *Location for error page* to `/login_error.jsp`.
8. Click on the *Web Files* icon on the diagram toolbar and place it on the current Web Application diagram. Choose the path to the source directory with your Web files (`Webfiles` package) that you want to include in the project in the *Web files directory*.
9. Add shortcuts to all necessary servlets and classes on the current Web Application diagram.
10. Click on the *Web Resource Collection* button on the diagram toolbar and place it on the current Web Application diagram. Set the properties of this element using the Inspector, for example:

1. Set `/protected/*` value to the *URL pattern* property.
2. Set `GET` value to *HTTP method*.
11. To create an authorization constraints element, click on the Security Constraint button on the diagram toolbar and place it on the current Web Application diagram.
12. In the same way, place a Security Role element and a Principal element on the current diagram. Use the Inspector for the Principal element to set the *Stereotype* property to *Group*. Set the name property to `staff`.
13. To provide access to Web resources in your `/protected/` directory in the Web application only for the `staff` group, draw the following links between security supported elements:



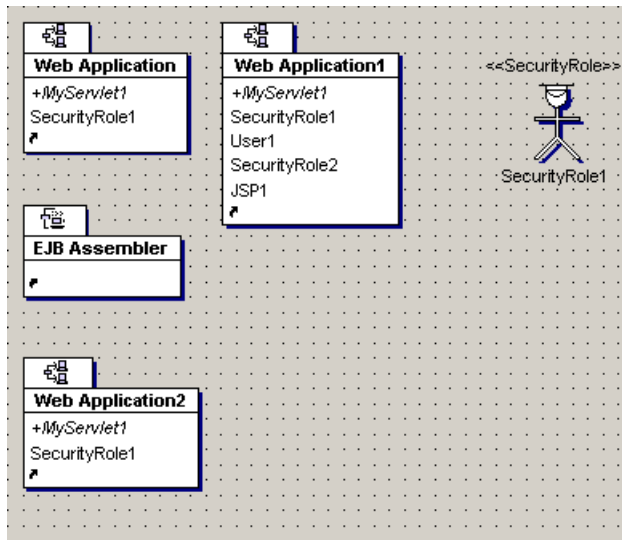
14. Start an application server. Invoke `%TG_HOME/bundled/j2ee/bin/j2ee.bat` to start SunEE Reference Implementation (1.3) Application Server.
15. Choose **Deploy | J2EE Deployment Expert** on the main menu to invoke the J2EEDeployment Expert. Select *SunEE Reference Implementation (1.3)* as the target server. Make sure that *Generate simple JSP client*, *Generate Command Line file*, and *Open XML Editor* options on the Process Options page are unchecked. Make sure that all other options are checked. Click **Next**.
16. Provide paths to the JDK and J2EE, click **Next** twice, and then **Finish** to start the deployment process.
17. After successful deployment, run your application. Open `http://localhost:8000/<Module name>/url` in your browser, and work with the protected resource.

NOTE: Remember that the protected resource stays in the browser's cache for some period after signing out; this depends on your browser.

Security support in enterprise applications

A Security Role design element defines the security role that represents one of the recommended security roles for the application clients. A Security Role element on an Enterprise Application diagram defines the security properties for all elements on the Enterprise Application diagram. The J2EE Deployment Expert generates the corresponding lines connected with a security role in the deployment descriptor.

Figure 189



CHAPTER 34

DEPLOYING J2EE

This chapter discusses the following topics:

- “Deployment process” on page 584
- “Supported application servers” on page 585
- “Generic server options” on page 585
- “Requirements for deployment” on page 586
- “Starting application servers from within Together” on page 586
- “Using the J2EE Deployment Expert” on page 587
- “Server-specific information” on page 590
- “Making transitions between application servers” on page 614

Deployment process

J2EE deployment is a process of generating deployment descriptors and packing archive files (*.jar, *.war, or *.ear) on the selected application server.

For example, assume that you have created an e-commerce or other enterprise application based on J2EE technology. This application might contain multiple EJBs, JSP files (Java Server Pages), Servlets, and other components.

Next, you need to select a specific application server, prepare the corresponding deployment descriptors (and possibly other necessary files for certain application servers), pack all files in the archive (*.jar, *.war, *.rar, *.ear), and deploy the result to a specific location. After deployment, the application is ready for use.

Together automates this procedure. Using the J2EE Deployment Expert, you can generate deployment descriptors, verify and correct your code according to the EJB specification used, compile all files, pack them in *.jar, *.war, *.rar, or *.ear files, and deploy to the selected application server. You can optionally generate a simple JSP client, which you can use to test the deployed EJB running on the application server.

In Together, J2EE deployment is implemented for the most widely-used application servers. On server platforms that support it, hot deploying directly to the application server is also available, and you can start some application servers from Together without exiting.

Supported application servers

To see a list of the currently supported servers:

1. Open a project.
2. Choose **Deploy | J2EE Deployment Expert** on the main menu.
3. On the first page of the expert, the drop-down list of application servers shows the server platforms currently integrated.

NOTE: The J2EE Deployment Expert is available for any of the following diagrams: Class, Enterprise Application, EJB Assembler, Web Application, Resource Adapter.

You can integrate other application servers and add them to the list of the supported application servers.

To find out information about the application servers that currently can be integrated:

1. Choose **Help | On the Web | Integration Updates** from the main menu.
2. Choose the J2EE Application Server hyperlink on the TogetherSoft Web site. You can see the current list of J2EE Application servers supported in Together.

This list provides the following information:

- The level of integration (*High, Medium, Low*).
- Where you can find this application server (*In Build, Download*).

If there is a high level of integration for the current application server, Together provides deployment from within Together for this application server.

If the location of this application server is *In Build*, you can find it in the `%Together/bundled/` directory (for example, Sun Reference Implementation, J2EE 1.2.1, or Tomcat).

If the location is *Download*, you can download this application server using the corresponding hyperlink.

If the level of integration is high for this application server, use the J2EE Deployment Expert for deployment. Otherwise download this application server and read the help on the TogetherSoft Web site about the possibilities of integration.

Generic server options

For development purposes, you can generate EJB 1.0, EJB 1.1, or EJB 2.0 compatible deployment descriptors, without choosing a specific application server. To do this, select Generic 1.0, Generic 1.1, or Generic 2.0 as the application server.

Requirements for deployment

Before deploying, the following requirements are necessary:

- An installation of Together with full deployment support.
- An accessible installation of the target application server with appropriate access rights.
- An accessible installation of Java Enterprise Edition 1.2 or higher.
- An accessible installation of Java 2 SDK (JDK 1.2 or higher).
- An accessible temp directory and sufficient disk space for temporary files that are generated by the J2EE Deployment Expert.
- An http-accessible location on the server for generated JSPs (if generating a JSP test client).

NOTE: Before deployment, Together checks whether an old *.jar file exists, and tries to delete it. If deletion is not possible, the J2EE Deployment Expert does not start, and an error message is displayed. This does not apply to BEA WebLogic 6.0 and 6.1.

WARNING! The Java Virtual Machine invokes the Together built-in compiler on the same JVM where JARs are placed. As a result, JAR files might be locked by Together after compilation. To avoid this situation, it is recommended to use an external compiler (you can set up the external compiler in the Options dialog). For a complete description of the problem, refer to the `Readme.html` file in the root of your Together installation.

Starting application servers from within Together

Together provides the following plug-ins:

- Start BEA WebLogic Application Server 5.1
- Start BEA WebLogic Application Server 6.0
- Start BEA WebLogic Application Server 6.1
- Start Sun EE

To start an application server:

1. Before starting the selected application server, make sure that all properties of this server are defined correctly (choose **Properties** on the right-click menu).
2. To start some application servers from within Together, you need to make changes in the corresponding Start command file. In this case, choose the command **Edit Start Command File** on the Server Explorer right-click menu. For details, see step 4 under “Starting BEA WebLogic Application Server from within Together” on page 605.
3. Choose the command **Edit Stop Command File** on the Server Explorer right-click menu, if it is necessary to edit the corresponding Stop command file.
4. To launch the required server from Together, right-click on the server under *Application Servers* in the Server Explorer. Choose **Start** on the right-click menu.

An alternative method to start an application server:

1. Open a project.
2. Open a diagram (see “Diagrams for deployment” on page 587).
3. Choose **Deploy | J2EE Deployment Expert** on the main menu.
4. On the first page of the Expert, the drop-down list of application servers shows the server platforms currently integrated. Generic 2.0 application server is set by default. Choose an application server to start from the available plug-ins:
 - Start BEA WebLogic Application Server 5.1
 - Start BEA WebLogic Application Server 6.0
 - Start BEA WebLogic Application Server 6.1
 - Start Sun EE

Using the J2EE Deployment Expert

Diagrams for deployment

You can invoke the J2EE Deployment Expert from diagrams of the following types:

- Class diagrams (JAR archive file is created).
- EJB Assembler diagrams (JAR archive file is created).
- Web Application diagrams (WAR archive file is created).
- Resource Adapter diagrams (RAR archive file is created).
- Enterprise Application diagrams (EAR archive file is created).

You can create one of these diagrams and then start the J2EE Deployment Expert.

You can find information about these diagrams in the following chapters:

- Class diagrams: “Class Diagrams” on page 154.
- EJB Assembler diagrams: Chapter 27, “Designing and Developing EJB Modules”.
- Web Application diagrams: Chapter 29, “Designing and Developing Web Applications”.
- Resource Adapter diagrams: Chapter 32, “Designing and Developing Resource Adapters” on page 569.
- Enterprise Application diagrams: Chapter 30, “Designing and Developing Enterprise Applications”.

If your application contains multiple EJB Assembler diagrams and Web Application diagrams, you can create an Enterprise Application diagram to map your existing modules.

Specifying parameters

The J2EE Deployment Expert provides a convenient GUI that greatly simplifies the J2EE deployment process. In this dialog, you can specify:

- The target application server platform.
- What deployment-related actions you want to take place (compile, and others).
- Paths to the server, server tools, and the deployment output.
- Connection parameters for the application server.
- Optional generation of a simple JSP client for live-testing the deployed EJB.
- Optional verification and correction, and so on.

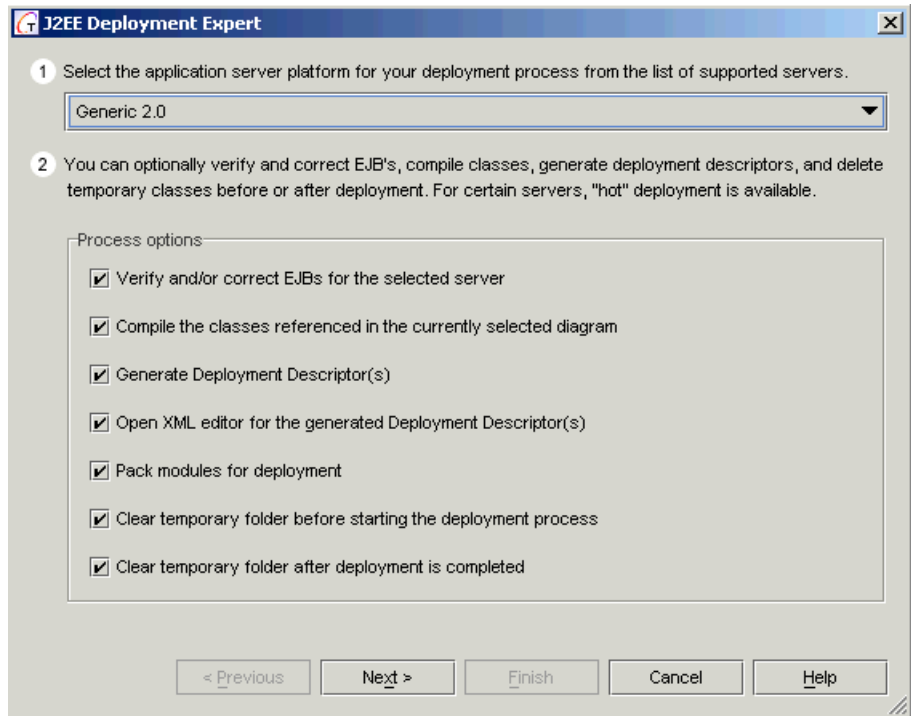
Running the J2EE Deployment Expert

To run the J2EE Deployment Expert:

1. Open a project.
2. Open a diagram (refer to the list of diagrams in “Diagrams for deployment” on page 587).
3. Choose **Deploy | J2EE Deployment Expert** on the main menu.
4. On the first page of the Expert, the drop-down list of application servers shows the server platforms currently integrated. Generic 2.0 application server is set by default. Select the application server that you are using.
5. Refer to the documentation for the selected server and become familiar with its requirements.

NOTE: Make sure all the necessary libraries are added. Together automatically adds the maximum number of available libraries in the plug-ins, but some servers require you to add paths. For example, according to SunEE requirements, the `client.jar` should be added to the project class path before running, otherwise an exception occurs. If the EJBs and other elements are deployed in different projects, the paths to these archive files should also be added to the project.

Figure 190



6. Set the other options as necessary. For example, to compile classes check the *Compile the classes* option; if they are already compiled, uncheck this option.

Other important options:

- Some servers offer you the generate a JSP client option. If desired, check *Generate a JSP client*. Note that this box does not appear for all servers. For more information, see “Server-specific information” on page 590.
- Check *Verify and/or correct EJBs* if you want to use verification and correction.
- Check *Generate deployment descriptor(s)* to generate them automatically.
- Check *Open XML editor* to view and edit *.xml files after deployment.

According to your option selections, Together handles interaction with the compiler to compile EJB or other classes, generates the XML deployment descriptors for the target application server, updates the EJB specification conformity, generate container classes, and then packages everything into a JAR (WAR, RAR, or EAR) file, and deploys to the location you specify. On server platforms that support it, you can specify “hot” deployment directly to the application server.

7. Click **Next** to advance through the page sequence of the Expert. Follow the J2EE Deployment Expert instructions on each page.

During the process you will specify the following:

- The paths to the server, various server tools, and a temp directory.

- The server host name, port number, and password (if you are going to “hot deploy” to the application server).
- The output location and base URL for JSPs (if generating a JSP test client).

8. On the last page, click **Finish** to start the deployment process.

NOTE: The number of pages and content of the expert varies according to the selected server platform because various servers have specific properties.

There are two main scenarios for using the J2EE Deployment Expert:

1. For *fast track* deployment with default values for security and permissions, or for deployment prototyping, run the J2EE Deployment Expert with an appropriate Class diagram current. Choose the target server platform and set the other options as desired.
2. If you need *full featured* assembly information with control over security and permissions, run the J2EE Deployment Expert from an EJB Assembler diagram, Web Application diagram, Resource Adapter diagram, or Enterprise Application diagram.

NOTE: The Web Application diagram and Enterprise Application diagram support Generic 1.1, Generic 2.0, IBM WebSphere Application Server 3.02/ 3.5/ 4.0, BEA WebLogic Application Server 5.1/ 6.0/ 6.1, iplanet 6.0, and SunEE 1.2/ 1.3, but do not support Generic 1.0.

The above procedure is intended as a basic guideline for running the expert. In order for the expert to succeed, you need to set up properties in the project and/or your environment. These are dependent on the target application server.

Server-specific information

The Development Expert requires users to select process options. Each page of the expert contains a number of tasks. After you have filled in all necessary fields and clicked Finish, the expert starts executing the selected tasks. This topic provides explanations of the available tasks. The tasks (with some exceptions) can be executed separately.

The pages of the J2EE Deployment Expert are dependent on the current application server. This chapter discusses only some of the most widespread and popular application server platforms supported by Together: IBM WebSphere 3.5 and 4.0, and BEA WebLogic 5.1.0, 6.0, and 6.1.

Deploying to IBM WebSphere Application Server

Together supports the IBM WebSphere 3.02, 3.5, and 4.0 Application Servers. Together supports both IBM WebSphere 4.0 Advanced Single Edition and IBM WebSphere 4.0 Advanced Edition.

This section describes the J2EE Deployment Expert for IBM WebSphere 4.0 and 3.5.

Setting project properties and environment, deployment steps

To set project properties and environment and then deploy:

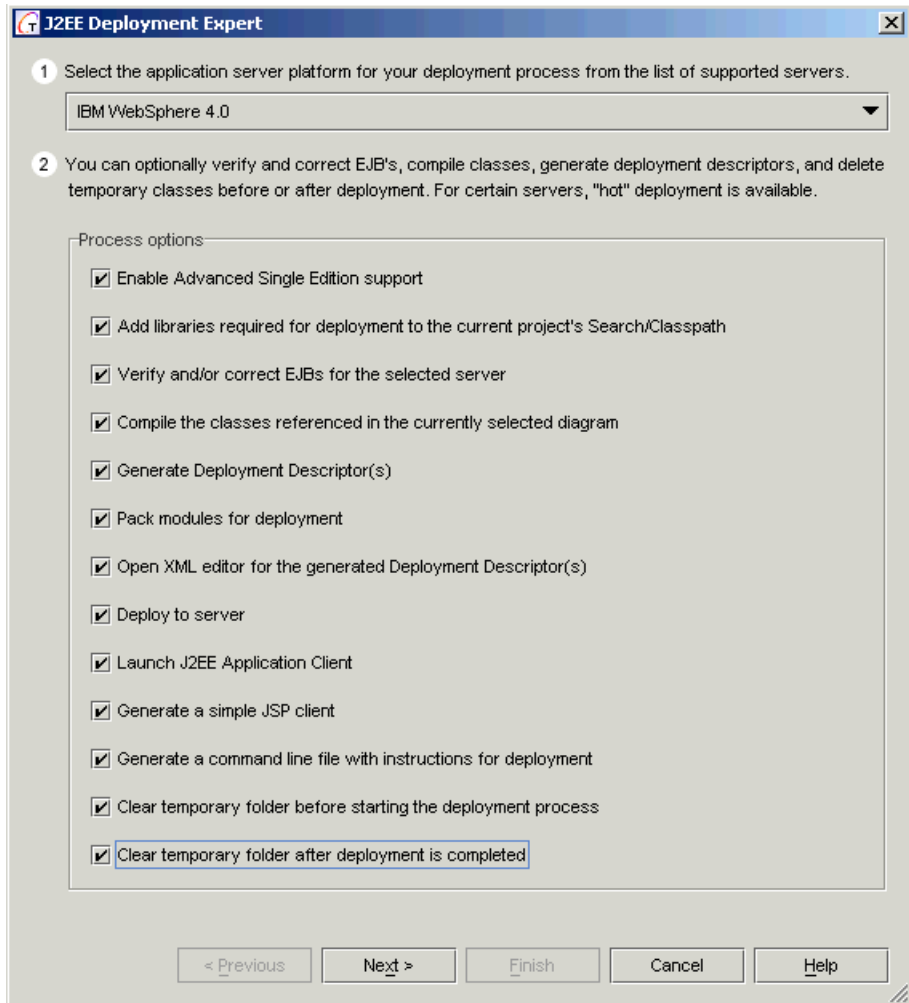
1. Open a diagram that can be used as the base diagram for deployment (Enterprise Application, EJB Assembler, Web Application, or Resource Adapter).
2. Choose **Deploy | J2EE Deployment Expert** on the main menu to start the J2EE Deployment Expert.
3. Select IBM WebSphere 4.0 (or AE 3.5, AE 3.02) from the drop-down list of application servers. The first page of the expert opens. You can use **Previous** and **Next** buttons to navigate through the expert's pages.
4. Fill in the process options on this page. Click **Next** to continue.
5. Set the common properties on the next page. Click **Next** to continue.
6. Set the deployment properties on the next page. Click **Next** to continue.
7. The following set of pages depends on the selected IBM WebSphere version. Click **Next** to continue when the current page is filled in. The optional page *Command Line File with Instructions for Deployment* is displayed if the corresponding box was checked on the first page.
8. Click **Finish** to start the deployment process.

Process options

Process options are similar for IBM WebSphere 4.0 and 3.5. The following section describes the process options for the first screen of the Deployment Expert for IBM WebSphere 4.0. Additionally, information for using the IBM WebSphere 3.5 version follows.

Figure 191

Enable Advanced Single Edition support



Check this box to use the Advanced Single Edition variant of IBM WebSphere 4.0. Do not check it if you are going to use the Advanced Edition version.

NOTE: This field is not used for IBM WebSphere 3.5 and others.

Add libraries required for deployment to the project's Search/Classpath

The J2EE deployment process requires a number of server-specific libraries. Because diverse servers use specific versions of the J2EE specification, they generally suggest their own libraries that cover the J2EE specification for the particular server usage. Therefore, a server-specific library is normally added to the Classpath instead of Sun J2EE. Check this box to compile and deploy the EJB or client program, and to make sure that you have all the required libraries in your Classpath.

Verify and/or correct EJBs for selected server

This task verifies that the classes on the current diagram correspond to the EJB specification supported by the destination server. Moreover, some servers have individual specification features. The primary objective of this task is to make sure that all the required classes exist on the diagram.

The current diagram is used as the source of beans, which is why you have to open the appropriate Class, EJB Assembler, Web Application, Resource Adapter, or Enterprise diagram.

Compile the classes referenced in the currently selected diagram

This task compiles the elements and packs them to a JAR, WAR, RAR, or EAR file. The new archive is stored in a temporary folder. Standard Sun tools (`javac.exe` and `jar.exe`) are used for compilation and packing.

Generate Deployment Descriptor(s)

Check this box to create deployment descriptors.

Open XML Editor for the generated Deployment Descriptor(s)

This option opens the XML Editor, where you can edit the generated deployment descriptor. Two additional descriptors are generated, one for each CMP entity bean. As soon as the descriptors are ready, they are added to the JAR archive together with some additional classes in a JAR file ready for deployment.

Pack modules for Deployment

Creates a deployable JAR, WAR, RAR, or EAR.

Deploy to server

If you check this box, deployment will execute immediately. WebSphere must already be started. This step makes use of a bean that has already been compiled and jarred.

NOTE: IBM WebSphere 4.0 Advanced Single Edition does not support “hot deployment.” You should restart the application server. IBM WebSphere 4.0 Advanced Edition and IBM WebSphere 3.5 support “hot deployment.” The name of this option is **Hot Deploy to server** for IBM WebSphere 3.5.

Generate executable files for the client *(for IBM WebSphere 3.5 only)*

Together and WebSphere use different versions of the JDK. WebSphere requires using the IBM JDK for client programs. For this reason, you cannot make use of Together internal tools. You have to generate two batch files; designate one to compile and the other one to run the client program. Subsequently, you can start them separately from Together.

Process Servlet(s) *(for IBM WebSphere 3.5 only)*

Check this box to deploy Servlets.

Launch J2EE Application Client *(for IBM WebSphere 4.0 only)*

You might have used an Application Client diagram for creating your application client. To deploy it to the current application server, you have to place the shortcut for this Application Client diagram on the current Enterprise Application diagram and then deploy.

If you check this box, an application client that was already deployed will be invoked automatically.

Generate a simple JSP client

Check this box to generate a JSP client for the Enterprise Beans. The client is a set of interrelated JSP and HTML files, which can be viewed in any Internet browser. The intention of this client is to demonstrate access to remote a EJB objects through an open interface. The semantic information required for a JSP client generation is taken from the active diagram of the current project. Due to the lack of information, it is impossible to create a client program ready for commercial use. For this reason, regard a JSP client as a universal testing facility.

It is possible to generate a JSP client from Web Application and Enterprise Application diagrams.

WARNING! At the present moment a generated JSP has some restrictions :

- JSP Client does not support EJBs that do not have remote or home interfaces.
- Local home and local remote interfaces are ignored. They cannot be accessed from a JSP Client.
- JSP Client does not support message- driven beans.

Generate a command line file with instructions for deployment

Check this box to generate a command file. The deployment process involves a sequence of calls to various console tools. This sequence can be generated and stored in a command file. Then this file can be launched to perform all actions required for deployment.

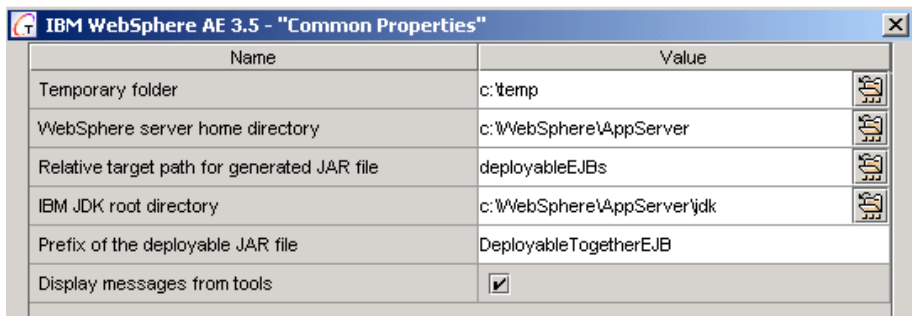
Clear temporary folder before starting the deployment process / Clear temporary folder after deployment is complete

Uncheck these options if you want to reuse temporary files.

Deployment steps for IBM WebSphere 3.5

Common Properties page

Figure 192



The fields on this page are self-explanatory. However, this section provides brief descriptions for some of them.

Relative target path for generated JAR file

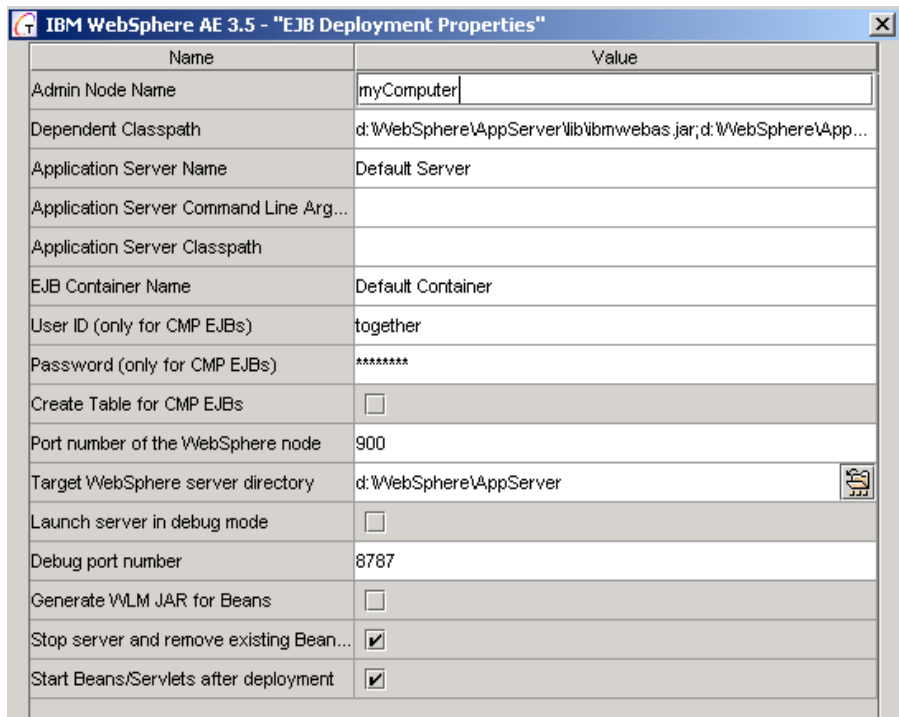
Specify the location where the resulting JAR file will be stored and from where it will be loaded by WebSphere.

Prefix of deployable JAR file

This name is assigned to the JAR file (or files) generated in the Compile Classes task from the selected diagram. While preparing for deployment, an additional JAR file is created, whose name is based on the prefix assigned here.

EJB Deployment Properties page

Figure 193



Name	Value
Admin Node Name	myComputer
Dependent Classpath	d:\WebSphere\AppServer\lib\ibmwebas.jar;d:\WebSphere\App...
Application Server Name	Default Server
Application Server Command Line Arg...	
Application Server Classpath	
EJB Container Name	Default Container
User ID (only for CMP EJBs)	together
Password (only for CMP EJBs)	*****
Create Table for CMP EJBs	☐
Port number of the WebSphere node	900
Target WebSphere server directory	d:\WebSphere\AppServer
Launch server in debug mode	☐
Debug port number	8787
Generate WLM JAR for Beans	☐
Stop server and remove existing Bean...	☒
Start Beans/Servlets after deployment	☒

Admin Node Name

By default, this name corresponds to the local computer name and is assigned automatically.

Dependent Classpath

This field should contain the path to source classes, service library jars, and so on.

Application Server Name

A single WebSphere node can hold several virtual application servers. By default, only one is allocated. Its name is Default Server. The J2EE Deployment Expert uses the same name by default. If you use a different application server, specify its name in this field. If the server with the specified name does not exist, it is created.

Application Server Command Line Arguments / Application Server Classpath

Contents of these fields should correspond to the appropriate properties on the specific application server's Administrative Console.

EJB Container Name

A single WebSphere virtual Application Server can hold several EJB containers. By default, only one is allocated. Its name is Default Container. The J2EE Deployment Expert uses the same name by default. If you use a different container, specify its name in this field. If the container with the specified name does not exist, it is created.

User ID / Password (only for CMP beans)

These fields are necessary to maintain container-managed persistence. You can use the user name and password assigned during WebSphere installation.

Create Table for CMP EJBs

When checked, this task creates new tables in the server persistence repository. These tables are destined to support Container Managed Persistence. The structure of the tables corresponds to the beans being deployed. This task pertains to CMP beans only.

Port number of the WebSphere node

This port is used for communication between the application server and client program. It is assumed that the host is localhost. Before proceeding with the next page of the J2EE Deployment Expert, make sure that the server is available for the selected port and host.

Target WebSphere server directory

Specify the path to the WebSphere application server product installation root.

Launch server in debug mode

If you are going to debug your beans, you can run the server in debug mode and attach to it from the debugger process in Together. Actually, this option does not restart the server in debug mode, but adds special command-line arguments to the application. You have to restart the application server from the WebSphere Administrative Console.

NOTE: A number of additional libraries are required to launch WebSphere in debug mode. See WebSphere InfoCenter documentation for details.

Debug port number

Enter the port number for communication between WebSphere and the external debugger. The server and debugger must use the same port number. (Good advice: keep this number in a safe place.) You can use the default port number 8787.

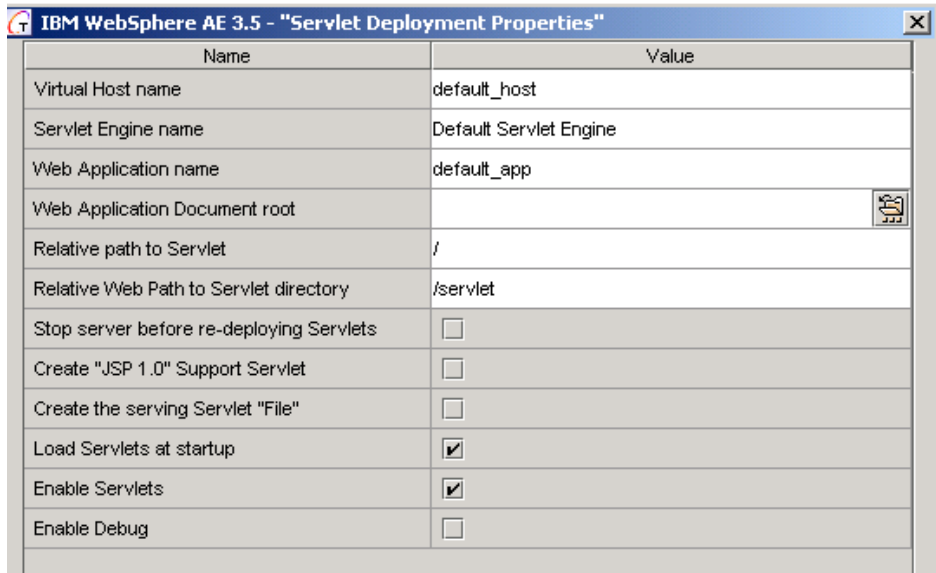
Generate WLM Jar for Beans

Check this box to create a Work Load Management file.

Servlet Deployment Properties page

You can view this page only if *Process Servlet(s)* is checked. You can use this page for creating servlets and WebSphere service servlets.

Figure 194



Name	Value
Virtual Host name	default_host
Servlet Engine name	Default Servlet Engine
Web Application name	default_app
Web Application Document root	
Relative path to Servlet	/
Relative Web Path to Servlet directory	/servlet
Stop server before re-deploying Servlets	☐
Create "JSP 1.0" Support Servlet	☐
Create the serving Servlet "File"	☐
Load Servlets at startup	☒
Enable Servlets	☒
Enable Debug	☐

In order to access any WEB resource, you have to specify the URL. A standard Web URL includes protocol, host, port, web resource alias, and path. The path to a resource is implied relative to the document root. It consists of directories and a file name at the end.

Virtual Host name

A single WebSphere Application Server can hold and serve several virtual Web hosts. Each virtual host is associated with a set of aliases. By default, only one host is allocated. Its name is `default_host`. It is associated with the `localhost`. The `default_host` is used in the J2EE Deployment Expert by default. If you want to use a different virtual host, specify its correct name here.

Servlet Engine name

Each WebSphere virtual Application Server can hold several Servlet Engines. By default, only one engine is allocated. Its name is `Default Servlet Engine`. The same name is used in the J2EE Deployment Expert by default.

If you use a different Servlet Engine, specify its name in this field. If the engine with the specified name does not exist, it is created.

Web Application name

Each WebSphere Servlet Engine can hold several Web Applications. After the server installation there are several Web application tasks. J2EE Deployment Expert uses the default_app Web application. If you are going to use a different Web application, specify its name in this field. If the application with the specified name does not exist, it is created.

Web Application Document root

Each Web application on the WebSphere server has its own Document root. Document Root is a file system address accessible through an Internet browser. All subdirectories of the root are also accessible. Contents of this field should be equal to the corresponding property of the Web Application in the WebSphere Advanced Administrative Console.

Relative path to Servlet

Each Web Application on the WebSphere server has its own Web Path. This is a string similar to a directory path, which however does not refer to any actual directory. This is a Web resource alias for the Web application. This Web path is appended to the virtual host alias. For example, if the virtual host is localhost:900 and the Web path is /webapp/Examples, then the result will be http://localhost:900/webapp/Examples. If *Web Application* is empty, the Web path consists of a single slash character (/).

Relative Web Path to Servlet directory

In this field you have to specify the part of Web path to access a servlet. This path is a string similar to the directory path, which however does not refer to any actual directory. It is appended to the Relative path to Servlet (see previous paragraph). For example, if the virtual host is localhost:900, Web path is /webapp/Examples and the relative Web path to a servlet is /servlet, then the result will be http://localhost:900/webapp/Examples/servlet.

If *Relative Web Path to Servlet directory* is empty, it consists of a single slash character (/).

Stop Server before re-deploying Servlets

Stopping or restarting a running servlet requires Application Server Node shutdown. However, this results in the loss of all deployed objects, including the EJB's. Check this box if the running beans are of no interest, and restart the servlets together with the Application Server. If you preserve the running beans, the new servlets still deploy, but they do not stop or start, regardless of the previously set options *Stop (remove) EJB's, Servlets*, and *Start EJB's, Servlets* on the page *EJB Deployment Properties*.

Create “JSP 1.0” Support Servlet

Check this box if you are going to use JSP technology in your project. This flag indicates whether a special servlet will be added to the Web application. The name of this servlet class is `com.sun.jsp.runtime.JspServlet`. In WebSphere it is normally called `jsp10`. This servlet can be added manually or by checking the box.

Create the serving Servlet “File”

Check this box if you are going to use HTML resources in your project. This flag indicates whether a special servlet, responsible for processing requests to HTML resources, will be added to the Web application. The name of this servlet class is `com.ibm.servlet.engine.webapp.SimpleFileServlet`. In WebSphere it is normally called `file`. This servlet can be added manually or by checking the box.

Load Servlets at Startup

This option indicates whether to load a servlet when the application server starts or wait until the servlet is requested. By default it is set to `False`.

Enable Servlets

This option indicates whether the servlet is available to handle requests. By default it is set to `True`.

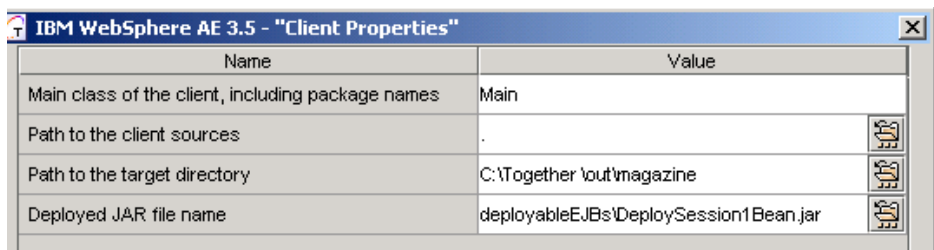
Enable Debug

This option indicates whether to start the servlet in Java debug mode. By default it is set to `False`.

Client Properties page

The *Client Properties* page contains entries for the properties required to generate start and compile files for the client task (if the appropriate box is checked on the first page of the expert).

Figure 195



Main class of the client, including package names

Enter the class name that contains the main method to run the client application. Automatically generated batch files will refer to this class.

Path to the client sources

Enter the fully qualified path to the directory where client source files are stored.

Path to the target directory

Enter the fully qualified path to the directory where compiled files will be placed.

Simple JSP Client Generation page

The *Simple JSP Client Generation* page is displayed if you have checked the appropriate box on the first page of the Expert. It contains entries for the properties required to generate a Simple JSP client.

Figure 196

Name	Value
Root path for JSP storage	d:\WebSphere\AppServer\hosts\default...
Subdirectory for JSP files	myJSP
URL for browsing JSP	http://localhost/webapp/examples
WebSphere JNDI service provider URL	iiop://localhost:900
Show JSP client in default Internet browser	☒

The fields are self-explanatory.

Root path for JSP storage

Specify the root of the file hierarchy, which can be accessed externally. To allow access to some HTML or JSP files, you have to place these files under the document root.

Subdirectory for JSP files

To avoid disorder in the public directory, it is advisable to group the files in a subdirectory. Enter the path to the subdirectory where your files will be stored. This path is relative to the root public directory. If the subdirectory does not exist, it is created automatically (after confirmation).

URL for browsing JSP

In order to browse a remote WEB resource, we have to set its URL. A standard Web URL includes protocol, host, port, web resource alias, and path. Specify all parts of the URL, except for the path. During request processing, the server substitutes this base URL to the WebSphere public directory in order to access the requested Web resource. An invalid value in this field or the WebSphere public directory causes access error. By default, the base URL is `http://localhost`.

WebSphere JNDI service provider URL

This field is required to establish connection between a JSP client and an EJB server. Usually, the client first sends a request to the name service provider in order to get access to an instance of an EJB. The value entered in this field is inserted in a JSP file during generation and is subsequently used during execution. Normally, you do not need to change this field. The default value is `iiop://localhost:900`.

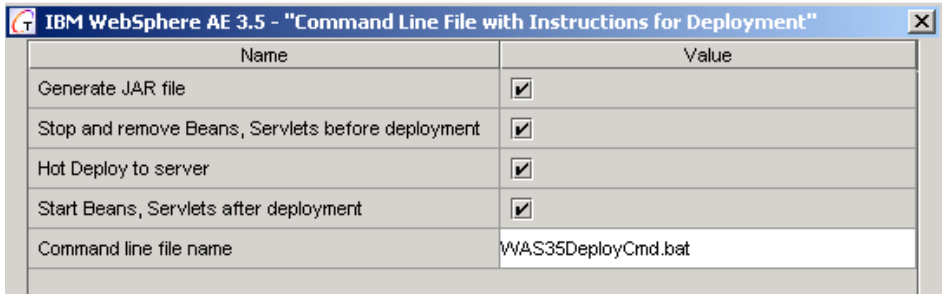
Show JSP client in default Internet browser

Check this box to launch your default Internet browser and display the start page.

Command Line File with Instructions for Deployment page

This page contains options for creating the command line file.

Figure 197



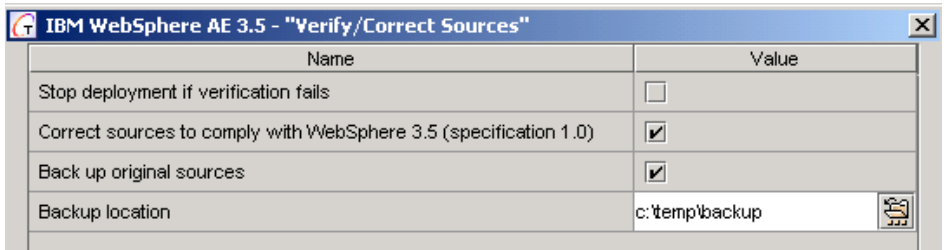
Name	Value
Generate JAR file	☒
Stop and remove Beans, Servlets before deployment	☒
Hot Deploy to server	☒
Start Beans, Servlets after deployment	☒
Command line file name	WAS35DeployCmd.bat

The fields are self-explanatory. They enable you to define the command line file's contents and properties.

Verify/Correct Sources page

The *Verify/Correct Sources* page contains entries for the Verify/Correct task. The fields are self-explanatory. They enable you to terminate deployment if verification fails, correct sources to make them compliant with WAS 3.5, make backup copies, and change return types in the created class.

Figure 198



Name	Value
Stop deployment if verification fails	☐
Correct sources to comply with WebSphere 3.5 (specification 1.0)	☒
Back up original sources	☒
Backup location	c:\temp\backup 

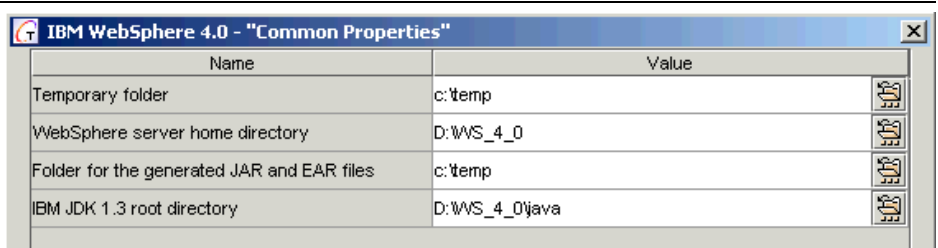
Correct sources to comply with WebSphere 3.5 (specification 1.0)

IBM WebSphere Application Server version 3.5 supports EJB specification 1.0. This is not the latest EJB specification. Later specifications require that create methods of a home interface should return the primary key type, while the 1.0 specification requires that create methods return void. For the sake of compliance with EJB 1.0 specification, the correction program replaces all changed types with void.

Deployment steps for IBM WebSphere 4.0

Common Properties page

Figure 199



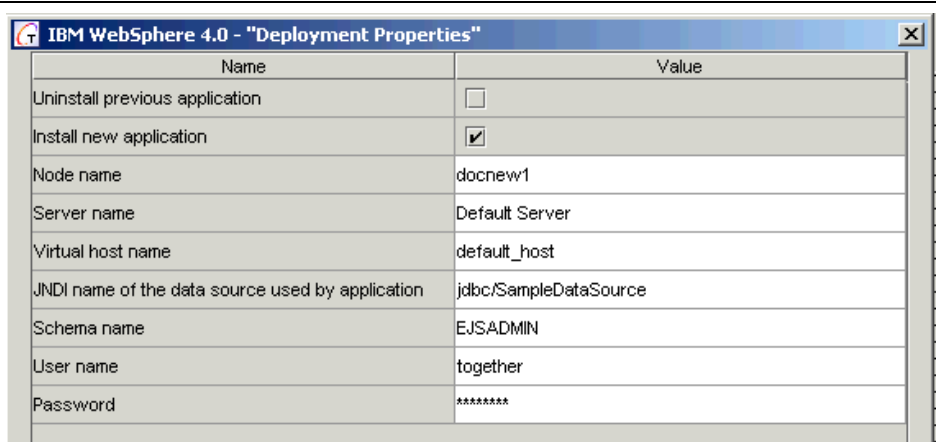
Name	Value
Temporary folder	c:\temp
WebSphere server home directory	D:\WS_4_0
Folder for the generated JAR and EAR files	c:\temp
IBM JDK 1.3 root directory	D:\WS_4_0\java

The fields on this page are self-explanatory.

Deployment Properties page

This page also varies for different versions of IBM WebSphere.

Figure 200



Name	Value
Uninstall previous application	☐
Install new application	☒
Node name	docnew1
Server name	Default Server
Virtual host name	default_host
JNDI name of the data source used by application	jdbc/SampleDataSource
Schema name	EJSADMIN
User name	together
Password	*****

The *Deployment Properties* page provides entries for the “hot deploy” task and some other properties.

Uninstall previous application

Check this box if there is an older deployed application with the same name on the current application server.

Install new application

Check this box if you are going to deploy a new application.

Node name

By default, this name corresponds to the local computer name and assigns automatically.

Server name

This is the name of the current application server in the IBM WebSphere Administrative Console.

Virtual host name

This is the name of the host in the IBM WebSphere Administrative Console.

JNDI name of the data source / Schema name

These fields are used for CMP beans only. JNDI name is the name used by the data source, and the Schema name is the name connected to your database. To fill in these fields you should start the IBM WebSphere Administrative Console and set the same settings as you have there.

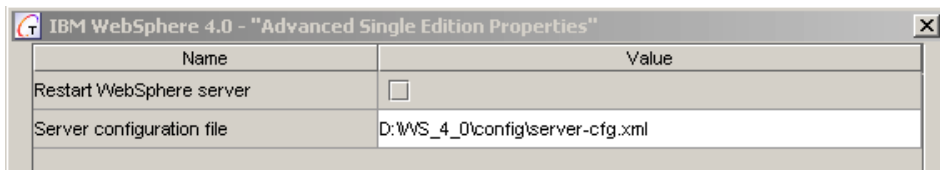
User name and Password (for CMP EJBs)

The name and password that were defined during IBM WebSphere installation.

Advanced Single Edition Properties page

The next page is shown only for IBM WebSphere 4.0 Advanced Single Edition. You can view this page only if you checked the *Advanced Single Edition support* option on the first page of the expert.

Figure 201



Name	Value
Restart WebSphere server	☐
Server configuration file	D:\WS_4_0\config\server-cfg.xml

IBM WebSphere 4.0 Advanced Single Edition supports the “hot deploy” process, but IBM WebSphere 4.0 Advanced Single Edition does not support “hot” deployment. You should restart the current application server for deployment.

WARNING! IBM WebSphere 4.0 Advanced Single Edition application server does not verify the correctness of JNDI names for data sources, and relationships with other beans. For this reason, a JAR file might contain incorrectly defined JNDI names.

Restart WebSphere server

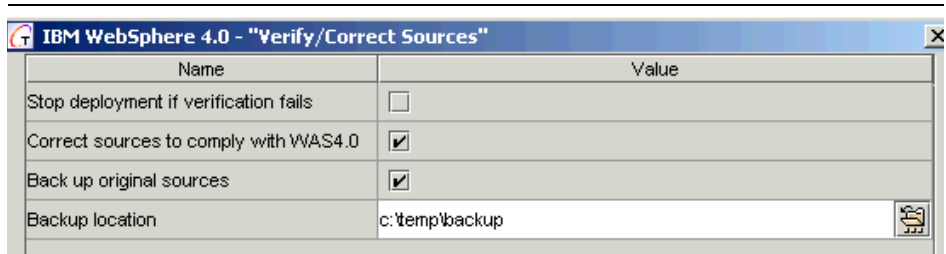
If you check this box, IBM WebSphere 4.0 Advanced Single Edition will be restarted.

Server configuration file

Information about EJBs is saved in the corresponding server configuration file `server-cfg.xml`. Set the path to this configuration file.

Verify/Correct Sources page

Figure 202



Name	Value
Stop deployment if verification fails	☐
Correct sources to comply with WAS4.0	☒
Back up original sources	☒
Backup location	c:\temp\backup

Use these fields to provide verification or correction of the source Java code. For example, IBM WebSphere 3.5 supports the EJB 1.0 specification, but IBM WebSphere 4.0 supports the EJB 1.1 specification.

Stop deployment if verification fails

Check this box to stop deployment if the result of verification is negative.

Correct sources to comply with WAS 4.0

Check this box if you want to correct the corresponding Java code.

Back up original sources

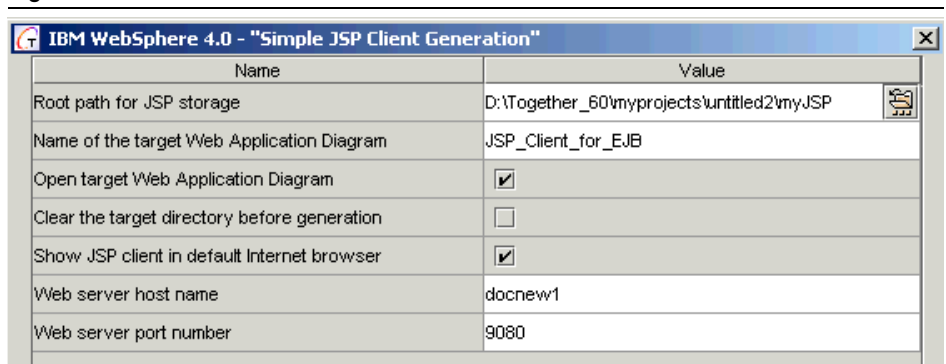
Check this box if you want to save a backup file.

Backup location

Define the directory where you will back up your original sources.

Simple JSP Client Generation page

Figure 203

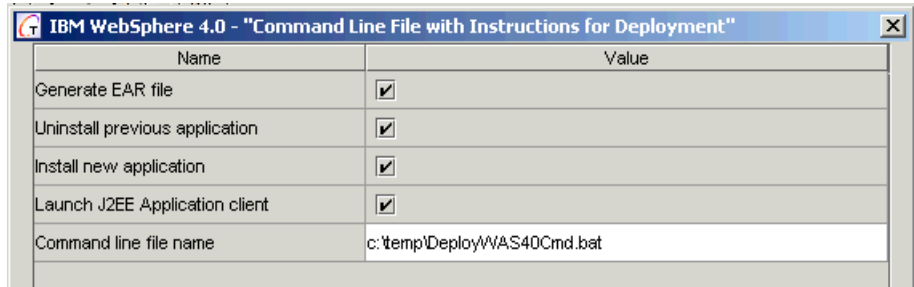


Name	Value
Root path for JSP storage	D:\Together_60\myprojects\untitled2\my.JSP
Name of the target Web Application Diagram	JSP_Client_for_EJB
Open target Web Application Diagram	☒
Clear the target directory before generation	☐
Show JSP client in default Internet browser	☒
Web server host name	docnew1
Web server port number	9080

This page displays only if you checked the box *Generate a simple JSP client* on the first page of the expert. Most of these fields are self-explanatory; other fields are the same as for IBM WebSphere 3.5.

Command Line File with Instructions for Deployment page

Figure 204



Name	Value
Generate EAR file	☒
Uninstall previous application	☒
Install new application	☒
Launch J2EE Application client	☒
Command line file name	c:\temp\Deploy\WAS40Cmd.bat

All properties on this page are self-explanatory. Check these options to generate an EAR file, uninstall an existing application with the same name, install a new application, or invoke the J2EE application client. Set the full path to the command file.

When you have made all your settings, click Finish to deploy your application.

Deploying to BEA WebLogic Application Server

Together supports BEA WebLogic 4.5.1, 5.1, 6.0, and 6.1 Application Servers. This section describes the J2EE Expert for BEA WebLogic 6.1 Application Server, with notes regarding variations for other versions of BEA WebLogic Application Server.

This section describes how to set up your Together project and your environment so that you can use Together to compile both EJBs and clients and deploy them to BEA WebLogic 5.1.0, 6.0, or 6.1.

Starting BEA WebLogic Application Server from within Together

When deploying an existing enterprise or Web application to BEA WebLogic, you can first start the chosen application server (this is referred to as “hot” deployment). You can do this using the BEA WebLogic *Start Default Server* option, or the BEA WebLogic starting command file.

BEA WebLogic 5.1, 6.0, and 6.1 Application servers can be started from the Together Server Explorer.

To start BEA WebLogic 5.1, 6.0, or 6.1 Application Servers:

1. Go to the Server Explorer tab in the Explorer pane. Navigate to the necessary node under *Application Servers*, such as *BEA WebLogic 6.1- default* (or another application server with existing settings).

2. Right-click on the server name and choose **Properties** on the right-click menu.
 3. In the Server Properties dialog, make the necessary settings. Click the file chooser button to set the root directory of the BEA WebLogic server, and click **Ok**.
 4. Choose **Edit Start Command File** on the Server Explorer right-click menu to invoke the editor. For BEA WebLogic 6.x, you should make the following changes in the command file `startWebLogic.cmd`:
 - 1) In the beginning of this file after the `SETLOCAL` program line, and before the `cd ..\..\..` program line, add the line:


```
cd D:\bea\wlserver6.1\config\mydomain\
```

 where `D:\bea\wlserver6.1` is the full path to the directory where BEA WebLogic 6.x is installed.

The result should look like this:

```
SETLOCAL
cd D:\bea\wlserver6.1\config\mydomain\
cd ..\..\..
```
 - 2) Before the program line `set STARTMODE=true`, and set `WLS_PW` the same as your BEA WebLogic password, so that you will not be prompted for the password during server startup. The result should look like this:


```
set WLS_PW=together
set STARTMODE=true
```
 5. Right-click on the chosen application server node in the Server Explorer and choose **Start** from the right-click menu.
 6. Your application server starts, and then you can see the BEA WebLogic Console.
- Alternatively, to start BEA WebLogic Application server:*
1. Open a project.
 2. Open the current diagram for deployment.
 3. Choose **Deploy | J2EE Deployment Expert** on the main menu.
 4. On the first page of the Expert, choose one of the plug-ins from the drop-down list of application servers.
 - Start BEA WebLogic Application Server 5.1
 - Start BEA WebLogic Application Server 6.0
 - Start BEA WebLogic Application Server 6.1

Setting project properties and environment, deployment steps

To set project properties and environment, and then deploy:

1. Open a diagram that can be used as the base diagram for deployment (Enterprise Application, EJB Assembler, Web Application, or Resource Adapter).
2. Choose **Deploy | J2EE Deployment Expert** on the main menu to start the J2EE Deployment Expert.

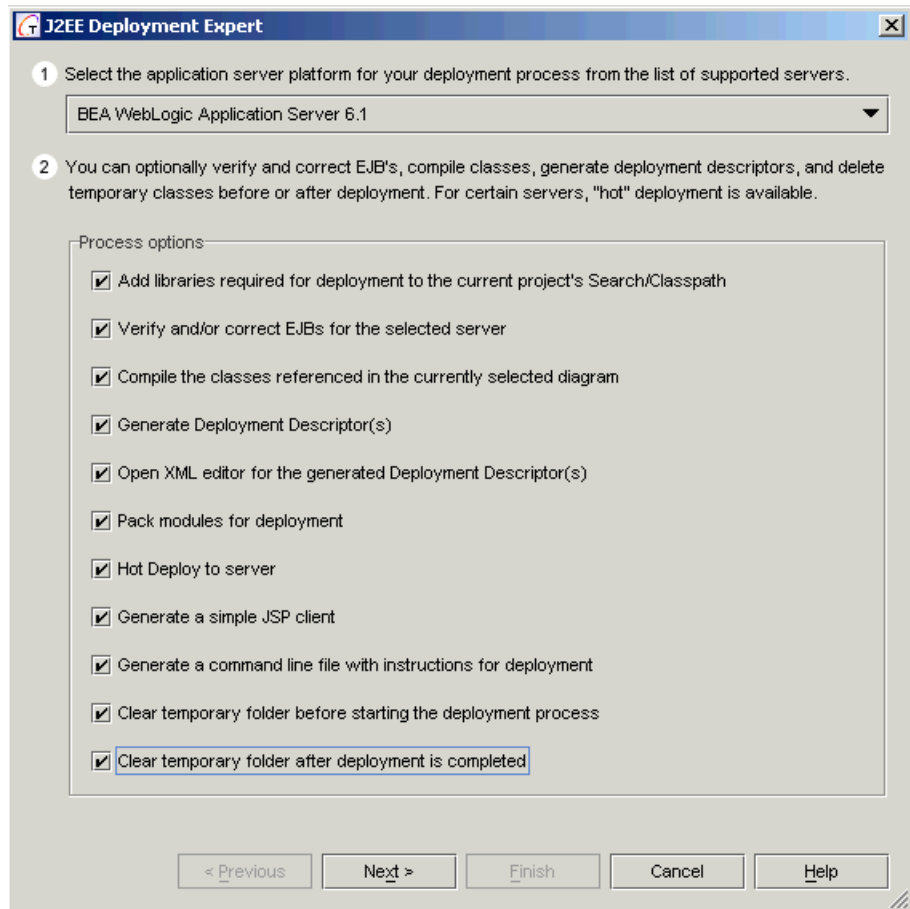
3. Select *BEA WebLogic 6.1* (or 6.0, 5.1, 4.5.1) from the drop-down list of application servers. The first page of the expert opens. You can use **Previous** and **Next** buttons to navigate through the pages of the expert.
4. Fill in the Process options on this page. Click **Next** to continue.
5. Set the common properties on the next page. Click **Next** to continue.
6. If you use verification during deployment (if you checked the box *Verify and/or correct EJBs for the selected server* on the first page), the optional *Verify/Correct Sources* page is displayed. Set parameters on this page. Click **Next** to continue.
7. Set the server-specific properties on the *Run-time deploy* page. Click **Finish** if this page is the last one. Otherwise, click **Next** to continue.
8. If you are deploying an enterprise application from an Enterprise Application diagram, and you checked the box *Generate a simple JSP client* on the first page of the expert, the optional *Simple JSP Client Generation page* is displayed. Set parameters on this page. Click **Next** to continue.
9. The optional *Command Line File with Instructions for Deployment* page is displayed if you checked the corresponding box on the first page.
10. Click **Finish** to start the deployment process.

The J2EE Deployment Expert for BEA WebLogic Application Server

Process options

When you choose one of the BEA WebLogic Servers as the current application server, you can see the first page of the J2EE Deployment Expert as shown in .

Figure 205



NOTE: The following options are available when choosing the BEA WebLogic 4.5.1 application server:

- Verify and/or correct EJBs for the selected server
- Compile the classes referenced in the currently selected server
- Generate Deployment Descriptor(s)
- Hot Deploy to server
- Clear temporary folders before starting the deployment process
- Clear temporary folders before/after deployment is completed

BEA WebLogic 4.5.1 does not offer some of the possibilities that are available for later servers.

Add libraries required for deployment to the current project's Search/Classpath

EJB deployment requires a number of server-specific libraries. Because diverse servers use specific versions of the J2EE specification, they generally suggest their own libraries that cover J2EE specification for the particular server usage. Therefore, a server-specific library is normally added to the Classpath instead of the Sun J2EE. Check this box to compile and deploy the EJB or client program and to verify that you have all the required libraries on your Classpath.

NOTE: If all the necessary libraries have already been added to the classpath, it is recommended to uncheck the *Add libraries* box to avoid problems with the compiler locking JAR files (for information about this problem, refer to the `readme.html` file in the root of your Together installation).

Verify and/or correct EJBs for the selected server

This task verifies that the classes on the current diagram correspond to the EJB specification supported by the destination server. Moreover, some servers have individual specification features. The primary objective of this task is to make sure that all the required classes exist on the diagram.

NOTE: The current diagram is used as the source of beans. For this reason, you must open the appropriate Class or EJB Assembler diagram.

Compile the classes referenced in the currently selected diagram

Check this box to compile the bean and pack it in a JAR file. The new archive is stored in a temporary folder. The standard Sun tools (`javac.exe` and `jar.exe`) are used for compilation and packing.

Generate Deployment Descriptor(s)

Check this box to generate deployment descriptors. One descriptor is generated for each CMP entity bean. As soon as the descriptors are ready, they are added to the JAR, WAR, RAR, or EAR archive together with some additional classes. The BEA WebLogic tool `ejbc.exe` is responsible for these operations, which result in an archive file ready for deployment.

Open XML Editor for the generated Deployment Descriptor(s)

Check this box to open the XML Editor and to edit the generated Deployment Descriptor.

Pack Modules for Deployment

Check this box to create deployable JAR, WAR, RAR, or EAR archives.

Hot Deploy to server

Check this box to perform deployment. This step makes use of a bean already compiled and jarred, so you have to use this step in conjunction with the others. BEA WebLogic Application Server must already be started.

WARNING! Be aware that all hot deployed beans will be lost after BEA WebLogic termination. You will have to re-deploy any necessary beans again. In order to deploy your beans permanently, you have to add certain lines manually to the file `weblogic.property` for BEA WebLogic 5.1, or to `config.xml` for BEA WebLogic 6.0 and 6.1.

NOTE: As a rule, it is strongly advised to compile classes, generate descriptors, and deploy the beans all together, because some tasks use the results of the previous tasks. For example, the Hot Deploy task requires a *.jar file to be prepared in a temporary folder.

Generate a simple JSP client

Check this box to generate a JSP client for the Enterprise Beans. The client is a set of interrelated JSP and HTML files, which can be viewed in any Internet browser. The intention of this client is to demonstrate access to remote EJB objects through the opened interfaces. The semantic information required for JSP client generation is taken from the active diagram of the current project. Due to the lack of information, it is impossible to create a client program ready for commercial use. For this reason, regard a JSP client as a universal testing facility.

It is possible to generate a JSP client from Web Application and Enterprise Application diagrams.

WARNING! “At the present moment a generated JSP has some restrictions :” on page 594

Generate a command line file with instructions for deployment

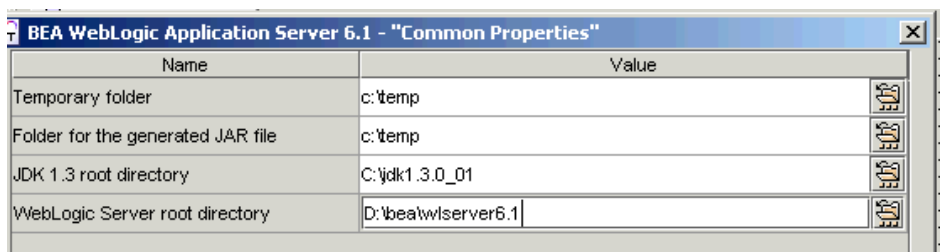
Generates a command file for deployment.

Clear temporary folder before starting the deployment process / Clear temporary folder after deployment is complete

Uncheck these options if you want to reuse temporary files.

Common Properties page

Figure 206



Name	Value
Temporary folder	c:\temp
Folder for the generated JAR file	c:\temp
JDK 1.3 root directory	C:\jdk1.3.0_01
WebLogic Server root directory	D:\bea\wlserver6.1

The fields on this page are self-explanatory. However, this section provides brief descriptions for some of the fields.

Folder for the generated JAR file

Specify the location where the resulting JAR file will be stored and from where it will be loaded by WebLogic.

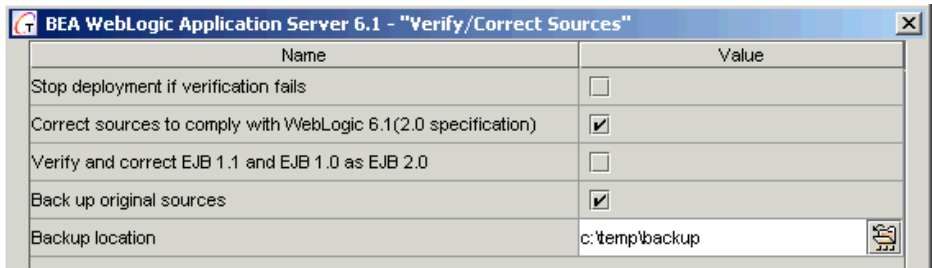
JDK 1.3 root directory

Specify the JDK location.

WARNING! Set paths only to jdk1.2 or later for BEA WebLogic 5.1, and to jdk1.3 for BEA WebLogic 6.0 and 6.1. You cannot use jdk1.2 for BEA WebLogic 6.0 and 6.1.

Verify/Correct Sources page

Figure 207



The dialog box titled "BEA WebLogic Application Server 6.1 - 'Verify/Correct Sources'" contains a table with the following data:

Name	Value
Stop deployment if verification fails	☐
Correct sources to comply with WebLogic 6.1(2.0 specification)	☒
Verify and correct EJB 1.1 and EJB 1.0 as EJB 2.0	☐
Back up original sources	☒
Backup location	c:\temp\backup

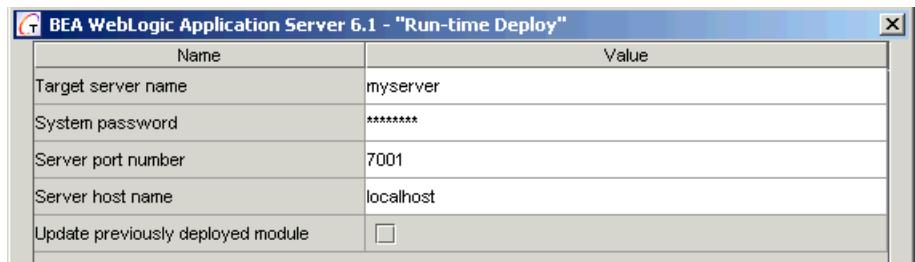
On the *Verify/Correct sources* page, you can select the option to stop deployment if verification fails, correct the source code to comply with the requirements of the corresponding specification, and create a backup copy of the source code.

NOTE: If you have both EJBs satisfying the EJB1.1 specification and EJBs satisfying the EJB 2.0 specification in the same project, all EJBs are deployed to the application server without any correction of the code.

Run-time Deploy page

This page enables you to assign properties required for the “hot deployment” task.

Figure 208



The dialog box titled "BEA WebLogic Application Server 6.1 - 'Run-time Deploy'" contains a table with the following data:

Name	Value
Target server name	myserver
System password	*****
Server port number	7001
Server host name	localhost
Update previously deployed module	☐

System password is defined during BEA WebLogic installation.

Server port number

The default server port is 7.3X. If you do not know the server port number, you can find it in the file:

- `weblogic.property` for BEA WebLogic 5.1.0.
- `config.xml` for BEA WebLogic 6.0 and 6.1.

Server host name

As a rule, the server host is assigned to `localhost`. Because this deployment expert uses the file system access to BEA WebLogic server, the server runs on the same computer.

Update already deployed module

During run-time deployment to BEA WebLogic x.x, it is possible to update any modules that have already been deployed. Check this box to update modules during deployment.

NOTE: This functionality is not available for the other application servers.

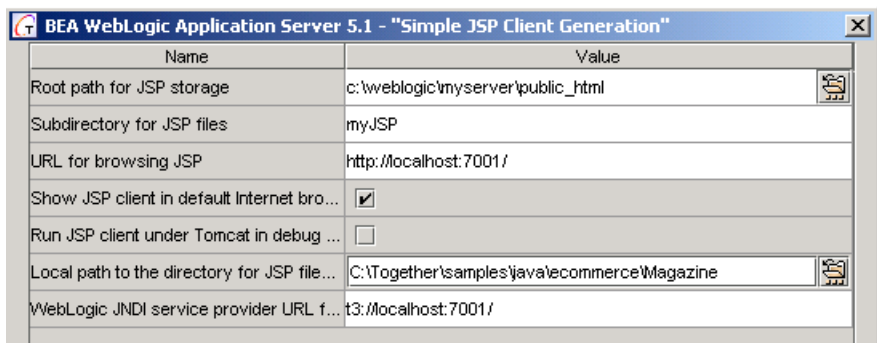
Before proceeding to the next page, make sure that the server is available at the selected port and host and that the current application server has been started.

Simple JSP Client Generation page

This page displays if you checked the appropriate box on the first page of the expert. It contains entries for the properties required to generate a simple JSP client. This page varies depending on the WebLogic version.

For BEA WebLogic 5.1

Figure 209



Name	Value
Root path for JSP storage	c:\weblogic\myserver\public_html
Subdirectory for JSP files	myJSP
URL for browsing JSP	http://localhost:7001/
Show JSP client in default Internet bro...	☒
Run JSP client under Tomcat in debug ...	☐
Local path to the directory for JSP file...	C:\Together\samples\java\ecommerce\Magazine
WebLogic JNDI service provider URL f...	t3://localhost:7001/

Root path for JSP storage

This directory is the root of the file hierarchy, which can be accessed externally. If you want to allow access to some HTML or JSP files, you have to place these files under the

document root. By default, the WebLogic public directory resides at [%WebLogic home]/myserver/public_html

Subdirectory for JSP files

To avoid disorder in the public directory, it is advisable to group the files in a subdirectory. Enter the path to the subdirectory where your files will be stored. This path is relative to the root public directory. If the subdirectory does not exist, it is created automatically (after confirmation).

URL for browsing JSP

In order to browse a remote WEB resource, a URL must be set. A standard Web URL includes protocol, host, port, web resource alias, and path. Specify all parts of the URL, except for the path. During request processing, the server substitutes this base URL to the BEA WebLogic public directory in order to access the requested Web resource. An invalid value in this field or the WebLogic public directory causes an access error. By default, the base URL is `http://localhost:7001`.

Show JSP client in default Internet browser

Check this box to launch your default Internet browser and display the start page.

Run JSP client under Tomcat in debug mode

If this box is checked, the generated JSP client will run under Tomcat, rather than BEA WebLogic server. In this case, JSPs interact with an EJB remotely. For this purpose, the source code of these JSPs contains lines for initialization of the Initializing object. The initialization text is stored in a configuration file, and can be modified if necessary.

Local path to the directory for JSP files for Tomcat

This field defines allocation of the generated JSP files for running under Tomcat. The folder specified in this field is alternative to the one in the WebLogic public directory. By default, the root directory of the current project is assumed.

WebLogic JNDI service provider URL for Tomcat

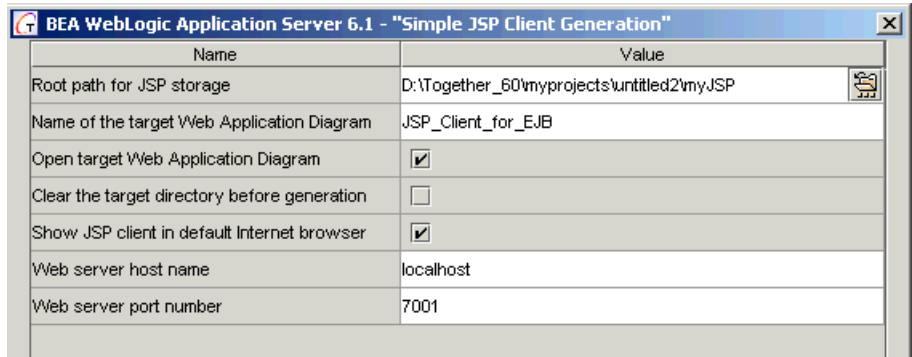
This parameter is self-explanatory. The value of the URL is used for creating parameters for the `InitialContext` object. This helps avoid making manual changes to the configuration file if the BEA WebLogic server is remote, or uses a different port number.

If debug mode is selected, the Navigation Page of the JSP client runs automatically. In this case, the *Show result after generation* option is ignored.

For BEA WebLogic 6.1

The fields on this page are self-explanatory. Some of the fields are the same as for BEA WebLogic 5.1. You should define the name of the target Web Application diagram, and check *Open target Web Application diagram* to see this diagram. You should also specify *Web server host name* and *Web server host number*, which are `localhost` and `7001` by default.

Figure 210

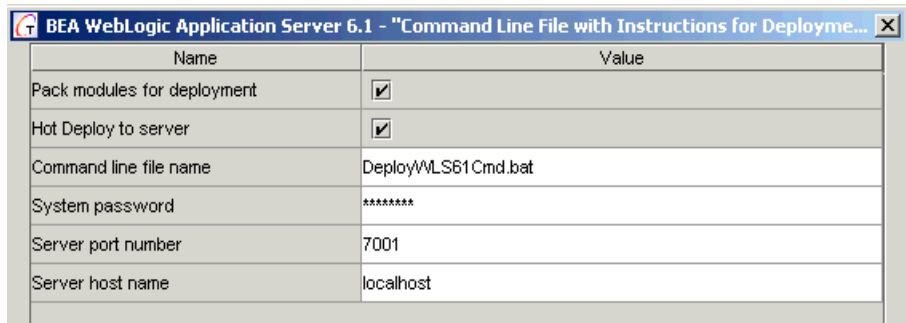


Name	Value
Root path for JSP storage	D:\Together_60\myprojects\untitled2\myJSP
Name of the target Web Application Diagram	JSP_Client_for_EJB
Open target Web Application Diagram	☒
Clear the target directory before generation	☐
Show JSP client in default Internet browser	☒
Web server host name	localhost
Web server port number	7001

Command Line File with Instructions for Deployment page

This page is displayed if you checked the box *Generate a command line file with instructions for deployment* on the first page of the expert. The fields are self-explanatory.

Figure 211



Name	Value
Pack modules for deployment	☒
Hot Deploy to server	☒
Command line file name	DeployWLS61Cmd.bat
System password	*****
Server port number	7001
Server host name	localhost

Making transitions between application servers

Transitions between specifications

Assume that you have already implemented your project and have deployed your application to the current application server. However, you want to test how this application works on a different application server, or you need to change the current application server for some reason.

In this case you have two main problems:

- Different servers support different EJB specifications (EJB 1.0, 1.1, 2.0).
- Each application server has its own specific features.

If you want to redeploy your application to a new application server, check which EJB specification the new server supports.

If the new application server supports the same specification as the first server (for example, your project was implemented using EJB 1.1, and the targeted server also supports EJB 1.1), you only need to consider the specific features of the application server.

If the new application server does not support the current EJB specification (for example, it supports a later EJB specification), correct the Java code of the EJB to meet the targeted EJB specification. The details of the Java code implementing the EJB component vary for different EJB specifications and different EJB components. An Entity CMP Bean will require the most serious code corrections to make a transition from one specification to another.

Together verification and correction of EJBs

To automatically perform the main part of EJB code corrections connected with the specification's version, open the J2EE Deployment Expert and select the application server for redeployment. Check the box *Verify and/or correct EJBs*, and during the deployment process Together will determine which EJB specification this application server supports, and EJBs will be verified and corrected according to the selected specification (EJB 1.0, 1.1, or 2.0).

Only the methods described in the EJB x.x Specification are verified and corrected in Together automatically. The list of verification features (what actually is verified and corrected) are located in “Verification and Correction of EJBs” on page 506.

Manual correction of EJBs

Unfortunately, not all necessary corrections can be made in your code automatically, especially if you move from an earlier EJB specification to a later one.

For example, if your own methods exist in the EJB code (methods that are not included in the selected EJB specification), Together verifies these methods, and if there are methods that do not match the selected EJB specification, Together tries to correct it. The main purpose of correction is getting the correct code (according to the selected EJB specification), so that it can be compiled and deployed. Mainly, this concerns the methods implementing the business logic. If Together corrects a business method, the body of this method is replaced with the default (empty) body.

The backup of all Java classes of the project, including Home and Remote interfaces, is saved in the backup folder. Only you can decide what to do with these methods. You can rewrite the code for the body of the method, or use the fragments of your old code from the backup file.

Together corrects Home and Remote interfaces automatically. If you do not want to keep the corrections, you can use the backup code, but make any necessary adjustments to meet the selected EJB specification.

So, after Together verification and correction you will still need to correct your Java code manually (according to the selected EJB specification), but these changes should be minimal.

How to move from EJB 2.0 to EJB 1.0

Assume that you have developed an application using the EJB 2.0 specification, and you want to deploy it to an application server that supports the EJB 1.0 specification. It is rare to move from a later specification to an earlier one, but this situation is possible if you only have access to an application server that supports the EJB 1.0 specification.

Naturally, your code written according to the EJB 2.0 specification can contain constructions unknown in the EJB 1.0 specification. For understanding the correction process, you can take a simple sample from BEA WebLogic 6.1 (EJB 2.0 specification), and see make corrections to the EJB code for deploying this application to IBM WebSphere 3.5 (EJB 1.0 specification).

Note that it is simpler to correct the code of a Session Bean than to do this for an Entity EJB, especially for a CMP Entity EJB.

When Together verifies and corrects your Java code, the specific features of EJB 2.0 Specification not defined in EJB 1.0 Specification are removed, and the specific features of EJB 1.0 Specification are added.

Together automatically makes the following main changes in the Java code of the EJB implementation made during the verification/correction process:

- Removes EJB Exceptions.
- Adds RemoteException.
- Changes the returned value of the `ejbCreate` method to void for CMP EJBs.
- Changes `Collection` class to `Enumeration` in `findNullAccounts` and `findBigAccounts` methods.
- Comments out (removes) `returnstatement` in the method body.
- Changes transaction attributes.
- Removes the statement `abstract` in CMP EJB implementation classes and in accessors and mutators for business properties representing database fields for a CMP EJB.

Together verification/correction does not take into account any EJB elements defined in the later specification and not defined in the earlier one (for example, message-driven beans are defined only in the EJB 2.0 Specification).

After automatic correction, you should analyze your code and correct it manually if necessary.

How to move from EJB 1.0 to EJB 1.1

Assume that you have developed an application using the EJB 1.0 specification, and you want to deploy it to an application server that supports the EJB 1.1 specification.

Possibly you might need to move from an earlier specification to a later one, but this can require some changes in Java constructions.

For understanding the correction process, take a simple sample from IBM WebSphere 3.5 (EJB 1.0 specification) and make corrections to the EJB code for deploying this

application to BEA WebLogic 5.1.0 (EJB 1.1 specification). When Together verifies and corrects your Java code, the specific features of the EJB 1.0 Specification not defined in the EJB 1.1 Specification are removed, and the specific features of the EJB 1.1 Specification are added. The changes required for a Session Bean are minimal.

Transitions between application servers

If you want to transfer a project from the current application server to another one, you will need to make changes in the following places:

- Server-dependent deployment descriptors.
- Code of the corresponding client connected with the selected application server.
- EJB Java code, if the two application servers support different EJB specifications.
- Properties of the targeted application server.

Changes in the deployment descriptors

The J2EE Deployment Expert generates all deployment descriptors, including server-dependent ones. You can edit these descriptors only during the deployment process. The next time you deploy, the J2EE Deployment Expert generates the deployment descriptors from scratch.

Changes in the client code

You will need to change the `getInitialContext()` method (`Context.INITIAL_CONTEXT_FACTORY`) in the code of the client, according to the selected application server. You will also need to change the `localhost` in the `main()` method (for examples, see “Developing and deploying clients for various servers” on page 617).

Changes connected with the EJB specification

Changes in the code connected with the difference between EJB specifications are described in “Transitions between specifications” on page 614.

Changes in the properties of the application server

Different application servers have different properties and should be defined for deployment in the control panel of the selected application server. You should fill in the server parameters on the server-specific page of the J2EE Deployment Expert.

If your application works with a database, before deployment you will need to edit the file `config.xml` to create a database pool and a data source.

Developing and deploying clients for various servers

The client code highly depends on the selected application server. To understand the main approaches to this, you can refer to “Creating a client.”

For example, the simple client for the HelloWorld project for BEA WebLogic application server (%TG/samples/java/ejb/J2ee/HelloWorld) is implemented with the code:

```
package client.weblogic;
import javax.naming.*;
import java.util.Properties;
import hello.*;
/**
public class HelloClient {
public static void main(String[] argv) {
try{
Properties props = System.getProperties();
props.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
props.put(Context.PROVIDER_URL, "t3://localhost:7001");
Context ctx = new InitialContext(props);
HelloHome home =
(HelloHome)javax.rmi.PortableRemoteObject.narrow(ctx.lookup("hello.Hello
Home"),HelloHome.class);
Hello hello = home.create();
System.out.println(hello.hello());
hello.remove();
}catch(Exception e){ e.printStackTrace(); }
}
}
```

Together can generate a simple JSP Client automatically. If you are going to redeploy your client to another application server, you should edit the Client class and correct the methods listed in the following paragraphs.

Correct findByPrimaryKey method

If there are CMP Entity EJBs in your project, you should use the findByPrimaryKey method (the code above is for a project that contains only a Session EJB, so there are not any findByPrimaryKey methods). There is a difference in the notation of the findByPrimaryKey method for different EJB specifications.

For example, for the EJB 2.0 specification you should use the notation findByPrimaryKey(id), and for the EJB 1.0 specification you should use the notation findByPrimaryKey(new <Name>PK(id)), where <Name>PK is the returned type of Primary Key.

Correct localhost

The localhost is different for different application servers, so you should correct the main() method according to the selected application server.

For example:

BEA WebLogic 6.0: `String url ="t3://localhost:7001"`

IBM WebSphere 3.5: `String url ="iiop://localhost:900"`

Correct getInitialContext method

The `getInitialContext()` method depends on the application server too.

For example,

for BEA WebLogic 6.0 you should write:

```
h.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.initialContextFactory")
```

for IBM WebSphere 3.5 you should write:

```
h.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.ejs.ns.jndi.CNInitialContextFactory")
```

Correct lookup

In CMP EJBs, the lookup method depends on the selected EJB specification.

For example,

for BEA WebLogic 6.0 you should write:

```
in lookupHome, Object home = (<Name>Home)
ctx.lookup("containerManaged.AccountHome")
```

for IBM WebSphere 3.5 you should write:

```
Object home = (<Name>Home) ctx.lookup("<Name>Home")
```

PART
8

WEB SERVICES

- Chapter 35, “Designing and Developing Web Services”
- Chapter 36, “Deploying a Web Service”
- Chapter 37, “UDDI Browser”

CHAPTER 35

DESIGNING AND DEVELOPING WEB SERVICES

This chapter includes the following topics:

- “Supported application servers” on page 621
- “Creating a Web service for deployment to a specific application server” on page 622
- “Creating Web services by patterns” on page 632
- “Generating a WSDL file from a Web service” on page 633
- “Generating a Web service client” on page 634
- “Generating a WSDL file and a WSDL client using UDDI Browser” on page 637

Supported application servers

Having created a class, you might want to use it as a Web service. Together helps accomplish this task. It is possible to transform a class into a Web service and deploy it to the target application server.

Together provides plug-ins for deployment to the following platforms:

- Apache SOAP ([http:// xml.apache.org/soap](http://xml.apache.org/soap))
- BEA WebLogic 6.1 (<http://e-docs.bea.com/wls/docs61>)
- IBM WebSphere AES 4.0 (<http://www-4.ibm.com/>)

NOTE: Apache SOAP is a lightweight protocol for the exchange of information in a decentralized, distributed environment. It can be used as a client library to invoke SOAP services available elsewhere, or as a server-side tool to implement SOAP accessible services. To use Apache SOAP, it is necessary to install the server-side under an application server, such as Apache Tomcat, or BEA WebLogic Application Server. For details, see Apache SOAP documentation: <http://xml.apache.org/soap/index.html>

Creating a Web service for deployment to a specific application server

To create a Web service, the first step is to choose the target application server. To do this, open the Options dialog, go to *Web Services: Application Server*, and choose the application server from the drop-down list.

The process depend on the current application server. For instructions, go to the appropriate section for the application server you have chosen:

“Apache SOAP” on page 622

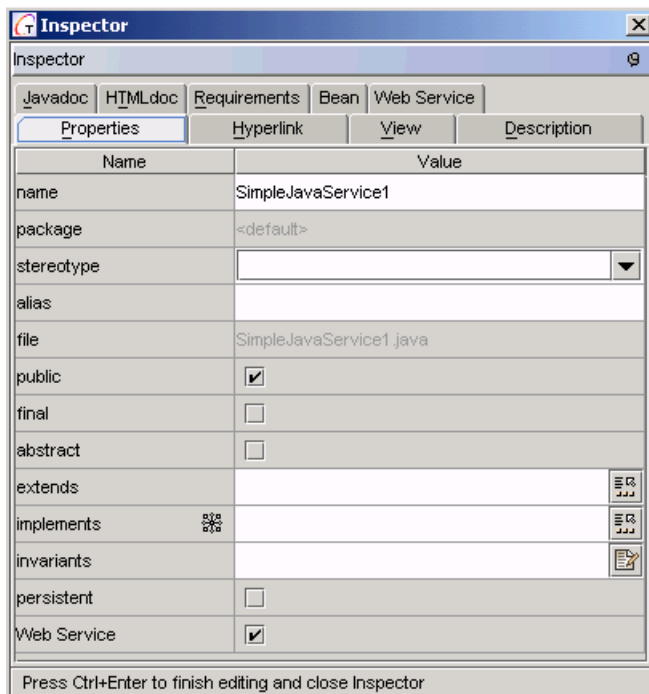
“BEA WebLogic 6.1” on page 629

“IBM WebSphere AES 4.0” on page 632

Apache SOAP

You can create a Web service from a class. Open the necessary Class diagram, and select the class you want to transform to a service. In the example shown in Figure 212, the class is named `SimpleJavaService1`. The class Inspector contains a *Web Service* checkbox. When it is checked, the *Web Service* tab displays in the Inspector:

Figure 212 Creating a web service from a class

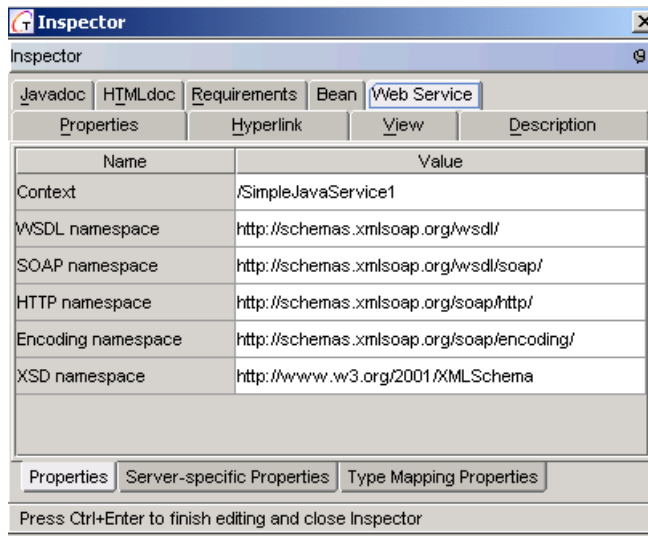


Selecting the *Web Service* tab displays three subtabs:

- Properties
- Server-specific Properties
- Type Mapping Properties

Properties

Figure 213



These parameters are defined in the Options dialog and have default settings.

Context

Context is the relative path to this Web service (relative to the base path where all Web services are stored on the current application server).

Tomcat ignores this path, so it can be left empty. It does not matter which path is specified because in Tomcat all Web services have the same path such as `http://localhost:8080/SOAP/servlet/rpcrouter/`.

Other properties

Other properties on this page are the WSDL namespace prefixes for Apache SOAP. These properties should satisfy an exact WSDL specification.

Additional properties are the default settings for *WSDL namespace*, *SOAP namespace*, *HTTP namespace*, *Encoding namespace*, and *XSD namespace* in .

All parameters from this page are used for generating a WSDL file for the current Web service.

For more information, see <http://www.w3.org/TR/wsdl>.

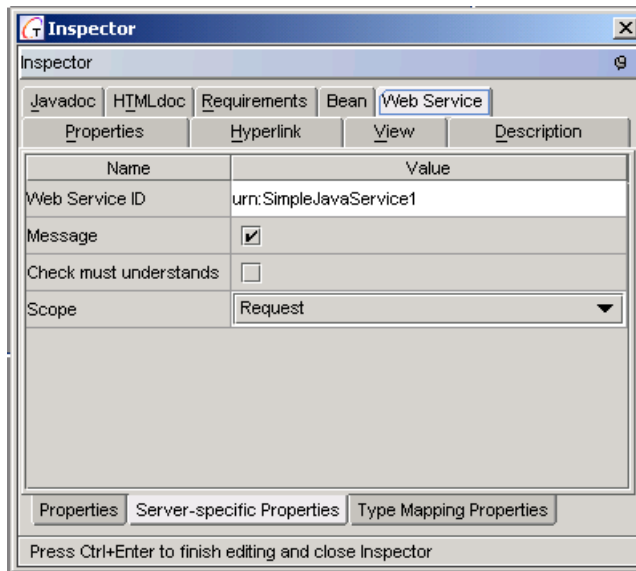
Server-specific properties

If a Web service uses only standard Java classes, a deployment descriptor generated for it looks like the following:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
            id="urn:service-urn" [type="message"]
[checkMustUnderstands="true|false"]>
  <isd:provider type="java"
                scope="Request | Session | Application"
                methods="exposed-methods">
    <isd:java class="implementing-class" [static="true|false"]/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</
isd:faultListener>
</isd:service>
```

The mandatory properties of the deployment descriptor are generated in Together automatically, and you should set some of these properties on the *Server-specific Properties* page.

Figure 214



Web Service ID

The property *service-urn* in a deployment descriptor is the URN that you want to give to a service. *Web Service ID* is a unique Web service identifier on the server. This

parameter is necessary for Tomcat in order to identify the Web service because all Web services are stored in the same place on Tomcat.

Message

If you check *Message*, this class will be defined as a message-style Web service.

The parameter *message* in a deployment descriptor shows whether this service is document-oriented or is a PC-invoked service.

CheckMustUnderstands

SOAP has a global attribute `mustUnderstand` that is used to indicate whether the processing of a SOAP header block is mandatory or optional at the target SOAP node. A value of *1* means that processing the block is mandatory, and *0* means that processing the block is optional.

CheckMustUnderstands is an optional attribute of a deployment descriptor that may be set to either `true` or `false`. If `true`, the server should throw an exception if there are incorrect SOAP headers that were marked as `mustUnderstand`. If the meaning of *checkMustUnderstands* is `false`, the `mustUnderstand` parameter is not analyzed.

Scope

The *scope* property defines the lifetime of a Web service. Select one of the possible values:

- *Request* - The Web service is accessible only during a request. The object will be removed after this request has completed.
- *Session* - The Web service is accessible during an HTTP session.
- *Application* - The lifetime of the Web service depends on the lifetime of the corresponding application. It will be accessible until the servlet which is servicing the requests is terminated.

See <http://xml.apache.org/soap/index.html> for details.

Type mapping properties

A Web service organizes a query using SOAP, and can get and send arguments of the definite base simple types. A WSDL file contains the descriptions of the simple canonical types.

The *types* element encloses data type definitions that are relevant for the exchanged messages. WSDL prefers using XSD as the canonical type system and treats it as the intrinsic type system.

```
<definitions ....>
  <types>
    <xsd:schema .... />*
  </types>
</definitions>
```

Use the XSD type system to define the types in a message, regardless of whether or not the resulting wire format is actually XML, or whether the resulting XSD schema validates the particular wire format.

A SOAP has serializers and deserializers for converting from an XML description to a real object, and from a real object to an XML description.

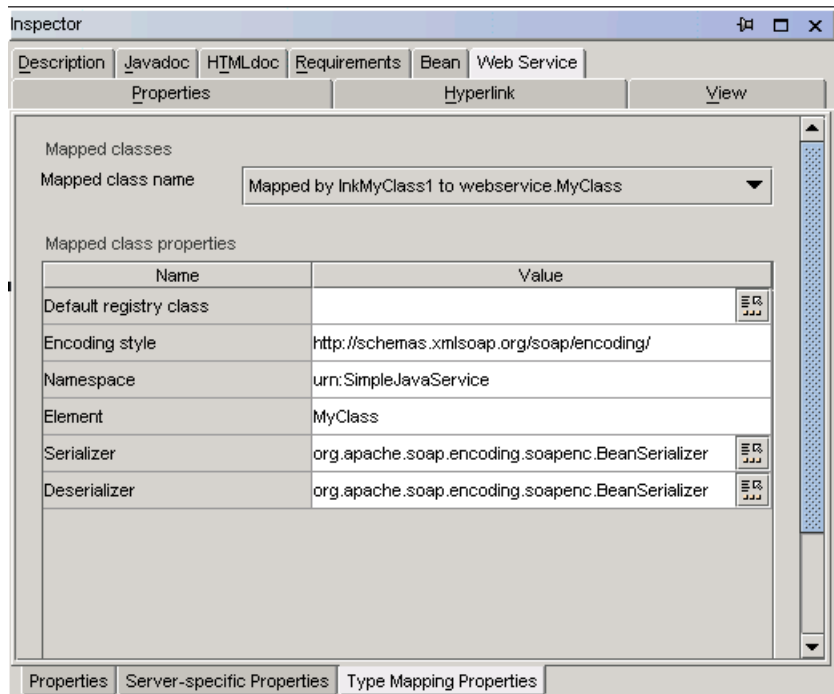
Apache SOAP uses type mapping for more complex types defined on the basis of the simple types. The type mapping is a set of complex types (their names) and a table of conformity of these types to the corresponding serializers. Serializers are the interfaces of Apache SOAP.

For example, you define the exposed `public` method `operation1` in your `SimpleJavaService` Web service, and this method has a complex type `MyClass`. `MyClass` is the type defined by the user. This class is not a canonical type, and there is not a standard description of this type in XSD.

The resulting XSD schema validates the particular wire format. Use the *Type Mapping Properties* tab in this case.

You can find all information connected with Apache SOAP at <http://xml.apache.org/soap/index.html>.

Figure 215



Mapping class name

Select the mapping class name from the list.

Serializer

Apache SOAP uses type mappings to determine how Java data types should be marshalled to and unmarshalled from XML.

To encode a Java class, the current serializer creates a description of the binary data of the fields of this class in the *.xml file.

Select a serializer from the set of Apache SOAP serializers. Complex classes will be referred to by this name in SOAP. `BeanSerializer` is set by default.

Deserializer

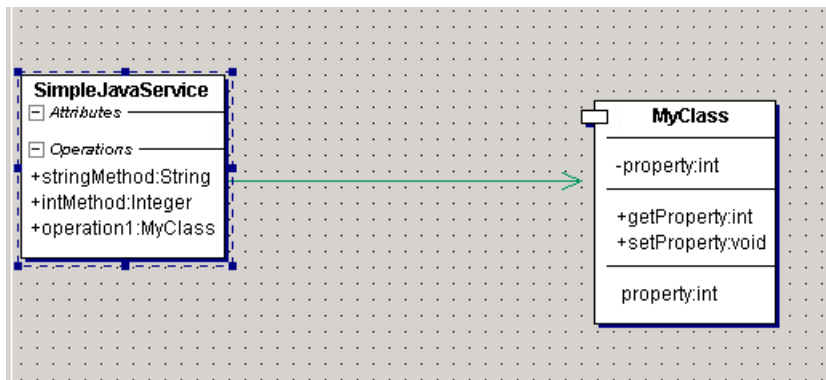
Deserializers are used for decoding from an *.xml file to the fields of a Java class.

Select a deserializer from the set of Apache SOAP deserializers. `BeanSerializer` is set by default. Other parameters are the namespace parameters of a Web service.

Result of type mapping on a Class diagram

As a result, you can see the green type mapping link between the `SimpleJavaService` Web service and the complex class `MyClass` on the diagram:

Figure 216



This link appears automatically. You are ready to deploy your Web service: the descriptions of the complex `MyClass` type will be included in the deployment descriptor.

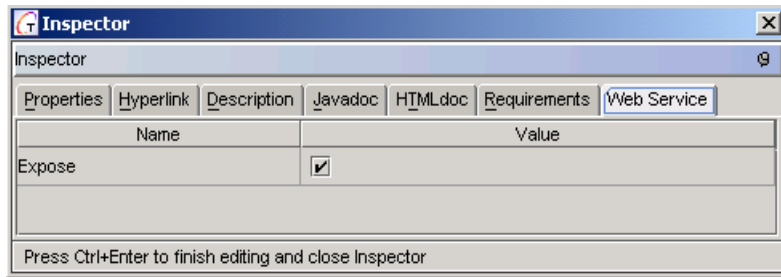
Exposed methods

The parameter *exposed-methods* in a deployment descriptor is a list of exposed methods. Add operations to your class, and define methods of the Web service as exposed methods.

An additional *Web Service* tab appears in the Inspector when the class declares as a Web Service. A client has access to a method of a Web service only if this method is public and exposed. So, the property *expose* is one of the most important properties.

Right-click on the method (for example, `operation1`), choose Properties on the right-click menu, and select the *Web Service* tab.

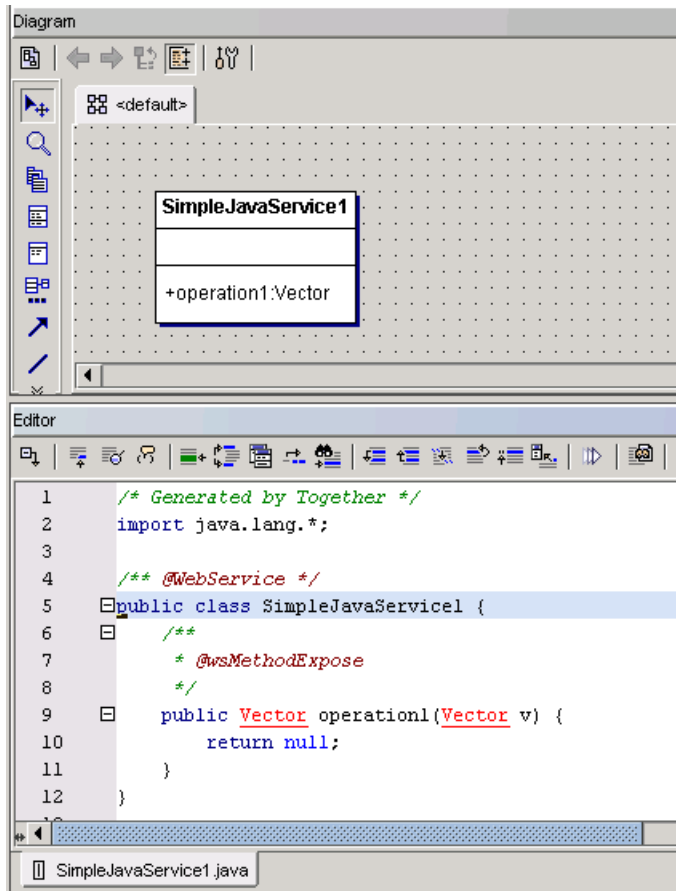
Figure 217



Expose

Check *Expose* to provide access to this method for all clients of this Web service. The following information appears in the corresponding code: `SimpleJavaService1` class is a Web service and `operation1` is an exposed method.

Figure 218



BEA WebLogic 6.1

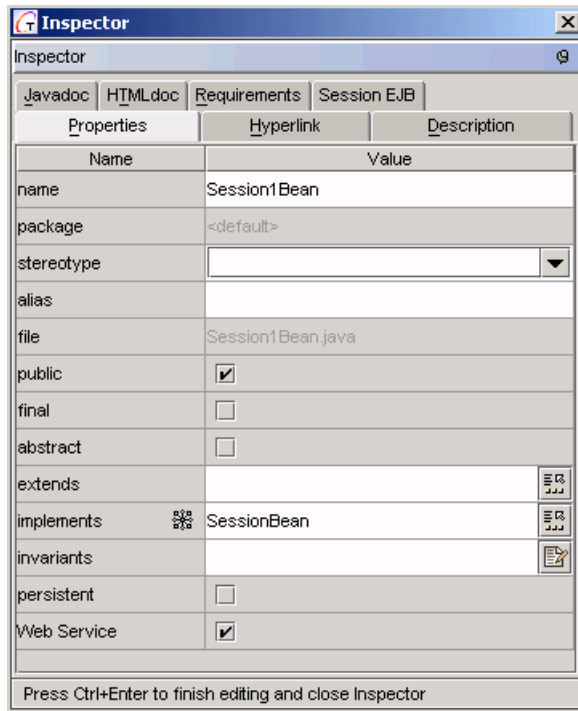
Using WebLogic 6.1, you cannot create a Web service from a class.

You can use two types of Web services: RPC-style (remote procedure call) and message-style.

RPC-style Web service

An RPC-style Web service uses a stateless session EJB. You can create an RPC-style Web service from a session EJB. Open a Class diagram, select the session EJB to base the Web service on, and open the bean's Inspector. The Inspector for a session bean contains the *Web Service* checkbox.

Figure 219



When *Web Service* is checked, this session EJB is defined as an RPC-style Web service.

Message-style Web service

Open an existing EJB Assembler diagram or create a new one. You can add a message-style Web Service using the *Message Web Service* design element:



shows the Message Web Service Inspector.

Figure 220

The screenshot shows a window titled "Inspector" with a tabbed interface. The "Properties" tab is selected. Below the tabs is a table with two columns: "Name" and "Value". The table contains the following rows:

Name	Value
Web Service name	Message Web Service1
Action	send
Destination JNDI name	
Destination type	topic
Connection factory JNDI ...	
URI	

At the bottom of the window, there is a text box with the instruction: "Press Ctrl+Enter to finish editing and close Inspector".

Web Service name

The name of the current Web service.

Action

Specify whether the client that uses this message-style Web service is a sender or a receiver of XML data to the JMS destination.

Select from the possible values: *send* or *receive*.

Destination JNDI name

Type the JNDI name of a JMS topic or queue.

Destination type

Select the type of JMS destination: *topic* or *queue*.

Connection factory JNDI name

This field contains the JNDI name used to create a connection to the JMS destination.

URI

Fill in this field with the URI used by clients to invoke the Web service.

The full URL to access the Web service is:

[protocol]://[host]:[port][context][uri]

IBM WebSphere AES 4.0

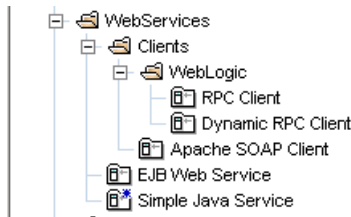
To create a Web service for WebSphere AES 4.0, use the instructions for Apache SOAP in “Apache SOAP” on page 622.

Creating Web services by patterns

To use a pattern for creating a Web service:

1. Create a new Together project or open an existing one.
2. Create a new Class diagram or open an existing one.
3. Click the Class by Pattern button on the diagram toolbar, and place it somewhere on the diagram.
4. In the Choose Pattern dialog, select one of the following patterns for Web services:
 - *EJB Web Service*
 - *Simple Java Service*
5. Click **Finish**. The corresponding Web service appears on the diagram.

Figure 221



In addition, you can use patterns for creating Web service clients.

To use the Web Service Type Mapping pattern:

1. Create a new Together project or open an existing one.
2. Create a new Class diagram or open an existing one.
3. Create a Web service by pattern as described above.
4. Add a complex class on the diagram.
5. Add the operation using the complex class to the Web service.
6. Click the Link by Pattern button on the diagram toolbar for type mapping and then choose the *Web Service Type Mapping* pattern in the Choose Pattern dialog.
7. Draw the link from the Web service component to the class component.

Generating a WSDL file from a Web service

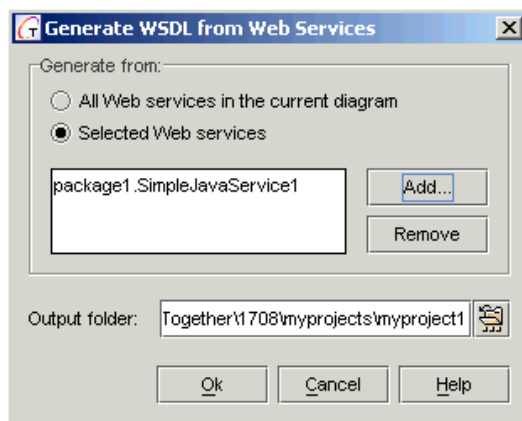
According to the WSDL specification, “WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).” Together generates WSDL files automatically using existing Web services. Web services can be either your own visual design elements on diagrams, or Web services found on the Internet. Together describes all the properties of the current Web service in the generated WSDL file.

Generating a WSDL file using an existing Web service

To generate a WSDL file from a Web service:

1. Create a new Together project or open an existing one.
2. Create a new Class diagram or open an existing one.
3. Create a Web service as described in “Creating Web services by patterns” on page 632. For example, you can use the pattern *Web Services - Simple Java Service* to create a simple Java Web service named `SimpleJavaService1` with the exposed method `operation1`.
4. Choose **Tools | Web Services | Generate WSDL from Web Services** on the main menu. The Generate WSDL from Web Services dialog opens.

Figure 222

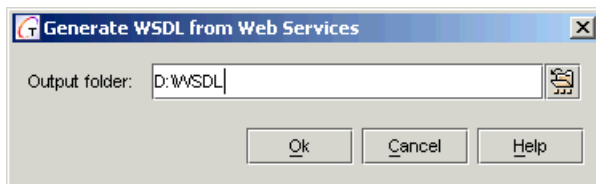


5. Specify which Web services that exist on the diagram should be used for generating WSDL files. You can choose to include all the Web services from this diagram, or only the ones that you select. If you choose *Selected Web services*, click the **Add** button to select the Web services that you want to include. Use the **Remove** button if you want to exclude a Web service from the list.

6. In *Output folder*, use the file chooser to set the output folder for the generated WSDL files.
7. Click **Ok**. The corresponding SimpleJavaService1.wsdl file is created in the specified folder.

Alternatively, to generate a WSDL file for one specific Web service, right-click on this Web service and choose **Generate WSDL** from the right-click menu. Then choose the output folder for saving the WSDL file in and click **Ok** to finish.

Figure 223



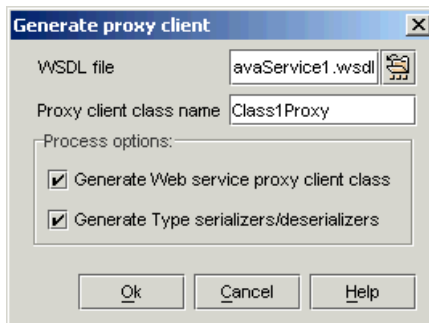
Generating a Web service client

Generating a proxy Web service client using a WSDL file

To generate a proxy Web service client:

1. Create a new Together project or open an existing one.
2. Create a new Class diagram or open an existing one.
3. Make sure that you have the necessary WSDL file.
4. Choose **Tools | Web Services | Web Service Client Generation** on the main menu. The Generate Proxy Client dialog opens.

Figure 224

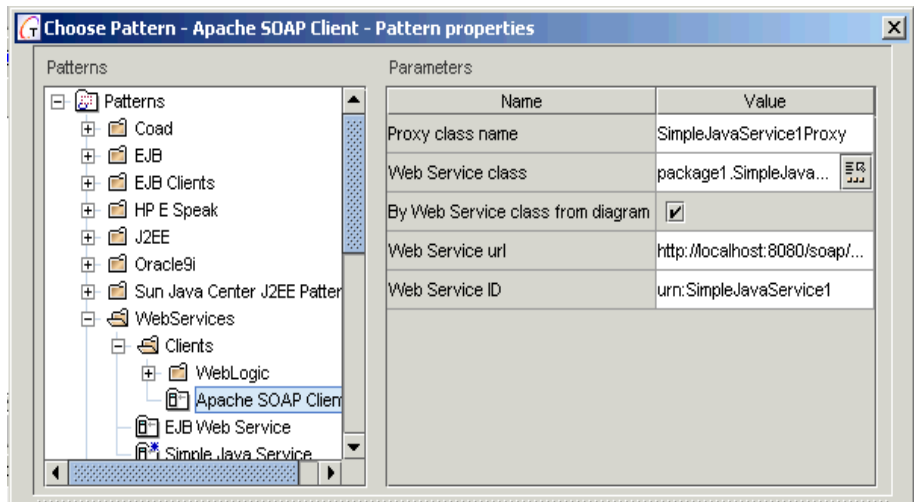


5. Use the file chooser to select the WSDL file. Define the proxy client class name and the process options.
6. Click **Ok**. The corresponding client class displays on the current diagram.
7. Add to and correct the code for the client class as necessary.

Generating a proxy Web service client for Apache SOAP by pattern

To generate a proxy Web service client using patterns:

1. Create a new Together project or open an existing one.
2. Create a new Class diagram or open an existing one.
3. Create a Web service on this diagram, or make sure that you know the folder where the Web service is located.
4. Click the Class by Pattern button on the diagram toolbar, and place it somewhere on this diagram.
5. Select the *Apache SOAP Client* pattern for a Web services client in the Choose Pattern dialog,



6. Set the properties for the client and click **Finish**. The corresponding Web service client displays on the current diagram. View the corresponding code in the Editor pane.

Properties of the pattern for an Apache SOAP proxy client

- The *Proxy class name* field contains the name of the proxy class.
- *Web service class* is the name of the Web service for which you are going to create a proxy client.

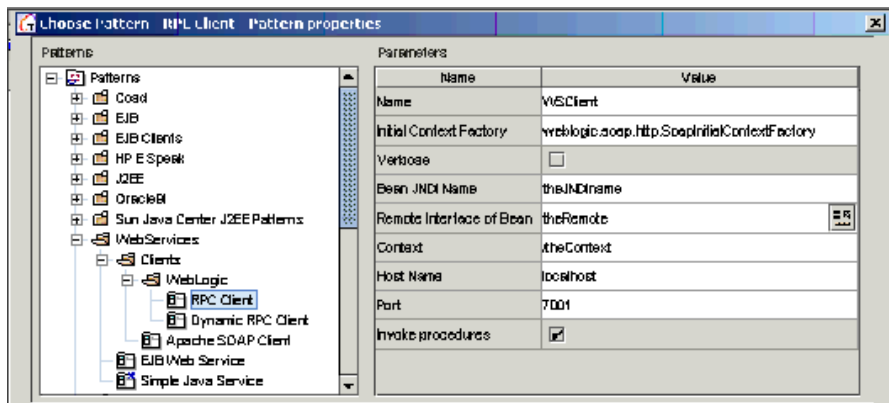
- Check *By Web service class from diagram* if you are using an existing Web service on the Class diagram. An empty proxy client creates without any generative Web service class if the box is unchecked. The corresponding code contains commented lines with a usage example.
- *Web service URL* defines where this Web service is located.
- *Web service ID* defines a unique number that identifies this Web service if other Web services have the same URL.

Generating a Web service client for BEA WebLogic by pattern

To generate a Web service client using patterns:

1. Steps 1 - 4 are the same as for the Apache SOAP Web service client pattern described in “Generating a proxy Web service client for Apache SOAP by pattern” on page 635.
2. In the Choose Pattern dialog, select one of the following patterns in *Web Services - Clients - WebLogic*:
 - *RPC Client* - Remote Procedure Call, a protocol that one program uses to request a service from a program located on another computer in a network without details.
 - *Dynamic RPC Client* - Web service dynamic client for BEA WebLogic server. This kind of client does not use a remote interface to access the Web service.
3. Set the pattern properties and click **Finish**. The corresponding Web service client displays on the current diagram. View the corresponding code in the Editor pane.

Figure 225



Properties of the pattern for a WebLogic client

- *Name* - The name of the client class to generate.
- *Initial Context Factory* - The Initial Context Factory that should be used to invoke the Web service.

- *Bean JNDI name* - The JNDI name of the bean that implements the Web service.
- *Remote interface of Bean* - The name of the remote interface of the bean that implements the Web service.
- *Context* - The context of the Web service that can be defined during deployment using the Web Services Deployment Expert.
- *Host name* - The name of the server where the Web service is deployed and can be invoked.
- *Port* - The binding HTTP port of the server where the Web service is deployed.
- *Invoke procedures* - If this box is checked, code is generated in the client to invoke each Web service method that is in the bean.

Generating a WSDL file and a WSDL client using UDDI Browser

To generate a WSDL file using the UDDI Browser:

1. Choose **Tools | Web Services | UDDI Browser** on the main menu to open the UDDI Browser.
2. Search for a Web service in the UDDI Browser (for instructions, see “Searching for Web objects” on page 649).
3. If the Web service that you found has a corresponding WSDL file, go to the **WSDL** tab on the Search page of the UDDI Browser and select the necessary address in the drop-down list. View the corresponding WSDL file in the right pane.
4. For information on how to save the WSDL file and generate the corresponding WSDL client, refer to “Saving WSDL files” on page 653 and “Generating a WSDL client” on page 653.

CHAPTER 36

DEPLOYING A WEB SERVICE

This chapter includes the following topics:

- “Deployment properties” on page 638
- “Running the Web Services Deployment Expert” on page 639
- “Web Services Deployment Expert for Apache SOAP” on page 639
- “Web Services Deployment Expert for BEA WebLogic 6.1” on page 642
- “Web Services Deployment Expert for IBM WebSphere AES 4.0” on page 645

Deployment properties

Having created a Web service, you can deploy this service to the selected application server using the Web Services Deployment Expert.

Together provides deployment to the most popular platforms: Apache SOAP (<http://xml.apache.org/soap>), BEA WebLogic 6.1 Application Server (<http://e-docs.bea.com/wls/docs61>), and IBM WebSphere AES 4.0 (<http://www-4.ibm.com>).

You can change the default deployment properties for different application servers in the Options dialog: **Tools | Options | <level> - WebServices**. These options include the target hosts for ApacheSOAP and BEA WebLogic 6.1, and namespace parameters.

The current application server gets these parameters when your Web service is deploying to the server.

Running the Web Services Deployment Expert

Together provides a convenient user interface that greatly simplifies the Web service deployment process. The interface of the Web Services Deployment Expert is almost the same as the interface of the J2EE Deployment Expert (see Chapter 34, “Deploying J2EE”).

In the expert, you can specify:

- The target server platform.
- Which deployment-related actions you want to take place (such as compiling).
- Paths to the server, server tools, and deployment output.
- Connection parameters for the server.

NOTE: Before using the Web Services Deployment Expert, be sure that the server for the deployment of Web services is installed on your computer.

To run the Web Services Deployment Expert:

1. Open the project and the Class diagram containing the Web services to be deployed.
2. Choose **Deploy | Web Services Deployment Expert** on the main menu to launch the expert.
3. Choose the target server platform, and set the other options as desired. Click **Next**.
4. On the Common Properties page, specify the path to the JDK, the path to the server, and the path to the temporary files folder. The specific fields on this page depend on the current application server. Click **Next** to continue.

View default settings for registering Web services during runtime for the current application server.

5. Click **Finish**. Depending on your option selections, Together can handle interaction with the compiler to compile classes, generate the XML deployment descriptors, and register Web services on the target application server.

Each page of the Web Services Expert contains a number of tasks. After you have filled in all necessary fields and clicked Finish, the Web Services Expert starts executing the selected tasks.

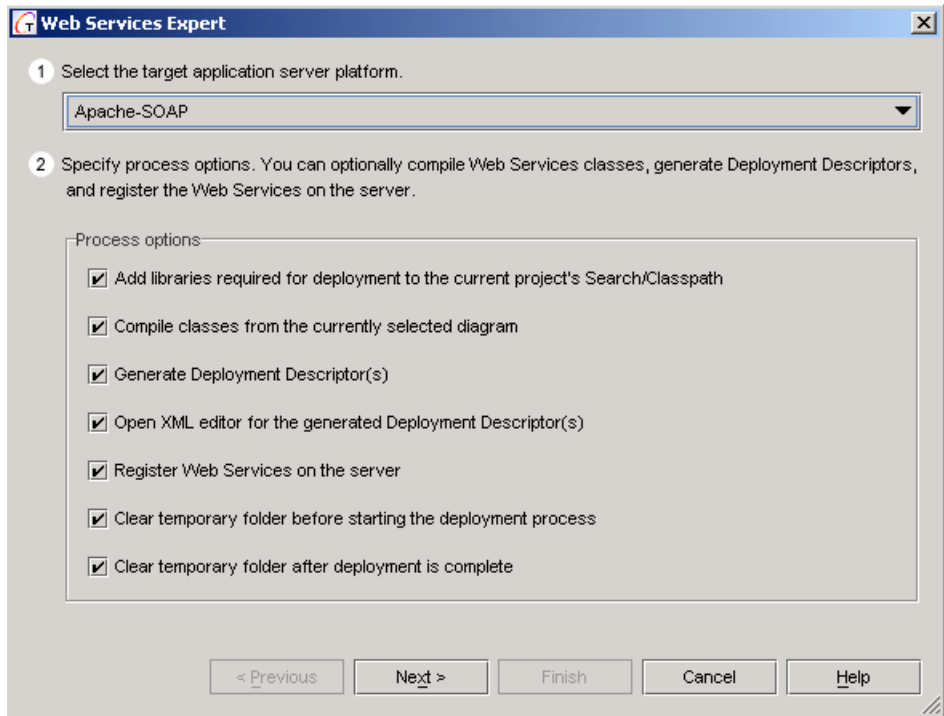
The following sections describe the server-specific pages.

Web Services Deployment Expert for Apache SOAP

When you open the Web Services Deployment Expert from the main menu (Deploy | Web Services Deployment Expert) and select *Apache SOAP* as the current application server, the expert displays options specific to Apache SOAP.

Process options

Figure 226



Add libraries required for deployment to the current project's Search/Classpath

It is recommended to check this box to add all libraries necessary for deploying.

Compile classes from the currently selected diagram

If the classes need to be compiled, check the *Compile classes* option. If they are already compiled, clear this option.

Generate Deployment Descriptor(s)

Check this box if it is necessary to generate deployment descriptors. Clear this checkbox if you do not want to generate them at this moment.

Register Web Services on the server

Check this box if you want to register Web services on the application server.

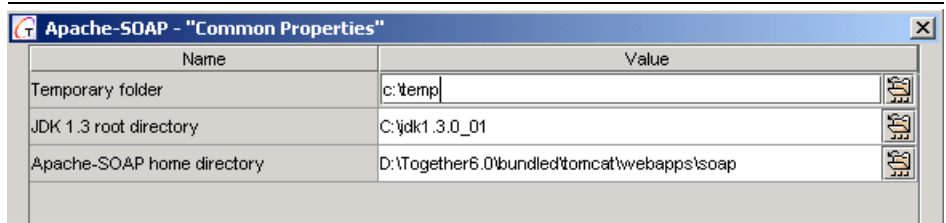
Clear temporary folder

The other options are used for clearing the temporary folder before or after deployment is complete. This is necessary for removing unneeded information.

When all options are set, click **Next** to continue.

Common Properties

Figure 227



The fields on the Common Properties page are quite self-explanatory. However, here is a brief description of the Apache SOAP home directory:

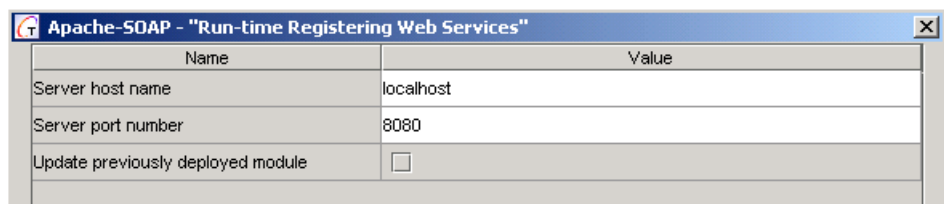
Apache SOAP home directory

Tomcat, which comes bundled with Together, is pre-configured to support Apache SOAP. If you are going to create a client or a service, the library `$TGH$/bundled/tomcat/soap` should be added to the project path. Make sure that Together added `soap.jar` and `xerces.jar` by default. If you want to use another version of Tomcat instead of the bundled version, you need to add Apache SOAP to Tomcat manually.

Click **Next** to continue.

Run-time Registering Web Services

Figure 228



Server host name

As a rule, the server host is assigned to localhost because the Web Services Deployment Expert uses the file system access to the application server with Apache SOAP, and therefore the server runs on the same computer.

Server port number

The default server port is 8080.

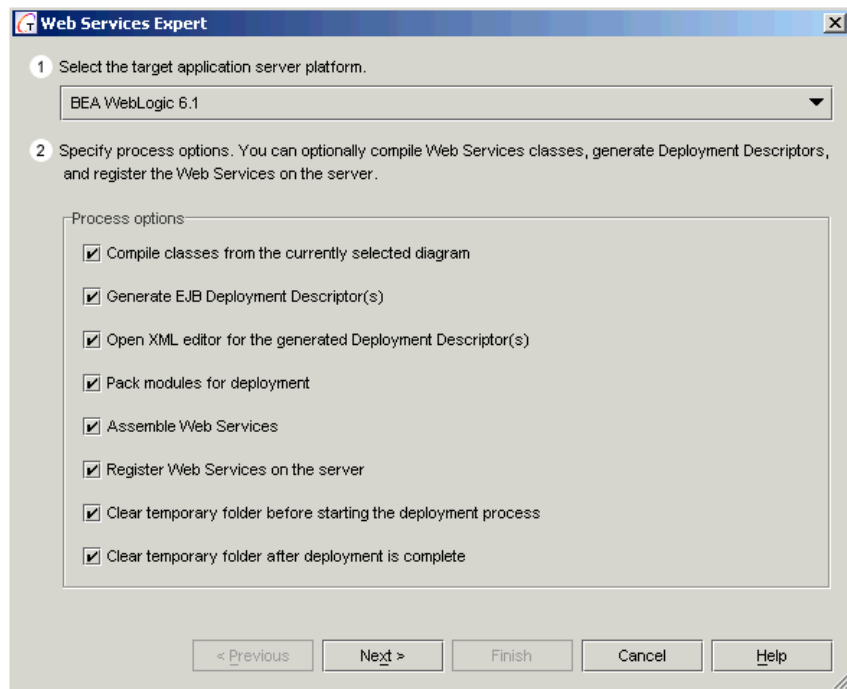
Start Tomcat, and click **Finish** to deploy from the current diagram.

Web Services Deployment Expert for BEA WebLogic 6.1

When you open the Web Services Deployment Expert from the main menu (Deploy | Web Services Deployment Expert) and select *BEA WebLogic 6.1* as the current application server, the expert displays options specific to BEA WebLogic 6.1 AS.

Process options

Figure 229



Compile classes from the currently selected diagram

Check this box to compile the classes from the selected diagram and pack them in a JAR file. The new archive is stored in a temporary folder. Standard Sun tools (`javac.exe` and `jar.exe`) are used for compilation and packing.

Generate EJB Deployment Descriptor

Check this box to create a Deployment Descriptor for all RPC Web Services and the `build.xml` file.

Open XML editor for the generated Deployment Descriptor and build.xml

Check this box to automatically open the XML Editor, edit the generated deployment descriptor, and the `build.xml` file, which are created during this process.

Pack modules for deployment

Check this box to create a deployable `*.jar` archive (standard JAR for deployment of an RPC Web Service Session EJB).

Assemble Web Services

Check this box to generate a `build.xml` file and generate a deployable `*.ear` archive using Java Ant.

Register Web Services on the server

Check this box to register RPC-style or message-style Web Services on the target application server (deployment of the generated `*.ear`).

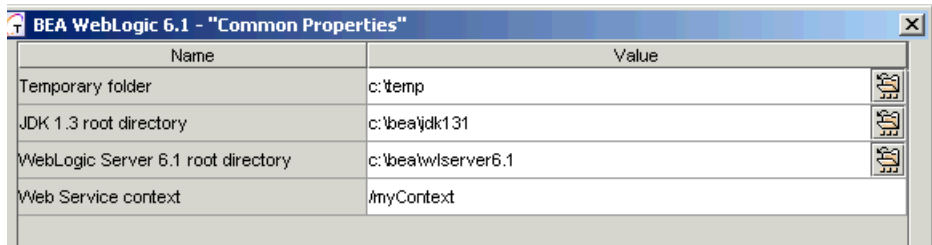
Clear temporary folder...

The other checkboxes are used for clearing a temporary folder before or after deployment is complete. This is necessary for removing unneeded information.

When all properties are set, click **Next** to continue.

Common Properties

Figure 230



The dialog box titled "BEA WebLogic 6.1 - 'Common Properties'" contains a table with two columns: "Name" and "Value".

Name	Value
Temporary folder	c:\temp
JDK 1.3 root directory	c:\bea\jdk131
WebLogic Server 6.1 root directory	c:\bea\wlserver6.1
Web Service context	/myContext

Almost all fields on this page are self-explanatory.

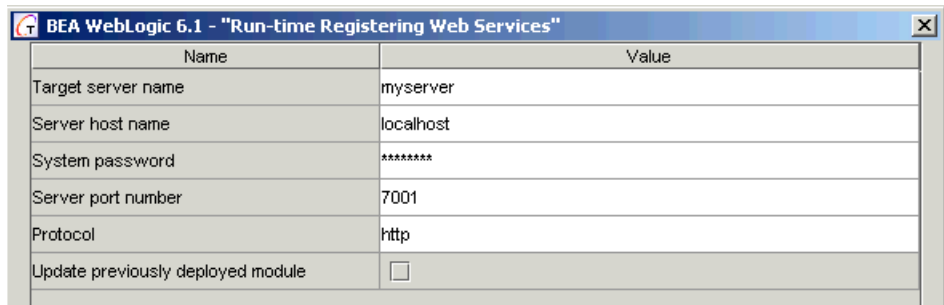
Web Service context

Specifies the name of the context root of the Web service, such as /myContext. You can later use this context root in the URL to access the deployed Web service and the client *.jar file.

Click **Next** to continue.

Run-time Registering Web Services

Figure 231



The dialog box titled "BEA WebLogic 6.1 - 'Run-time Registering Web Services'" contains a table with two columns: "Name" and "Value".

Name	Value
Target server name	myserver
Server host name	localhost
System password	*****
Server port number	7001
Protocol	http
Update previously deployed module	☐

Target server name

The name of the server that you want to use for your applications.

Server host name

This field contains the name of the host that is running the BEA WebLogic Server hosting the Web service. As a rule, the server host is assigned to localhost because this

Web Services Deployment Expert uses the file system access to BEA WebLogic 6.1 application server, and therefore the server runs on the same computer.

Server port number

This field contains the port number of BEA WebLogic Server 6.1. The default server port is 7001.

Web Service context

This field contains the context root of the Web services. Use this value to access the deployed Web service with an URL.

Protocol

This field contains the protocol that clients use to access the Web service. There are two possible values: http or https. The default value is http.

Server root directory

This field contains the path to the directory that includes the BEA WebLogic startup file, config files for your applications, and so on.

Start BEA WebLogic 6.1, and click **Finish** to deploy from the current diagram.

NOTE: Start BEA WebLogic 6.1 from Together (see “Starting BEA WebLogic Application Server from within Together” on page 605).

Web Services Deployment Expert for IBM WebSphere AES 4.0

When you open the Web Services Deployment Expert from the main menu (Deploy | Web Services Deployment Expert) and select *IBM WebSphere AES 4.0* as the current application server, the expert displays options specific to IBM WebSphere AES 4.0.

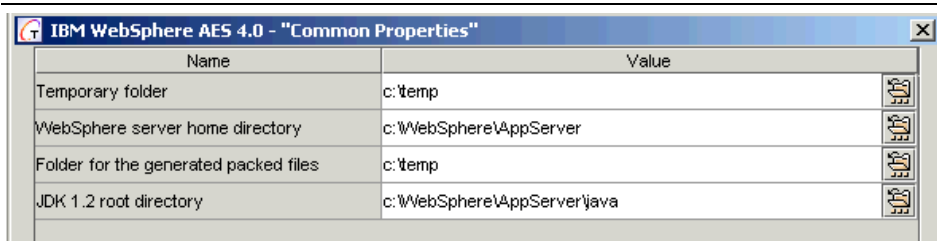
Process options

This page is the same as for Apache SOAP (see “Process options” on page 640). The only difference is in the selected application server.

After setting the options you want, click **Next** to continue.

Common Properties

Figure 232

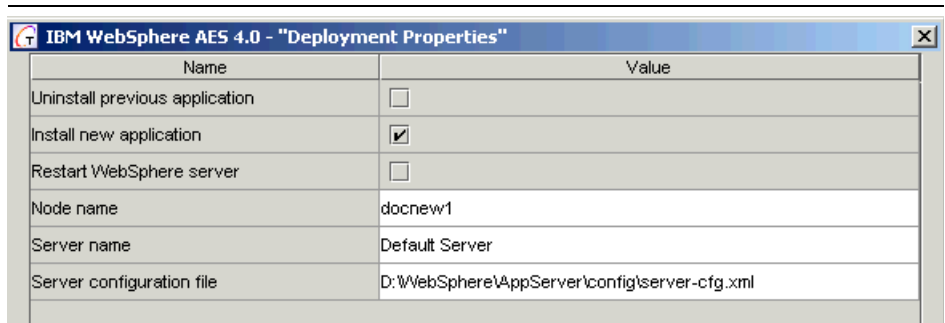


Name	Value
Temporary folder	c:\temp
WebSphere server home directory	c:\WebSphere\AppServer
Folder for the generated packed files	c:\temp
JDK 1.2 root directory	c:\WebSphere\AppServer\java

These properties specify the location of IBM WebSphere 4.0, JDK 1.2 folder, a folder for storing packed files, and a temporary folder.

Deployment Properties

Figure 233



Name	Value
Uninstall previous application	☐
Install new application	☒
Restart WebSphere server	☐
Node name	docnew1
Server name	Default Server
Server configuration file	D:\WebSphere\AppServer\config\server-cfg.xml

The fields on this page are self-explanatory.

Check the necessary boxes to uninstall a previous application, install a new application, or restart IBM WebSphere Application Server.

Node name

This is a node name in the WebSphere console.

Server name

This is the name of your server in the WebSphere console.

Server configuration file

This field specifies the path to the configuration file server-cfg.xml.

Start IBM WebSphere 4.0 Application Server and click **Finish** to deploy your Web services from the current diagram.

CHAPTER 37

UDDI BROWSER

This chapter includes the following topics:

- “Functionality of the UDDI Browser” on page 37
- “Using the UDDI Browser” on page 37
- “Searching for Web objects” on page 37
- “Publishing Web services” on page 37

Functionality of the UDDI Browser

Universal Description, Discovery and Integration (UDDI) is a SOAP-based protocol.

The UDDI Browser is a Together tool for working with UDDI registries that are accessed on operator sites on the Internet.

The UDDI Browser can be used in multiple ways: to access the UDDI registry, to obtain information about Web services, to execute UDDI requests, to call Web services, and to register your own Web services.

In addition, you can obtain the WSDL (Web Services Description Language) file for a Web service (if the corresponding WSDL file exists). The WSDL file is an endpoint for obtaining information about a Web service. The WSDL file contains information about interfaces and method calls in the selected service. The UDDI Browser processes the WSDL file and displays the information about the Web service.

Using the UDDI Browser

This section describes how to open and use the Together UDDI Browser.

The UDDI Browser does not require an open project in order to work, so you can open the UDDI Browser in Together without creating or opening a project. To generate a WSDL proxy client, open a project.

Choose **Tools | Web Services | UDDI Browser** on the main menu to open the UDDI Browser.

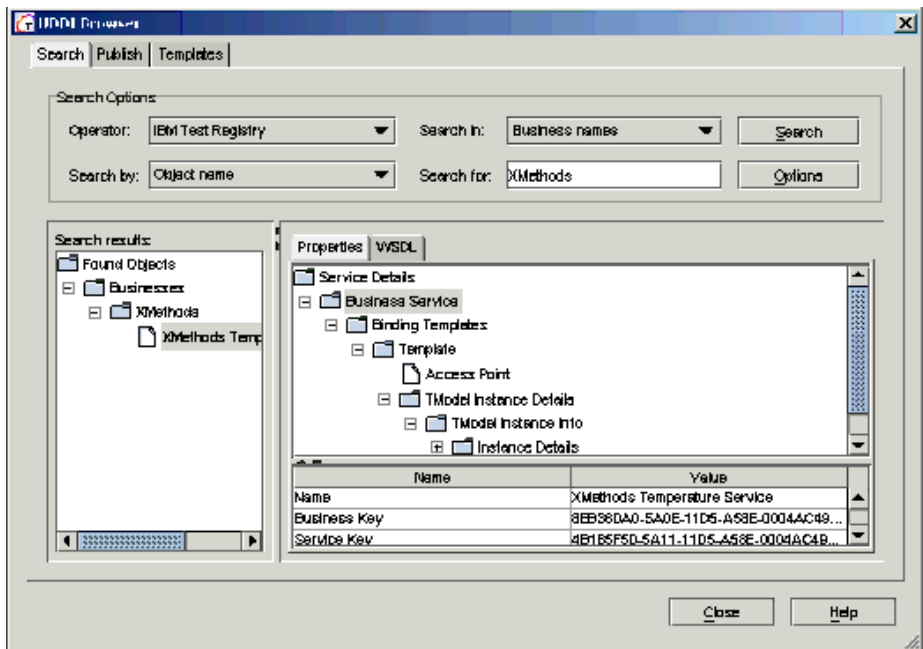
The UDDI Browser has three main tabs: *Search*, *Publish*, and *Templates*.

Searching for Web objects

Use the *Search* tab to organize a search for Web objects. You can find a UDDI registry and the names of registered objects using some context information about the Web business or Web service.

The left pane of the window contains a search results tree with a list of registered Web services or Businesses that were found according to the search options specified. The right pane contains information about properties of selected objects.

Figure 234



Search Options pane

Under *Search Options*, you can set up the search procedure.

In *Operator* you can select an inquiry operator Internet site with an installed UDDI registry from the default list of site addresses. The *IBM Test Registry* operator is set by default. In addition, you can add a different Internet address to the list of operators.

Click **Options** to see the properties of the current operator, or to add a new one (refer to “UDDI Browser Options” on page 37).

Search in sets the type of UDDI object to search for.

- Select *Business names* to search Businesses (objects that contain a list of Web services).
- Select *Service names* to search Services (the Web service objects themselves).
- Select *TModel names* to search Service Types (tModels).

Search by switches between different search methods.

- Select *Object name* to search by a simple object name mask.
- Select *Category bag* to search by a category bag mask.
- Select *Identifier bag* to search by an identifier bag mask.
- Select *TModel bag* to search by a tModel bag mask.
- Select *Discovery URLs* to search by a discovery URLs mask.

Different types of objects can be found using different search methods:

- Businesses support search by object name, category bag, identifier bag, TModel bag, or discovery URLs.
- Services support search by object name, category bag, or tModel bags.
- TModels support search by object name, category bag, or identifier bags. The category of objects contains some information about Web services (the area of knowledge, geographical location, and so on).

In *Search for*, type the object name or appropriate context information to search for. You can define a name mask for searching UDDI objects (such as *XMethods* in Figure 234).

Search button

After you set the search options, click **Search** to start the search operation. The results are displayed in a tree under *Search results* on the left side of the window.

UDDI Browser Options

Click the **Options** button to view and change the operator’s settings and search settings. The UDDI Browser Options dialog opens.

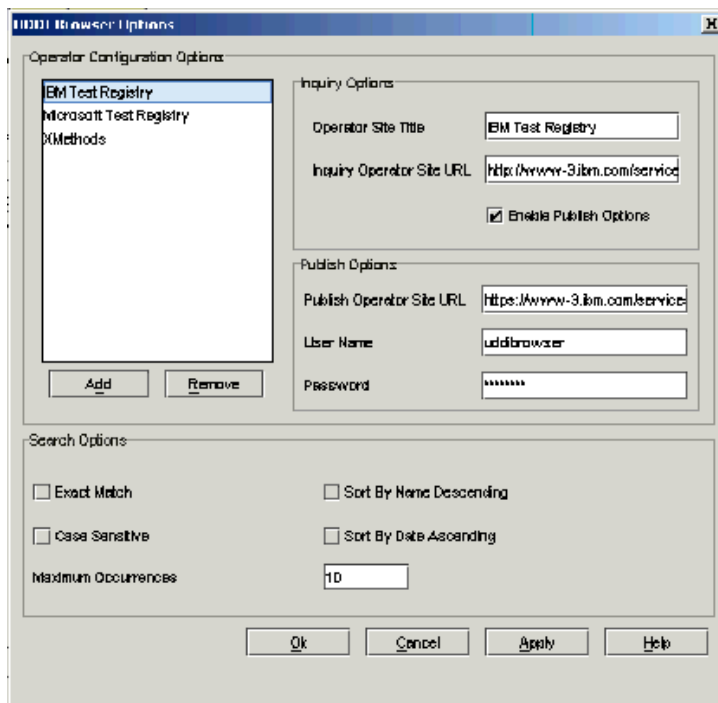
Under *Operator Configuration Options*, the left panel contains the list of operators. You can add and remove operators using the Add and Remove buttons, respectively.

Under *Inquiry Options* you can set the properties of the selected operator.

- *Operator Site Title* sets the operator title that is shown in the *Operator* drop-down list on the *Search* tab.
- *Inquiry Operator Site URL* sets the inquiry registry entry point URL, and gives you the opportunity to select a URL for an Internet site with an installed UDDI registry from the default list of site addresses. In addition, you can add a new Web address.
- *Enable Publish Options* is checked if this operator has a publish registry to enable publish operations. If unchecked, the *Publish Options* group is disabled, and this

operator is not displayed in the list of publish operators in the *Operator* drop-down list on the *Publish* tab. In this case, it is displayed only on the *Search* tab.

Figure 235



Under *Publish Options*, you can define publishing parameters.

- *Publish Operator Site URL* sets the publish registry entry point URL.
- *User Name* defines the user name for authorization on the site.
- *Password* defines the user password for authorization on the site.

For a user name and password register on one of the publish operator sites. To register on the IBM Test Registry, go to <https://www-3.ibm.com/services/uddi/testregistry/protect/registry.html>.

Under *Search Options*, you can set filters for a more complex search procedure.

This section defines the search filtering options.

- *Exact Match* requires that the name mask in *Search for* is the exact name of the searched object. If checked, the string specified in the Search for field of the UDDI Browser is the exact name of the object to search for. If this box is unchecked, all objects whose names start with this name mask are retrieved.
- *Case Sensitive*: If checked, the case of the object name mask characters is taken into account when searching.

- *Sort By Name Descending*: If checked, the results are sorted by name in descending order (starting from the end of the alphabet). If unchecked, names are sorted in ascending order by default (starting from the beginning of the alphabet).
- *Sort By Date Ascending*: If checked, the results are sorted by date from least recent to most recent.
- *Maximum Occurrences*: This option defines the maximum number of objects to be found and displayed in the search results. Searching stops after reaching this limit.

Click **OK** to save changes to the option settings. **Cancel**: Discards changes and closes the dialog.

Properties subtab on the Search tab

The *Properties* tab contains common organizational information about the object selected in the results tree.

This information contains the business key, the service key, a description of the selected object, and other service information. This information depends on the type of object, and it is stored in the corresponding SOAP message when the UDDI registry object is requested. The request is made when you select the object from the list of results.

WSDL subtab on the Search tab

The WSDL tab is used for visualization of the WSDL data file for the selected Web service. The WSDL file describes the Web service by providing a list of its methods that can be invoked.

Properties of Web businesses

When you are searching for a UDDI object using a mask for the business name, you get a list of businesses with sublists of Web services for each Web business.

If you select a specific Web business in the results tree, you can view the detail properties for this business, such as the owner name, telephones, email, and addresses.

Web businesses do not have WSDL files.

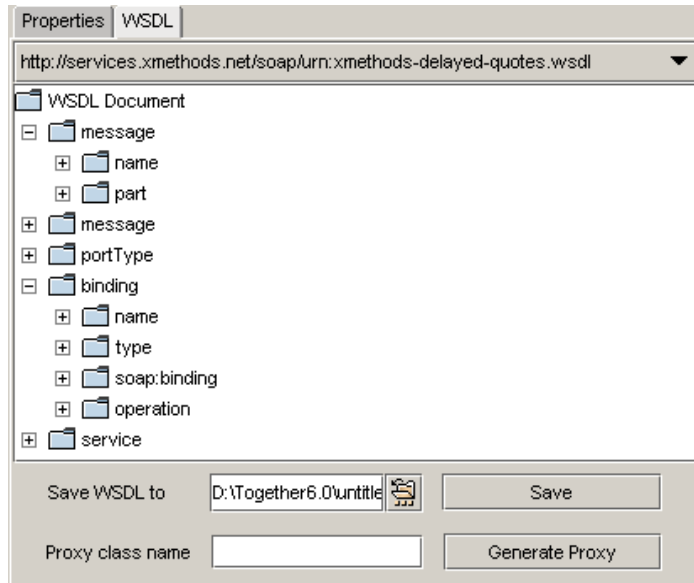
Properties of Web services

If you search in *Service names*, you get information about Web services.

You get the same information as when you search in business names and then select a specific Web service in the results.

A Web service can have a corresponding WSDL file. Go to the **WSDL** tab and choose the necessary address from the drop-down list to see the corresponding WSDL file. If the current Web service does not have a WSDL file, you see *No WSDL file*.

Figure 236



Saving WSDL files

A WSDL file contains a description of the functionality of the selected Web service.

If you want to save the WSDL file to the hard disk, use the file chooser to select the necessary path in *Save WSDL to* and click **Save**.

Generating a WSDL client

To generate a WSDL client:

1. Create a new project or open an existing one.
2. Create a new Class diagram or open an existing one and place a class shortcut on it.
3. For generating a proxy client, a corresponding WSDL file should be displayed on the WSDL subtab.
4. Specify a name for the client in the *Proxy class name* text field.
5. Click **Generate Proxy** to initiate the automatic generation procedure.

The resulting source file is stored in the opened project and a shortcut to it is placed on the active Class diagram.

Publishing Web services

Use the *Publish* tab to publish your own Web service on a Web site. The *Publish* tab is intended for working with publish registries: creating and managing your own UDDI Businesses, Service Types, and Services.

Logging on to the operator site

Before you can publish your Web service, you need to log on to the publish operator site.

To log on to the publish operator site:

1. In the UDDI Browser, choose the **Publish** tab.
2. In *Operator*, select the operator registry where you want to publish.
3. Click **Options** to open the UDDI Browser Options dialog. Set the publish operator site URL, and type in your user name and password (you need to register on the site in order to receive a user name and password).
4. Click **Log On** to launch the logon procedure. The selected publish operator registry is queried for the current user, according to the user name and password that are specified in the UDDI Browser Options.

If you are logged on successfully, you can see the list of registered objects in the left part of the window and the corresponding properties in the right part. These are your Web objects.

Click **Status** to open the Connection Status dialog, which displays the current connection status and the number of objects registered on the selected publish operator.

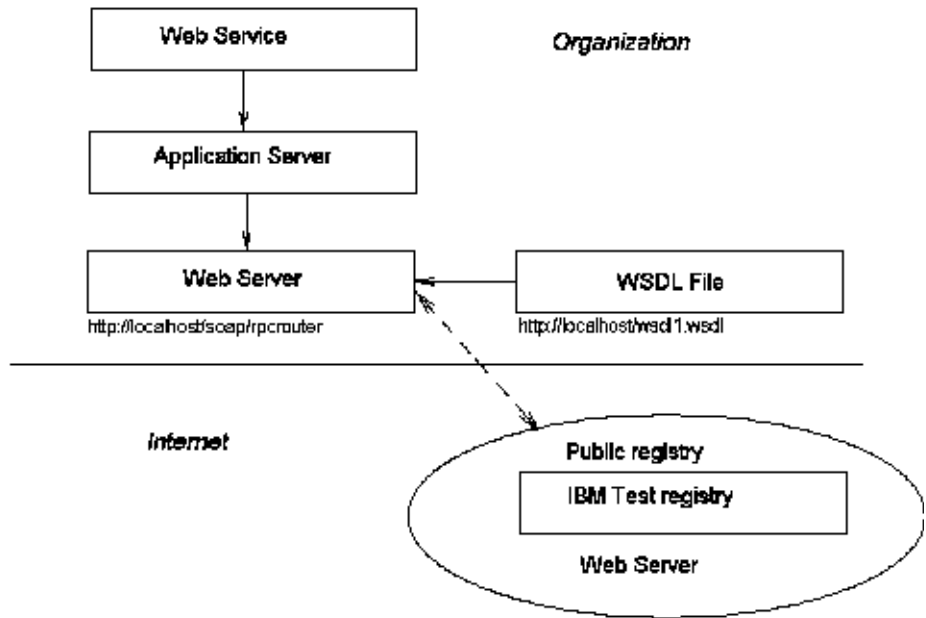
Click **Refresh** to update the list of registered objects. This button is available only after the logon procedure has been performed successfully.

Publishing Web services on the operator site

To publish your own application server on the operator site:

1. Create a new Web service (for details, see Chapter 35, “Designing and Developing Web Services”).
2. Deploy this Web service to the selected application server (for details, see “Deploying a Web Service” on page 36). An application server is connected with a Web server. You can call this Web service using the corresponding Web server and HTTP protocol. The exact address is set during deployment.
3. If you have a WSDL file describing your Web service, place this WSDL on the same Web server using Web server tools.
4. To provide access to the new Web service for other users, you need to make this Web service visible from the Internet. In other words, you have to add the Web service to the list of published Web services in the public registry on an operator’s site (this site is located on another Web server). Figure 237 illustrates the relationship of your Web service to the Internet.

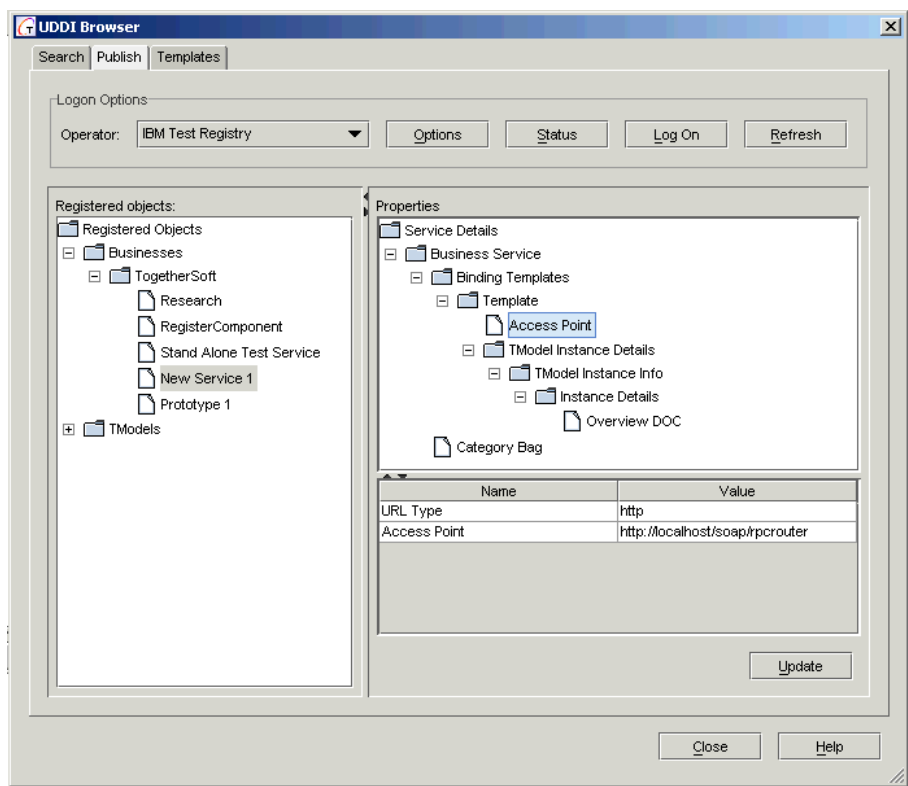
Figure 237



UDDI Browser provides the possibility to publish your own Web server on a public operator site, such as IBM Test Registry. If you are not in the list of registered users on the operator site, register yourself before you log on and publish. Define your login name and password during the registration process.

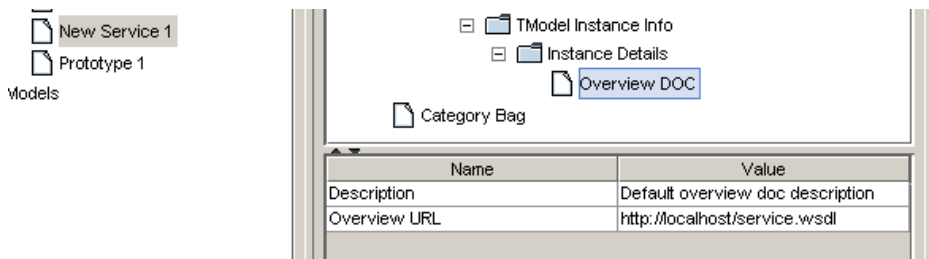
5. Log on to the site from UDDI Browser (for details, see “Logging on to the operator site” on page 37).
6. Choose the **Publish** tab. You can view the registered objects in the *Registered objects* pane.
7. Select the *Business* node and choose **New Business** on the right-click menu to create a new business.
8. Select the node for the business if you have already registered the business (for example, *TogetherSoft*) and you want to create a new service, (for example, the **TogetherSoft** node) and choose **New Service** on the right-click menu. The new Web service *NewService1* is created.
9. In the **Properties** pane, navigate to **Service Details - Business Service - Binding Templates**. Right-click on Binding Templates and choose **Add new “Template” node** from the right-click menu. Click the **Update** button.
10. In the **Properties** pane, navigate to **Service Details - Business Service - Binding Templates - Template - Access Point**. With Access Point selected, set the URL Type and the access point address.

Figure 238



11.If you want to include a WSDL file for your Web server, navigate to **Service Details - Business Service - Binding Templates - Template - TModel Instance Details - Instance Details - Overview DOC**. With Overview DOC selected, set the corresponding parameters.

Figure 239



12. Click **Update** to publish the Web service on the operator site.

NOTE: You can create more than one template for your Web service, if you are going to publish it on several operator sites. To create a new template, navigate to **Service Details - Business Service - Binding Templates** in the **Properties** pane and choose **Add new “Template” node** from the right-click menu. A new template node appears. You can add details to your service using additional nodes of the tree. You can also use existing templates for publishing (for more information, see “Creating Templates” on page 37).

Creating Templates

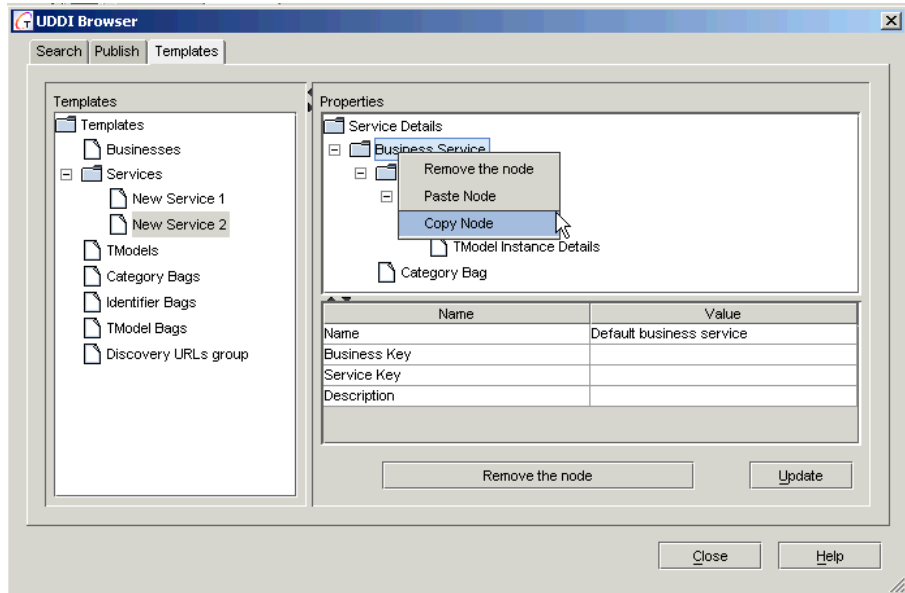
To create a new template:

1. In the UDDI Browser, choose the **Template** tab.
2. Navigate to **Templates - Services** in the Templates pane. Right-click on **Services** and choose **New service** from the right-click menu. A new service node appears.
3. Now you can create a template in the Properties pane in the same way as described in “Publishing Web services on the operator site” on page 37.

In the **Properties** pane, navigate to **Service Details - Business Service - Binding Templates**. Right-click on **Binding Templates** and choose **Add new “Template” node** from the right-click menu. Click the **Update** button. The default service with the default properties appears.

4. You can edit the default properties. If the new element is a business service, you can add new elements (templates) in it. Each time you add a new element, click **Update**.
5. If you want to use this template for publishing, navigate to the necessary node and choose **Copy Node** from the right-click menu.

Figure 240



6. Open the **Publish** page, and paste the information from the template, or replace the node on the Publish page with the node from the Templates page.
7. To remove an element, select the element and click **Remove**. Then click **Update**. The possibility to remove elements is especially useful when the number of services in the corresponding register is limited. If you want to register a new service, and there is not enough space in the list, you can delete an old one.

NOTE: You cannot remove service types from the list. When you “remove” them, they become inaccessible for searching.

PART

9

EXTENDING TOGETHER

- Chapter 38, “Advanced Customizations”
- Chapter 39, “Together Open API”

CHAPTER 38

ADVANCED CUSTOMIZATIONS

This chapter is intended for administrators or managers who need to create shared custom configurations or modify Together in order to meet corporate standards. Advanced users interested in the low-level customization capabilities of Together may also be interested in this information. Note that the most commonly-needed customizations of the configuration properties can be completed from the Options dialog. To invoke the dialog, choose Options from the Tools menu.

If you are interested in more detailed information beyond what is provided in this chapter, you can review the underlying configuration files located in the `./config` directory of the installation. Customizations that you can not configure by setting options may be possible by programming the Together API.

This chapter includes the following topics:

- “Customizing the user interface” on page 661
- “Creating custom diagram types” on page 662
- “Customizing Property Inspectors” on page 668
- “Customizing View Management Show options” on page 671

Customizing the user interface

Customizing the user interface requires editing the underlying configuration properties files located in the `./config` directory of the Together installation. Configuration files have the `.config` extension.

It may also be necessary to edit the resource files referenced by lines in the configuration files. These are located in the `./lib/i18n` directory and have the `.properties` extension. The files in this directory are of particular interest if you want to localize the Together installation, as they contain UI strings that would need to be translated to another language.

IMPORTANT: Before modifying any of the configuration or resource properties files, create backup of the original.

Creating custom diagram types

Together supports the currently-defined UML diagram types, plus Business Process, Entity Relationship, and special Together diagrams, but it does not confine you to using only these types. Together technology lets you define your own custom diagram types.

To create custom diagram types, complete the following steps:

1. Create diagram icons and icon references.
 - Create icon images to display in the Together user interface.
 - Create a folder to store the custom diagram icons and add it to the paths in the `Together.bat` file.
 - Edit the `%TGH%\config\resource.config` file to provide references to the icons of the created diagram type.
2. Create a diagram configuration file.
 - Create a new configuration file in `%TGH%\config` for each custom diagram type.
3. Define tree-view icons.
 - In the configuration file, define the diagram entity and name, and create references to the icons that will be displayed in the Explorer tree-view.
4. Define toolbar icons.
 - In the configuration file, define the diagram entity and name, and create references to the toolbar buttons.
5. Define the diagram elements for the new diagram.

The procedure is described in the following sections. You can also refer to `$TGH$/doc/guides/config/index.html`.

Step 1: Creating icons and icon references for the UI

Two icons are required for each diagram type to be represented in the Together UI: a small one for the Explorer tree-view, and a large one for the New Diagram dialog. The icon images should meet the requirements specified in Table 68.

Table 68 Specifications for graphical icon images

Icon	Pixel Dimensions	File Format
small (tree-view)	16 x 16	GIF (transparent bg)
large (dialog)	32 x 32	GIF (transparent bg)

Editing the `Together.bat` file

Add the path to the folder where the icons are stored to the `-cp` variable of the `Together.bat` file.

Example:

```
-cp "%TGH%\images;..."
```

Referencing images in the config file

To refer to the icon images of the new diagram, add the following lines to the `resource.config` file:

```
resource.element.{diagram_unique_name}.name = "{diagram_name_1}"
resource.element.{diagram_unique_name}.diagramName = "{diagram_name_2}"
resource.element.{diagram_unique_name}.icon.small = "{path_to_the_icon}"
resource.element.{diagram_unique_name}.icon.large = "{path_to_the_icon}"
```

Where:

- `diagram_name_1` is the name used for the new diagram.
- `diagram_name_2` is the name used in the New Diagram dialog box.
- `path_to_the_icon` is the path relative to the folder specified in the `-cp` variable of `Together.bat`.
- `resource.element` is a predefined construction to denote resources: icons, names, buttons.

Example:

```
resource.element.Heffalump_Trap_Diagram.name = "Heffalump Trap Diagram"
resource.element.Heffalump_Trap_Diagram.diagramName = "Heffalump Trap Diagram"
resource.element.Heffalump_Trap_Diagram.icon.small = "small_icon.gif"
resource.element.Heffalump_Trap_Diagram.icon.large = "big_icon.gif"
```

Step 2: Creating the diagram configuration file

1. Create a new file `%TGH%\config\{diagram_name}.config`.
2. Enter at least one line, which defines a new diagram type:

```
model.diagramType.{diagram_unique_ID}={diagram_unique_name}
```

Where:

- `diagram_unique_ID` is any word or number that will be used to sort diagram icons in the New Diagram dialog box. If a number is used, it should be greater than 9 because numbers from 1 to 9 are reserved for standard UML diagrams.
- `diagram_unique_name` is any word that will be used to refer to this new type of diagram.
- `model.diagramType` is a predefined construction to define the new diagram type.

Example:

```
model.diagramType.htd=Heffalump_Trap_Diagram
```

Step 3: Defining tree-view icons

First, define icons for the new diagram elements displayed in the Model tree-view.

To define the tree-view icons:

- In the diagram configuration file, add a line for each newly created diagram type:
resource.element.{element_unique_name}.icon.small = "{path_to_icon}"
Where:
element_unique_name is a word denoting the new element type.

Example:

```
resource.element.sdEntity.icon.small = "Images/tv-deepPit.gif"
resource.element.sdAttribute.icon.small = "Images/tv-honey.gif"
```

Step 4: Defining toolbar icons

Toolbar icons are dynamically constructed through their button definitions. Each button represents one element. The definition of each button has the following format:

```
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name} = \
node = createNode("{element_unique_name}")
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name}.cond
ition = "{enabling_condition}"
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name}.icon
= "{path_to_icon}"
```

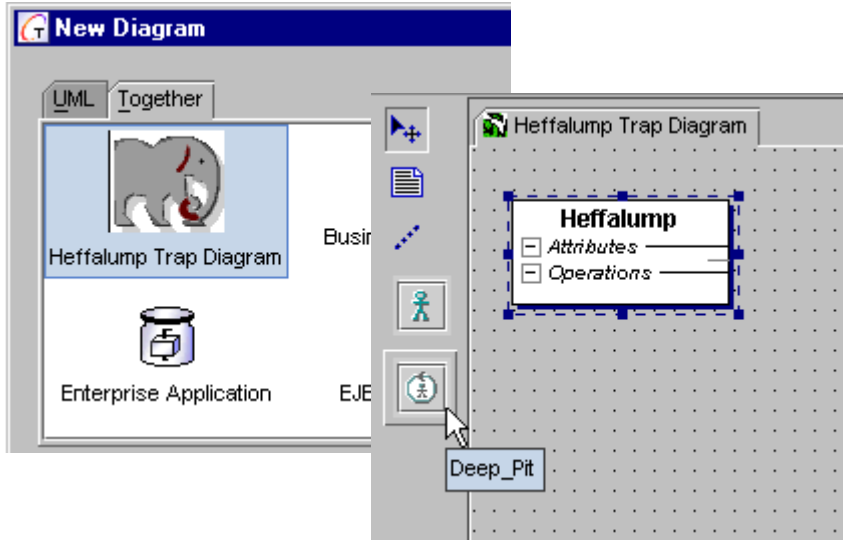
Where:

- element_toolbar_name is a unique name of a toolbar button.
- path_to_icon is the path to the button's icon.
- node is a variable that stores the new element.
- createNode is an internal function that creates a new node (entity). To create a new association (link), use the createLink function.
- enabling_condition is any valid boolean expression that defines the availability of the button.
- diagram.toolbar.button is a predefined construction to define a new button on the toolbar.

Example:

- diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit = \ node = createNode("Deep Pit")
- diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit = %DEFAULT_LANGUAGE% != "idl"
- diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit.icon = "toolbar_icon_2.gif"

Figure 241 Custom diagram and toolbar buttons



Step 5: Defining viewmaps

In order to create new elements in a diagram, it is necessary to describe the graphical presentation of each one in the `.config` files for the new diagram type. A construction similar to the one shown below must be added to the `.config` file of a new diagram type. This section presents each step based on the sample `.config` file.

You can refer to `$TGH$/doc/guides/config/Viewmaps.html` for detailed information on viewmaps.

Defining graphical presentation

Define graphical presentation starting with the header

```
view.map.*.{element_unique_name}.isTopLevel() =
```

where:

`view.map.*` is a predefined construction.

`element_unique_name` is the element name defined in the previous section.

`isTopLevel()` means that all following operations are performed if the element is on the diagram top level.

Example:

```
view.map.*.sdEntity.isTopLevel() =
```

Defining the shape of an element

Define the shape of the new element:

```
\ setLayoutConstraints( {parameter}, [parameter], ... );
```

where:

- `setGraphicObject` is a function that sets `shape_name` of an element. The predefined shapes are `Cube`, `RectangleVisible`, `Circle`, `RoundRectangle`, `Oval`, `Folder`, `Note`, `Actor`, and so on.
- `setLayoutConstraints` sets parameters of the element presentation. This function is optional.
- parameter is name-described. See the example below.

Example:

```
\ setGraphicObject("Cube");
\ setLayoutConstraints(minWidth(20),minHeight(20),
\                     preferredWidth(100),preferredHeight(100),
\                     canShiftX(true),canShiftY(true));
```

Defining the element's options

This section define options for the element's graphical presentation.

Enable display of the element's unique name

```
\ name = addCompartment("RectangleInvisible", "Name");
\ name->setLayoutConstraints(horizontalAlign("left"),
\   verticalAlign("top"),widthAlign
("parentDefined"));
```

Here register a new compartment with the name `Name` and shape `RectangleInvisible` and set its layout.

Add the element's name to the compartment

```
\ nameLabel = addToCompartment(label(getProperty("$name")), "Name");
```

Here the function `addToCompartment` adds a new label to the compartment `Name`. The function `getProperty` returns a value of `$name` for this element, which was defined in the beginning of the config file as the `uniqueName` for the element.

Enable in-place editing

This line enables in-place editing of the element's name in the label. The expression `property:= "$name"` shows the element property to be updated.

```
\ nameLabel-> setInplaceEditor({property:="$name",default:=true});
```

Define label properties

The following lines set label properties:

```
\ nameLabel->setAlignment("Center");
\ nameLabel->setLayoutConstraints(preferredHeight
(16),fixedHeight(true));
```

Define links

To enable element links to other elements, add the following line:

```
\ setCanHaveLinks()
```

See the complete sample config file below.

Sample configuration file

```
#####
# Defining the new diagram type: shape type, name, icons
#
model.diagramType.sd=SampleDiagram
resource.element.SampleDiagram.name = "Sample Diagram"
resource.element.SampleDiagram.diagramName = "Sample"
resource.element.SampleDiagram.icon.small = "Treeviews/tv-sam-
diagram.gif"
resource.element.SampleDiagram.icon.large = "DiagramTypes/
SampleDiagram.gif"
#####
# Defining icons for the new elements. These icons will be
# used in treeview
#
resource.element.sdEntity.icon.small = "Treeviews/tv-sdEntity.gif"
resource.element.sdAttribute.icon.small = "Treeviews/tv-attrib.gif"

#####
# Defining toolbar for the diagram
#
diagram.toolbar.button.SampleDiagram.Entity =
\ node = createNode("sdEntity");
\ node->setProperty("uniqueName","Entity")
diagram.toolbar.button.SampleDiagram.Entity.icon = "SampleDiagram/
entity.gif"
diagram.toolbar.button.SampleDiagram.Relationship =
\ link = createLink("sdRelationship");
\ link->setProperty("uniqueName","Relationship")
diagram.toolbar.button.SampleDiagram.Relationship.icon =
"SampleDiagram/relationship.gif"

#####
# Defining viewmap for the diagram
#
view.map.*.sdEntity.isTopLevel() =
\ setGraphicObject("Cube");
\ setLayoutConstraints(minWidth(20),minHeight(20),
\ preferredWidth(100),preferredHeight(100),
\ canShiftX(true),canShiftY(true));
\ name = addCompartment("RectangleInvisible","Name");
\ name-
```

```
>setLayoutConstraints(horizontalAlign("left"),verticalAlign("top"),
\ widthAlign("parentDefined"));
\ nameLabel = addToCompartment(label(getProperty("$name")), "Name");
\ nameLabel->setInplaceEditor({property:="$name",default:=true});
\ nameLabel->setAlignment("Center");
\ nameLabel-
>setLayoutConstraints(preferredHeight(16),fixedHeight(true));
\ setCanHaveLinks()
```

Customizing Property Inspectors

The abstract model represents properties of an object in a structured way. The Inspector content varies depending on the IDE context that contains information about the selected element. For example, if you look at the Inspector for a regular Java class and an EJB implementation class, the content displayed is quite different. The Inspector consists of several components, each representing a group of properties, their names, and values. Property Inspectors are flexible and can be customized. You can create custom tab pages, add new fields, change field names, add new stereotypes, and make other changes.

There are multiple ways to customize the Inspector, including:

- Using the Together Open API
- Using the Custom Properties module (provides convenient visual customization)
- Enabling the config-based inspector

Adding custom pages and fields to the Inspector

API-based Inspector customization

If you want to create your own page or add new properties to an existing Inspector, you can use the Inspector API. Inspector is a startup module located in %TGH%\modules\com\togethersoftware\modules\, and it does not display on the Modules tab of Together. If you need to customize the Inspector, you should edit the appropriate manifest files and classes. The updated startup module activates upon restart of Together.

Classes that enable adding new pages and fields to the Inspector are not included in the Inspector module, but reside in %TGH%\modules\com\togethersoftware\modules\inspector\examples.

To add new pages and fields to the inspector:

1. Open the appropriate manifest file (*.def) in %TGH%\modules\com\togethersoftware\modules\inspector\examples
2. Uncomment these lines:


```
MainClassName = com.togethersoftware.modules.inspector.examples.
MainClassName
Time = Startup
```
3. Create a Together project with the desired class: AddPageToInspector.java (or AddFieldsToInspector.java).

4. Create a backup copy of the class `AddPageToInspector.java` (or `AddFieldsToInspector.java`) and edit the source code, specifying the desired page and field names:

```
...
IdeInspectorProperty property;
    // Replace "myPage" with the custom page name
    property = new RwiInspectorStringProperty(rwiElements, "
taskDescription");
    // Specify custom name to be shown in the UI
    property.setName(" Task manager ");
    page.addProperty(property, null);
    // Replace "myBooleanProperty" with the custom name of a boolean
property
    property = new RwiInspectorBooleanProperty(rwiElements, " isReady ");
    // Specify custom name to be shown in the UI
    property.setName(" Is the task completed ");
    page.addProperty(property, null);
    // Replace "myStringProperty" with the custom string property name
    property = new RwiInspectorStringProperty(rwiElements, " rate ");
    // Specify custom name to be shown in the UI
    property.setName(" Select rate from the list ");
    // Specify custom names for the drop-down list
    SwingComboBoxEditor editor = new SwingComboBoxEditor(
        new DefaultComboBoxModel(
            new String[] { "rate1", "rate2", "rate3" }
        )
    );
...

```

5. Reassign the compiler destination folder to the same location as the source file. (Choose **Tools | Options | *Applicable_Level* - Builder - Built-in Javac - Compiler options: Destination directory.**)

6. Compile the class.

7. Restart Together and view the customized Inspector.

To remove additional Inspector pages:

1. Open the manifest file.
2. Comment out the `MainClassName` and `Time` lines in the manifest file.
3. Restart Together.

More API documentation

The best way to learn how to write an inspector is to study the source code of the Inspectors delivered with Together. Together's open API provides some technical documentation of the key classes and interfaces, and a commented example of how to

do a simple Inspector customization. Refer to the documentation for `com.togethersoftware.openapi.ide.inspector` through `%TGH%/doc/api/index.html`.

Customizing Inspector by means of the Custom Properties Module

It is possible to visually customize the Inspector for a particular object. This is done using the Custom Properties module. Each new property added to the selected object adds an appropriate tag to the source code. However, these changes apply to the object in reference only. Other objects of the same type are not affected.

To add or delete custom properties:

1. Make sure that the module Custom Properties is activated.
 1. Choose **Tools | Activate/Deactivate Features** on the main menu.
 2. Go to the Together Features tab.
 3. Check the *Custom Properties* box, if it is not checked.
2. Invoke the Inspector for the selected object. The Custom Properties tab appears in the Inspector.
3. On the Custom Properties tab, click the **Add** button.
4. Enter the property's name and value in the added field. Add as many properties as required.
5. Click **Remove** to delete the selected properties.
6. Press **Control+Enter** to apply the changes and close the Inspector.

User-defined inspector customization with the Inspector Property Builder

In the Options dialog, the General options group includes the *Support user-defined inspector* checkbox, which controls whether inspector pages are built from config entries, and enables visual creation of additional user-defined pages. When this option is checked, the Tools menu displays the *Customize Inspector* command, which opens the Inspector Property Builder dialog.

A new page with the specified property name appears on the Inspector for the selected elements. Together writes relevant entries to the `%TGH%/config/changes.config` file.

To customize the inspector using the Inspector Property Builder:

1. In the Options dialog, make sure that the option *Support user-defined inspector* is checked (**Tools | Options | Applicable_Level - General: Support user-defined inspector**). For *Applicable_Level*, choose *Default Level* or *Project Level*.
2. In the diagram, select the component for which the properties inspector should be customized.

TIP: When working with the Inspector Property Builder, keep in mind that:

- The dialog is non-modal. This means that you can navigate over the diagram, and every time you click on another component, the new selection displays in the *Choose shapetypes* field.

- You can select multiple components. Holding down the Ctrl key, click on the desired components. Their names are added to the *Choose shapetypes* field.

3. Choose **Tools | Customize Inspector** on the main menu.
4. In the Inspector Property Builder dialog, specify the property name, visible name, component, and type of the field.
5. Click **Add** to create a new entry. The changes apply the next time you click the component on the diagram.
6. Use the **Delete** button to remove any unnecessary entries.

For a detailed description of the controls, refer to the Inspector Property Builder section of the Context Help.

Customizing View Management *Show* options

The Options dialog provides the Show group of options under the View Management node to control the presentation of information in diagrams. These options result in configuration properties contained in the `filters.config` file and determine which elements are hidden in diagrams. By default, nothing is hidden.

At the properties level, Together essentially defines filters that remove the defined elements from view when the option is activated (hence the filename `filters.config`). The default state of the filters is off, meaning that the defined elements are not filtered (that is, they are shown). At the UI level, when a filter in the properties file is off, then the option's value ("show") is true and the option is checked in the Options dialog.

All of the Show options can be user defined, in addition to the options labeled *User-defined*. If necessary, you can change the option text, remove options from the dialog, or create additional custom filters.

Changing the display text of a Show option

You might want to change the name of the options that appear in the Options dialog (for example, to accommodate international users). By default, the names of the options are extracted from a resource file. Consider the following line from the `filters.config` file:

```
optionsEditor.item.View
Management.item.Filters.item.A.item.shortName.name = ["filters/
filter_shortName"]
```

The text in brackets (shown in blue) is a reference to a string in a resource file. To change the name that displays in the Options dialog, you should either edit it in the resource file, or replace the reference in the configuration file with a string literal in double-quotes.

To edit the resource file:

1. Open the resource file `$TOGETHER_HOME/lib/i18n/filters.properties` in a text editor.
2. Search for the name of the option (for example, "All Packages"). The search should turn up a line similar to this: `all_packages=All Packages`.

3. Edit the name as desired.
4. Save and close the properties file.

The changes take effect the next time you start Together.

TIP: Administrators with international users might want to make copies of the resource file for different languages and replace the English default file in the installations of non-English speaking users.

Removing a Show option from the Options dialog

If you do not want to use one of the predefined options and do not want it to appear in the Options dialog, comment out all lines of the option's section (for example, "All Classes") in the `filter.config` file. Update the sequential information (as described in the next section) to compensate for the removal of the commented section from the file's section sequence.

Adding a Show option to the Options dialog

Conversely, you can create a new option and display it in the Options dialog by copying any of the existing sections and modifying the lines as necessary for the view you want. Note that the sections in the properties file are arranged sequentially and contain sequential information imbedded in the lines. When you add a section, you need to modify this information throughout the section so it is the last in the sequence.

For example, the section for the All Packages option is the first section and contains the following lines:

```
filter.a = hasProperty("$physicalPackage")
...
optionsEditor.item.View Management.item.Filters.item.A.order = 10
```

The file has 12 sections. The above lines in the last section read as follows:

```
filter.l = hasProperty("$physicalPackage")
...
optionsEditor.item.View Management.item.Filters.item.L.order = 120
```

If you copy the first section and paste it at the end of the file to create a 13th section, then you need to modify the lines as follows:

```
filter.m = hasProperty("$physicalPackage")
...
optionsEditor.item.ViewManagement.item.Filters.item.M.order = 130
```

For all other lines in the section, you need to replace occurrences of `.a` with `.m` and occurrences of `.A` with `.M`. Then the sequential information in the section will be the highest in the alpha and numeric sequence.

Next, customize the filter definition in the first several lines of the section (`filter.[seq]`). The filter expression is contained in the first line of the section, such as:

```
filter.m = hasProperty("$physicalPackage")
```


The text in **bold** is the filter expression. This is usually a call to `hasProperty` or `hasPropertyValue`. Study the other filter expressions and observe their construction before coding your own filter expression.

For the remaining lines in this section, you can either use a reference to a resource:

```
filter.m.name = ["filters/my_new_filter_options"]
```

in which case you should add the property `my_new_filter` to the `filters.properties` file. Or you can use a literal instead:

```
filter.m.name = ["My New Filter Options"]
```

Finally, you can update the other lines with references to resources or literals as required.

CHAPTER 39

TOGETHER OPEN API

Together is highly extensible through an open Java API. This chapter provides an introduction to the Together Open API and explains how to write your own modules.

This chapter consists of the following topics:

- “Overview of the Together Open API” on page 674
- “Understanding modules” on page 675
- “Extending Together using modules” on page 678
- “Guidelines for developing modules” on page 680
- “Tutorial: Writing and deploying modules” on page 682
- “Accessing more API documentation” on page 691

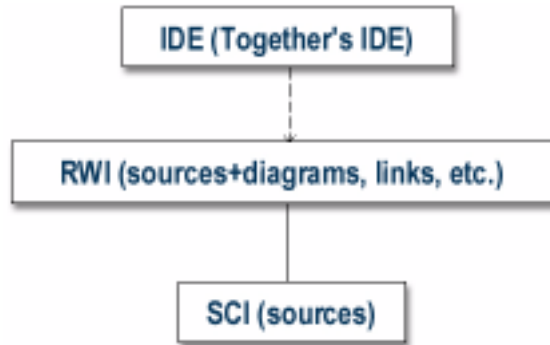
Overview of the Together Open API

The Together Open API consists of the Integrated Development Environment (IDE), Read-Write Interface (RWI), and Source Code Interface (SCI), packaged as follows:

- `com.togethersoft.openapi.ide` package and its subpackages
- `com.togethersoft.openapi.rwi` package and its subpackages
- `com.togethersoft.openapi.sci` package and its subpackages

The Together Open API is structured as a three-tier interface that allows varying degrees of access to the native infrastructure, as shown in Figure 242. The top tier represents the highest degree of constraint and the bottom tier the lowest degree of constraint.

Figure 242 Three-tier structure of the Together Open API



Following are descriptions for each component of the API:

- **IDE**

The IDE generates custom outputs based on information contained in a Together model. It is a read-only interface, meaning that you can extract information from the model but can not change the model (accidentally or otherwise). The IDE group provides the functionality related to the model's representation in Together's IDE and interaction with the user. Each package of the IDE group has a description highlighting its area of applicability.

- **RWI**

The RWI enables you to go deeper into the Together architecture. You can extract information from and write information to your models, and you can create extensions of Together's capabilities. RWI elements can represent more than packages, classes, and members. In a RWI model, elements can represent diagrams (such as class, use case, and sequence diagrams), links, notes, use cases, actors, states, and so on.

- **SCI**

The SCI provides access to the source code level. This API is the most granular, enabling manipulation of single bytes. An SCI model is a set of sources organized into packages (for Java, `.class` files are allowed). The SCI packages represent the Java packages, which can be stored in `.zip` or `.jar` files, or directories for other languages. The SCI model can also contain parts written in different languages.

The SCI allows you to work with the source code almost independently of the language being used. For example, a `SciClass` object can represent a class in both Java and C++.

Understanding modules

A *module* is a Java class that implements either `IdeScript` or the `IdeStartup` interface, or both. (For the location of these interfaces, see “Overview of the Together Open API” on page 674).

When implementing the `IdeScript` interface, you need to define the `run(IdeContext)` method; when implementing `IdeStartup` interface, you need to define the `autorun()` method.

Before you begin working with modules, it is helpful to understand the differences between modules, interfaces, and scripts within the context of Together:

- **Modules vs. Interfaces**

The `IdeStartup` interface defines a module whose `autorun()` method is invoked automatically during Together's startup process. This method performs some module-specific actions such as registering a menu command item with an appropriate listener. After it finishes executing, the `autorun()` method will not be invoked again during the current Together user session.

The `IdeScript` interface defines a module that can be invoked at any time, and any number of times during a user session by calling its `run(IdeContext)` method. An `IdeContext` instance (`com.togethersoftware.openapi.ide.IdeContext`) being passed to the `run` method contains information about selection at the moment you ran the module.

- **Modules vs. scripts**

The difference between a module and a script is mostly semantic. Both modules and scripts can use the Together API to interact with Together or access model information and process it. In Together documentation, *module* refers to a program written in Java, compiled by a Java compiler, and executed using the same Java virtual machine (JVM) as Together at runtime. *Script* refers to a scripting code that is interpreted by appropriate Together subsystems at runtime. Currently, Tcl, and JPython are supported as scripting languages, but this support may become deprecated in the future. For long-term compatibility, use Java.

Locating modules

A module defines its own subpackage with a name matching the name of the module and is located in the module package. For example:

```
package module.InsertTags;
import com.togethersoftware.openapi.ide.IdeContext;
import com.togethersoftware.openapi.ide.IdeScript;
public class InsertTags implements IdeScript{
    public void run(IdeContext context){
    }
}
```

Registering a module

To register a module, follow these steps:

- Compile a module. (For example, compile the `InsertTags` module described in the previous section.)
- Make a manifest file for your module. A manifest file is a simple text file which allows Together to identify the type of module.

The extension of a manifest file is `.def`, and the file name is the name of the module. It must be located in the classpath according to the package of the module. For example, the `InsertTags` manifest file is `New"%root%/modules/com/togethersoft/modules/inserttags/InsertTags.def`. For more information regarding `.def` files, see “Declaring a module in a `.def` file” on page 687

A manifest file consists of only one string, depending on the type of modules:

- For `IdeStartup` modules:
Startup=true
 - For `IdeScript` modules:
Script=true
- (For example, since `InsertTags` is an `IdeScript`, it contains `Script=true`.)

Starting modules

To start a module, first determine the type of module you have:

- **Startup modules implementing the `IdeStartup` interface:** You can *not* manually run these modules. Together automatically invokes these modules during the session startup.
- **Modules implementing the `IdeScript` interface:** You can use the `Modules` tab of the Explorer pane. Navigate to the script, right-click on it, and choose **Run** from the right-click menu.

For related information, see “Running modules” on page 679. For information about other module types, see “Types of modules” on page 678.

Examples for *Hello World*

Following are two examples for writing *Hello World*:

- **Startup script:** This script writes the message at Together's startup process.

```
package script.HelloWorldAutorun;
import com.togethersoft.openapi.ide.IdeStartup;
public class HelloWorldAutorun implements IdeStartup {
    public void autorun(){
        System.out.println("Hello world from HelloWorldAutorun script!!!");
    }
}
```

After compilation, compose the file `%root%/script/HelloWorldAutorun/HelloWorldAutorun.def` with the line `Startup=true` in it.

- **Script implementing the `IdeScript` interface:**

```
package script.HelloWorld;
import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
public class HelloWorld implements IdeScript {
```

```

public void run(IdeContext context){
    System.out.println("Hello World!!!");
}
}

```

After compilation, compose the file `root%\\script/HelloWorld/HelloWorld.def` with the line `Script=true` in it. Now open a project and locate this script on the Modules tab of the Explorer. Right-click on it and choose the Run command.

Extending Together using modules

Together provides an open Java API that allows you to write modules, as discussed in the previous section. Modules can use modeling information from Together or extend Together's native capabilities. Many of Together's features are implemented as modules, and the architecture makes it possible for Together to include modules (also called *plug-ins*) developed by strategic partners or other third parties.

Module development can take one of the following two forms:

- **Compiled:** Written in Java and compiled using a Java compiler; Such modules use the same JVM as Together at runtime.
- **Written in Tcl (or JPython):** Referred to as *scripts*, these modules are interpreted by appropriate Together subsystems at runtime.

Compiled modules deliver better performance and more depth, providing the full capabilities of Java.

Types of modules

A module is defined by its language, invocation time, and type of deployment. Generally, modules are categorized into the following groups:

- **User modules:** Added to the modules tree as icons and can be invoked by choosing Run from the module icon's right-click menu.
- **User with pre-initialization modules:** Similar to user modules, but the initialization method executes before the module; Assigned only to Java modules and assigned by default.
- **Startup modules:** Always run at Together startup and do not appear on the modules tree
- **Activated (Deactivated) modules:** Combine the advantages of both User and Startup types; Added both to the modules tree and Activate/Deactivate Features dialog box invoked from the Tools menu.

To change the state of an activatable module, you can use the main menu instead of browsing the modules tree. The Activate/Deactivate Features dialog box shows a list of checkboxes for all available modules. Checking a box activates a module and unchecking deactivates the module.

The state of an activatable module is persistent between Together sessions. Once it is activated, the module behaves like a startup module.

Interfaces implemented by the modules

Modules are Java classes that implement the `IdeScript`, `IdeStartup`, or `IdeActivatable` interfaces. For more general information about modules, see “Understanding modules” on page 675.

Modules of the `User` type implement the `IdeScript` interface that provides the method `run()`. Startup modules implement the interface `IdeStartup` that provides the `autorun()` method. The modules that should provide the possibility of on-demand invocation and deactivation implement the interface `IdeActivatable` that extends `IdeStartup` with the `shutdown` method. If a module is supposed to be used both as a startup and user module, it should implement `IdeScript` and `IdeStartup` interfaces, while a startup module with the possibility of on-demand invocation should implement both `IdeScript` and `IdeActivatable` interfaces.

Viewing modules

Modules are stored in the subdirectories under `$TGH$\modules\com\togethersoft / \modules`. You can navigate to the modules using the `Modules` tab of the Explorer. This tab displays several folders:

- **System** contains all the modules that are part of Together itself.
- **Sample** contains sample modules and scripts (including some source files).
- **Early access** contains modules that are works in progress — whether not fully implemented, undocumented, or both.

Table 69 shows how Together represents modules in the `Modules` tab.

Table 69 Modules Tab

Icon	Description
	Java source file for a module. If a compiler is configured, the source file can be compiled by choosing <code>Run</code> from the right-click menu.
	Compiled Java module.
	Tcl script. The <code>Run</code> command executes the script in interpreted mode.

Running modules

The modules from the `System` folder are incorporated into the Together menu system or run from dialogs. Some modules are activated through the list of `Activatable Modules` on the `Tools` menu. The others are available on the objects' right-click menus.

You can run any module (or script) from the `Modules` tab right-click menu. For modules or scripts that refer to or handle model information (as most do), you should open a Together project before running.

To run a module (or script):

1. Navigate to and select the module or script.
2. Right-click on the node and choose **Run**.

If you have defined a Java compiler in the Tools configuration options (Tools | Options | Default Level), you can also compile Java source code for a module when you choose Run from the right-click menu.

Guidelines for developing modules

Develop modules that access model information in order to:

- Generate model and code documentation in custom formats
- Export to different file formats
- Develop patterns and experts

If you create your own modules and save them to the relevant folders under `./modules/com/togethersoft/modules`, they display on the Modules tab and can run from there. Add commands to the menu system for launching your own modules by creating a Tool definition in the Options dialog (Tools | Options | Default Level | Tools), or by customizing the files `./config/menu.config` and or `./config/action.config`.

Applying naming conventions

Together modules are classes implementing either the `IdeScript` or `IdeStartup` interface, or both. The names of modules use mixed case (for example, `GenerateDocumentation`, `ImportFiles`, and so on). Table 70 provides guidelines for naming conventions.

Table 70 Naming conventions for modules

Item	Convention	Example	Benefit
Method	Methods should be named using a full English description, using mixed case with the first letter of any non-initial word capitalized. It is also common practice for the first word of a method name to be an active verb.	• <code>processModel()</code> • <code>printCurrentDiagram()</code> • <code>recheckAllNames()</code> • <code>iterateCurrentNode()</code>	Results in methods whose purpose can be determined by name, making it easier to understand and maintain the code.
Getter	Getters are methods that return the value of an attribute. Use “get” as a prefix to the name of the attribute; for a Boolean attribute, use the prefix “is”	• <code>getFirstName()</code> • <code>getAccountNumber()</code> • <code>getLostEh()</code> • <code>isPersistent()</code> • <code>isAtEnd()</code>	Indicates that a method returns an attribute of an object; or, for boolean getters, that it returns true or false.

Table 70 Naming conventions for modules

Item	Convention	Example	Benefit
Setter	Setters are methods that modify the values of an attribute. Use “set” as a prefix to the name of the attribute, regardless of the attribute type.	• setFirstName(String aName) • setAccountNumber(int anAccountNumber) • setReasonableGoals(Vector newGoals) • setPersistent(boolean isPersistent) • setAtEnd(boolean isAtEnd)	Indicates that a method sets the value of an attribute of an object.
Attribute	Use a complete English descriptor to name your attributes. Attributes that are collections such as arrays or vectors should be given names that are plural to indicate that they represent multiple values.		Indicates what the attribute represents.

Documenting the module

In the interest of brevity, use phrases instead of complete sentences when documenting a module. In particular, use brief descriptions for the initial summary and @param tag descriptions. This section provides additional guidelines for documenting modules.

Use 3rd person (descriptive) rather than 2nd person (prescriptive) as follows:

Gets the label. (preferred)

Get the label. (avoid)

Method descriptions begin with a verb phrase. A method implements an operation, so it usually starts with a verb phrase:

Gets the label of this button. (preferred)

This method gets the label of this button. (avoid)

Add a description beyond the name. The best names are “self-documenting,” meaning that they indicate what the method does. If a comment repeats the method's name in sentence form, it is not providing more information. The ideal comment goes beyond those words and should always reward you with some bit of information that was not immediately obvious from the name.

Avoid - The description below says nothing beyond what you know from reading the method name. The words “set”, “tool”, “tip”, and “text” are simply repeated in a sentence.

```
/**
 * Sets the tool tip text.
 *
 * @param text The text of the tool tip.
 */
```

```
public void setToolTipText(String text) {
```

Preferred - The following description more completely defines what a tool tip is, in the larger context of registering and being displayed in response to the cursor.

```
/**
 * Registers the text to display in a tool tip. The text
 * displays when the cursor lingers over the component.
 *
 * @param text The string to display. If the text is null,
 * the tool tip is turned off for this component.
 */
public void setToolTipText(String text) {
```

Order of Tags

Include tags in the following order:

```
* @author (classes and interfaces only, required)
* @version (classes and interfaces only, required)
* @param (methods and constructors only)
* @return (methods only)
* @exception (@throws is a synonym added in Javadoc 1.2)
* @see
* @since
* @deprecated
```

Deploying the module

A module defines its own subpackage with the name matching the name of the module, located in the modules package. For example:

```
package com.togethersoft.modules.InsertTags;
import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
public class InsertTags implements IdeScript{
    public void run(IdeContext context){
    }
}
```

Tutorial: Writing and deploying modules

This section describes the full process of writing and deploying extension modules. It includes instructions for writing a simple module and explains how to set-up Together to recognize it.

There are two major steps in module development. First, write the module source code. Second, declare the module. Confirm that module declaration and source code are consistent. For example, if a module is declared `OnDemand`, but the main class of this module implements the `IdeScript` interface, then the module will not work.

Source code for the module

The module described in this exercise will run both as a startup and as a regular Together module. Thus, you need to implement both of the following interfaces: `com.togethersoft.openapi.ide.Idescript` and `com.togethersoft.openapi.ide.Idestart`

To output messages to the Together Message pane, use the interfaces: `com.togethersoft.openapi.ide.message.IdemessageManagerAccess` and `com.togethersoft.openapi.ide.message.IdemessageType`
(Refer to the API documentation in `$TGH$/doc/api`.)

IMPORTANT: Each new module must define its own package in the `com.togethersoft.modules.InsertTags` package.

Name the module “MyFirstModule”. Following is the source code listing. To follow this exercise, create the directory

`$TGH$/modules/com/togethersoft/modules/myFirstModule`

and save the following code as

`MyFirstModule.java` : //the `MyFirstModule.java` file

```
package com.togethersoft.modules.myFirstModule;

import com.togethersoft.openapi.ide.Idescript;
import com.togethersoft.openapi.ide.Idestart;
import com.togethersoft.openapi.ide.message.IdemessageManagerAccess;
import com.togethersoft.openapi.ide.message.IdemessageType;

public void run(Idescript context){

//regular modules perform this method
IdemessageManagerAccess.printMessage(IdemessageType.INFORMATION,
"This is my first module called as a regular Together module.");
}

public void autorun(){ //startup modules perform this method
IdemessageManagerAccess.printMessage(IdemessageType.INFORMATION, "
This is my first module. It was called as a startup module.");
//this will appear in the Message pane when Together is loaded
```

```

    System.out.println("This is my first module, called as a startup
module.");

    //this will appear in the console window at startup

}

}

```

Declaring a module

The Modules tab of the Explorer does not show a module until you declare the module. The most important issue to consider when writing a module is the set of properties that will be used for module declaration. Module declaration consists of using a text manifest file to declare module type (by invocation time), name, location in the module tree, dependencies, additional libraries, and other important properties.

A manifest file is either a `Manifest.mf` file or a `.def` file. Generally, the `Manifest.mf` file applies to modules written in Java, or another language other than Tcl. Use the `.def` file if you used Tcl to write your module.

Declaring a module in a Manifest.mf file

For modules written in Java and other languages (except Tcl), use the `Manifest.mf` file to declare your module. If you used Tcl, see “Declaring a module in a `.def` file” on page 687.

Following is a list of properties available for the modules.

- **Main-Class (MainClassName):** This property is mandatory for all types of JAVA modules declared in the `Manifest.mf` file (except for JAVA modules declared in the `*.def` files). The name in brackets is the alternative property name that should be used when the module is declared in a `*.def` file.

The class specified as a value of this property should exist and should implement the Java interface that corresponds to its type (`IdeScript`, `IdeStartup`, or `IdeActivatable`).

Example of property declaration in `Manifest.mf` file:

`Main-Class: com.togethersoft.modules.ejb.EJB`

Example of property declaration in `*.def` file:

`MainClassName=com.togethersoft.modules.helloworld.HelloJava`

- **Name:** This property specifies the module name to be used in the module tree, in the Tools | Activate/Deactivate Features dialog box and in the Help | Modules submenu (for HTML modules). The value of this property is case-sensitive.

Example:

`Main-Class: com.togethersoft.modules.ejb.EJB`

Name: “EJB Support”

Or,

Name: “XPTest Support”

HelpFile: index.html

- **Folder:** This property specifies the module location in the module tree. It makes sense only for User, User with pre-initialization, and Activatable module types. If this option is not specified, the Early Access folder is used by default. The value of this property is case-sensitive.

Example:

Name: “Test Export”

Folder: “System/Test/Test Export”

As a result of the declaration above, the module Test will be found in the modules tree under All modules - System - Test, and the module node(s) will be named Test Export.

- **Time:** This property defines the module type by invocation time. It is mandatory for the JAVA modules. The value is case-sensitive.

Examples:

Time: User

Declares the module as User or User with pre-initialization (depends on the main class of the module).

Time: OnDemand

Declares the module as Activatable.

Time: Startup

Declares the module as Startup.

- **Script, Startup, OnDemand:** These properties are alternative for the Time property and are now deprecated.
- **ActivatedByDefault** This option can be used for Activatable modules only. Available values are true and false. By default, this option is false. Setting this option to true means that if an Activatable module was not explicitly deactivated, it is in the activated state.
- **Services:** This property should be used when the module can be used by some other modules. Service names (case-sensitive) should be specified as property values. Other modules will in turn specify these names as the names of the services they depend on.

Example:

Name: EJB

Time: Startup

Main-Class: com.togethersoft.modules.ejb.EJB

Services: EJB, EJBClassDiagram

This declares that services EJB and EJBClassDiagram are available and modules that depend on them will be loaded.

- **DependsOn:** This property is closely related to the previous property. It declares services that should be available to load this module. If at least one of the specified services is not available, the module does not load and Together generates an error message.

For example (module “A” depends on module “B”):

Name: |A|

Main-Class: com.togethersoft.modules.a.A

Time: User

DependsOn: B

Name: |B|

Main-Class: com.togethersoft.modules.b.B

Time: User

Services: B, ExtendedB

If module B is not deployed, module A does not load.

- **ClassPath**

This property specifies additional class libraries (jar archives or folders) that are used by this module. The value of this property can be either the fully qualified path or the relative path to this module home directory. It is also possible to use “\$TGH\$” macros.

In the following example, classes from the libraries are available for the module at runtime:

Class-Path: libInHomeFolder.jar;\$TGH\$\lib\lib.jar;C:\libs\lib1.jar

- **HelpFile**

This property adds the Help command to the module’s right-click menu in the module tree. The path to the help file must be fully qualified.

Example:

Main-Class: com.togethersoft.modules.test.export

Name: Test

HelpFile: \$TGH\$\module\com\togethersoft\modules\test\doc\index.html

- **Options**

This option adds the Options command to the module's right-click menu in the module tree. The value of this property is the name of the properties page, the same as used in the `IdeConfigManager.showConfigEditor` method.

For example:

Main-Class: com.togethersoft.modules.test.export

Time: User

Name: Test Export

Options: “Test Options”

For Activatable modules, this command displays only when the module is activated.

Known problem: only default level options can be shown this way. If an options page with the specified name does not exist, it displays an empty dialog.

- **Hidden**

Enables you to hide (true) or show (false) User or Activatable modules in the module tree.

Declaring a module in a .def file

This section includes instructions for declaring a module in a .def file. Generally, use the .def file to declare a module written in Tcl. Otherwise, declare your module in the Manifest.mf file.

Specifying whether a module is a startup module

Startup modules are modules that load when Together starts. If your module is intended to be a startup module, the manifest must contain the line:

```
Startup = true
```

For regular (i.e. non-startup) modules the manifest must contain the line:

```
Script = true
```

This should be the first line of the manifest file. Each manifest file must have a minimum of one line (either Startup = true or Script = true). You can add both lines if you want the module to run automatically on startup and you want to be able to run it during the Together session.

The manifest file can contain additional lines that provide information for Together so that the module appears in and is accessible from the Together user interface.

Defining a visible name for the module

You can specify a visible name for the module by adding this line to the manifest file:

```
Name = Visible Name
```

The string after the equal sign is displayed as the name of the module on the Modules tab of the Explorer. In this example exercise, specify a Visible Name.

Specifying the location of the Modules tab

You can specify where your module should appear on the Explorer's Modules tab. For example, you can display it in the existing Sample folder, or you can create a new folder. To specify the Modules tab location, add this line to the manifest file:

```
Folder = "<Folder name>"
```

"<Folder name>" can be anything you want. Examples:

```
Folder = "Sample" (module appears in the existing Sample folder node of the Modules tab)
```

```
Folder = "My first module" (module appears in a new folder My first module on the Modules tab)
```

Adding the module name to menus for classes, interfaces, and members

You can display the module in the Modules submenu of the right-click menu for classes, interfaces, attributes, and operations. Add the line:

```
PopupMenuItem = true
```

Adding the module name to menus of elements with specific shapetype

Display the module in the Modules submenu on the right-click menus of elements having a specific shapetype (for an explanation of this term refer to the documentation for the property `RwiProperty.SHAPE_TYPE` in the API documentation).

Briefly, by optional addition of a constraint using the shapetype parameter, you can display the module in the Modules submenu only for operations, or only for attributes, or only for classes and interfaces. Two lines are required to accomplish this:

```
PopupMenuItem = true
PopupMenuConstraints="shapeType=<shapetype identifier>"
```

Examples:

- Only for operations:

```
PopupMenuItem = true
PopupMenuConstraints = "shapeType=Operation"
```
- Only for attributes:

```
PopupMenuItem = true
PopupMenuConstraints = "shapeType=Attributes"
```
- Only for classes and interfaces:

```
PopupMenuItem = true
PopupMenuConstraints = "shapeType=Class"
```

NOTE: Since classes and interfaces have the same shapetype (i.e. Class), you can not selectively limit the display of a module to interfaces using the manifest file. Instead, refer to the module in `$TGH$\modules\com\togethersoft / \modules\tutorial` for information.

- Only for actors:

```
PopupMenuItem = true
PopupMenuConstraints = "shapeType=Actor"
```

TIP: The possible shapetypes are defined in the `RwiShapeType` interface (see the API documentation).

Rules for the manifest file

The manifest file must satisfy the following conditions:

- The manifest file name must be exactly the name of the module's main class (the class implementing `IdeStartup` or `IdeScript` interfaces).
- The manifest file location must be somewhere under the directories specified in the file `$TGH$/config/script loader.config`.

This file defines two possible root directories for manifest files:

1. `$TGH$\modules\com\togethersoft\modules`
2. `$TGH$\classes\com\togethersoft\modules`

Since `MyFirstModule` can be used as both kinds of modules (startup and runnable during session), we will use both of the module declaration lines (`Script = true, Startup = true`). Open your text editor and create the manifest file now. Add the following lines:


```
Script = true

Startup = true

Name = My First Module

Folder = "MyFirstModuleFolder"

Save the file as:
$TGH$\modules\com\togethersoftware\modules\myFirstModule\MyFirstModule.def
```

Compiling the module

Now you have to compile the module's source code. Use Together or your favorite Java compiler. Make sure to include the `$TGH$\lib\openapi.jar` file containing API classes in your compiler's classpath.

Storing the compiled class

Keep the module's `.class` file(s) in the same directory where the manifest file and sources reside. If you keep `.class` files somewhere else, make sure that their relative paths match the path of the manifest file counting from one of the following two directories:

```
$TGH$\modules\com\togethersoftware\modules
```

```
$TGH$\classes\com\togethersoftware\modules
```

For example, if your manifest file is:

```
modules\com\togethersoftware\modules\coolModule\CoolModule.def,
```

then your `.class` files must be somewhere in the classpath under the directory:
`com\togethersoftware\modules\coolModule\.`

Especially for your first modules, keep `*.class` files in the same directory as the manifest (`.def`) file and the sources. For example, you can keep them together in a separate directory under `$TGH$\modules\com\togethersoftware\modules`.

Evaluating the results

Once you have compiled the module, run Together (if you compiled the module using Together, restart). If the console window is open, you should see the module's startup message indicate when it was called at the startup and when it performed the `autorun` method.

Note that although the `autorun` method contains two output commands, only one console message appears. When Together is loading, its message pane is not visible, so all the messages written to it at the startup are displayed only after Together finishes loading.

When Together is loaded, select the Modules tab in the Explorer. Expand the All modules folder. You should see the `MyFirstModuleFolder` node with the `My First Module` files. One file is the module's source, and the other is the compiled `.class` file. Select either file, right-click, and choose **Run**. (For the source file, this command will

cause Together to try to compile and run it. To use your home-brewed compilation results, select the.class file.) If the message pane is open, you should see the module's message.

It is strongly recommended to sequentially comment out each line of the manifest file to see how this makes the module work in only one mode. Try it:

```
Script = true
;Startup = true
Name = This is my first module.
and
;Script = true
Startup = true
Name = This is my first module.
```

NOTE: Restart Together after each change in the manifest file to see your changes.

Troubleshooting your module

The declaration and .java files of the module described in this document are not provided with Together. You need to create your own first module. The process of creating, deploying and running your own module will allow you to develop future modules more efficiently.

After you successfully deploy and run this example, look through the sample modules located in the \$TGH\$\modules\com\togethersoftware\modules directory. Pay attention to the Tutorial folder, which contains a set of sample modules demonstrating the usage of Together's open API.

Try to take something from the first lesson modules in the tutorial folder and use it in a new module (or simply rename a tutorial lesson to be your second module).

If you receive compilation errors, check that you have:

- Imported all the required API interfaces/classes.
- Added all the required libraries to the compiler's classpath.
- Used API methods properly. See API documentation in \$TGH\$/doc.

Other common problems include:

- Your code compiles without errors, but you can not find your module on Together's Modules tab

Check that you created a manifest file and placed it under the directories specified by the scriptloader.config file.

By default, these are specified as:

\$TGH\$\modules\com\togethersoftware\modules

\$TGH\$\classes\com\togethersoftware\modules

- Your code compiles without errors, and the .def file is located correctly, but you still can not find your module on Together's Modules tab

- Check for spelling errors in the lines `Startup = true` or `Script = true` in the `.def` file.
- Check that the name of the manifest file is not identical to the name of the module. In such a case, Together's Message pane will display an error such as:

`"The manifest file c:\together\modules\com\togethersoft\modules / \coolModule\CoolScript.def exists but Together either cannot find or cannot read its associated module files."`
- Your code compiles without errors and you see the module's folder on the Modules tab, but there are only Java sources (assuming you want to see and run the compiled module)

 Check that the module's `.class` files are located in the right place so that Together can find them. Keep the `.class` files in the same location as the `.def` file.
- Your code compiles without errors and you see the module's compiled class on the Modules tab, but your module does not appear to function properly.

 Mark the start and finish of each of your modules (regular and startup) with appropriate messages in the Message pane. This clarifies whether your module was actually running or not. For example:


```

IdeMessageManagerAccess.printMessage
(IdeMessageType.INFORMATION, "MyCoolModule:started");//start
IdeMessageManagerAccess.printMessage
(IdeMessageType.INFORMATION, "MyCoolModule:finished");//finish
    
```

Accessing more API documentation

The information in this chapter is limited to an introduction of the Together Open API. For a more detailed technical reference, refer to the API documentation listed in: `$TGH$/doc/api/index.html`. To learn how to use the Open API documentation, refer to `$TGH$/doc/api/help-doc.html`.



LANGUAGE SUPPORT

Together ControlCenter was originally designed to support only Java fully. Now, Together ControlCenter supports six different languages:

- Java
- C++
- CORBA IDL
- VisualBasic (early access)
- C# (early access)
- VisualBasic.Net (early access)

Together also has Design-based projects, independent of any language.

The depth of support that Together offers differs among the languages. This appendix covers the variations in that support. The appendix topics are:

- “Language Support” on page 692.
- “Language-specific configuration files” on page 695.
- “Using Together with C++” on page 698.

Language Support

Live Source is the TogetherSoft technology that keeps class diagrams and source code in sync. You can turn Live Source on or off for each supported language.

TIP: Turn off Live Source support for unused languages, as using Live Source can impact system performance.

Switching languages on or off

Together ControlCenter provides Live Source for all languages. For other Together products, the license determines which languages have Live Source support.

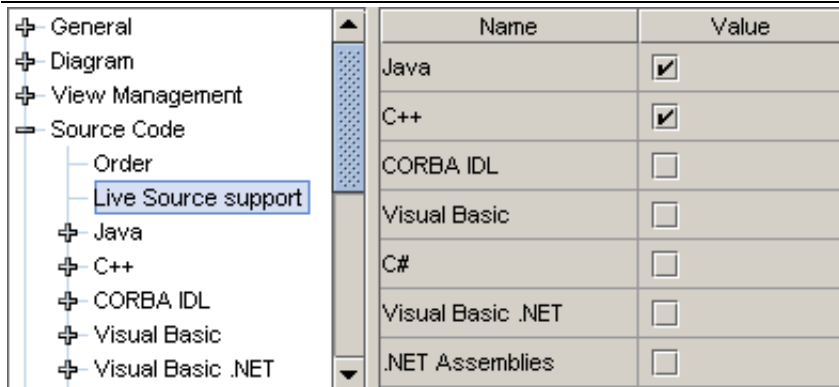
To switch Live Source for different languages on or off:

1. Choose **Tools | Options | Project** or **Tools | Options | Default** on Together's main menu.
2. Expand **Source Code** from the left pane of the dialog box.
3. Select **Live Source support**.
4. Check each language that is relevant to your work in the right pane.

The changes take effect after you restart Together.

Figure 243 is a snapshot of the panes for the Project or Default Options. This figure shows Live Source turned on for Java and C++ and disabled for other languages.

Figure 243 Language support toggles



You can optionally turn on or off language support through the file configuration file: \$TGH\$/config/languages.config.

This file contains entries in the format

```
language.language_name = yes/no
```

To disable a language, specify no in its corresponding line.

Essential features

All of the Together features work for Java. Support for other languages is limited. The limitations stem from the lack of deep parsing, the lack of object-orientation for some languages, and the inapplicability of some of the features to different languages.

- Basic functionality: provides parsing of the syntactical constructs that map directly to UML objects (classes, interfaces, methods, and so on). As of this writing, Together offers basic functionality for Java, C++, and Visual Basic. Basic functionality for C# and VB.Net is early access.

- **Deep Parsing:** functionality that handles syntactical constructs within the method bodies, initialization of variables, and so on. Deep parsing enables Together to generate sequence diagrams from methods.
- **Default templates:** textual templates for creating one-click elements on diagrams. The names of all textual templates start with `Default`. The list of currently available default templates is in the Templates folder of the Directory tab. Default templates do not appear in the Pattern Chooser.
- **Additional textual templates:** templates that are invoked from the Choose Pattern dialog. You can see the up-to-date list of available templates on the Pattern Chooser panel or in the Templates folder of the Directory tab. Textual templates in the Pattern Chooser are marked with an asterisk to distinguish them from the code-based patterns. Users can create their own custom code templates.
- **Code-based pattern:** Together patterns that are public Java classes implementing the `SciPattern` interface. The list of available patterns is on the Pattern Chooser panel. Patterns that are inapplicable to the current language are grayed out in the list. A minus sign in the table entry means that no patterns are currently available for this language. Users can develop patterns of their own.
- **Properties:** attributes with optional getter and setter methods. The properties feature originates from the Java Bean properties, where a property is represented by the property along with its getter and setter methods. Properties appear on different tabs in inspectors: Bean tab for Java, C++ properties tab for C++, Properties and Events tab for Visual Basic.

To recognize Properties:

1. Choose **Tools | Options | Project** or **Tools | Options | Default** on the main menu.
2. Expand **View Management** in the left pane of the dialog box.
3. Select **Java Beans / C++ Properties**.
4. Check the boxes in the right pane as desired to view properties.

Table 71 provides summary information on the features for the supported languages and brief notes about language-specific properties.

Table 71 Languages and feature support status

Feature	Java	C++	CORBA IDL	Visual Basic	C#	VB.Net
Basic functionality	yes	yes	yes	yes	early access	early access
Deep parsing	yes	yes	no	no	early access	early access
Default code templates	yes	yes	yes	yes	yes	yes
Additional textual templates	yes	yes	no	yes	yes	yes
Code-based patterns	yes	yes	no	no	yes	no
Properties	yes	yes	no	no	yes	no
Events	yes	no	no	no	yes	no
Syntax Highlight	yes	yes	yes	yes	yes	yes

Table 71 (continued)Languages and feature support status

Feature	Java	C++	CORBA IDL	Visual Basic	C#	VB.Net
Formatter	yes	yes	yes	yes	early access	early access
Metrics	full set	limited set	no	limited set	limited set	limited set
Audits	full set	limited set	no	no	no	no
Class diagram toolbar (language specific components)	entity, session, and message driven beans	aggregation	value type, exception	common	common	common
Documentation Generation	yes	yes	yes	yes	yes	yes
Search for usages	yes	yes	yes	only in declarations	only in declarations	only in declarations

Language-specific configuration files

Special Together configuration files enable you to tailor your development environment for different languages. These files, which reside in \$TGH\$/config, include the following:

- `cpp.config`: for C++ files.
- `csharp.config`: for C# files.
- `idl.config`: for CORBA IDL files.
- `java.config`: for Java files.
- `vb6.config`: for VB6 files
- `vbnet.config`: for VB.NET files.

You can edit these files to specify settings such as codegen and parsing options.

Language-specific inspectors

Choosing the language for a project means that any component created on the class diagram corresponds to the selected language. However, a project may contain components in different languages. The dynamic inspector tabset complies with the language of each component.

Visual Basic inspector

Visual Basic component inspectors have these special fields:

1. **Class / Interface Properties**: lists only the ones common for all languages.
2. **Class / Interface Properties and Events**: lists the properties as described in “Essential features” above.

3. Operation Properties:

- return value: Byte, Boolean, Integer, Long, Currency, Decimal, Single, Double, Date, String, Object, Variant.
- visibility: public, private, friend.
- static: checkbox for adding a static modifier to an operation.

4. Attribute Properties:

- type: Byte, Boolean, Integer, Long, Currency, Decimal, Single, Double, Date, String, Object, Variant.
- visibility: public, private.
- const: checkbox for adding a const modifier to an attribute.

C# inspector

C# component inspectors have these special fields:

1. Class Properties:

- namespace: text area for the class namespace.
- abstract: checkbox for adding an abstract modifier to the class. If the class is sealed, this field is disabled.
- sealed: checkbox for adding a sealed modifier to the class. If the class is abstract, this field is disabled.
- extends: text area and file chooser button for the base class name.
- implements: Multi-string field for the list of implemented interfaces.
- visibility: public, protected internal, protected, internal, private (visible for inner classes only).
- new: checkbox for adding a new modifier to a class (visible for inner classes only).

2. Interface Properties:

- namespace: text area for the class namespace.
- extends: text area and file chooser button for the list of inherited interfaces.
- visibility: public, protected internal, protected, internal, private (visible for inner interfaces only).
- new: checkbox for adding a new modifier to a interface (visible for inner interfaces only).

3. Operation Properties:

- return type: bool, byte, char, decimal, double, float, int, long, object, sbyte, short, string, uint, ulong, ushort, void.
- new: checkbox for adding a new modifier to an operation.
- visibility: public, protected internal, protected, internal, private.
- static: checkbox for adding a static modifier to an operation. Disabled for virtual, abstract, and override operations.

- **virtual**: checkbox for adding a virtual modifier to an operation. Disabled for static operations.
- **override**: checkbox for adding an override modifier to an operation. New, static and virtual modifiers are disabled if this property is selected.
- **abstract**: checkbox for adding an abstract modifier to an operation. Static and virtual modifiers are disabled if this property is selected.
- **extern**: checkbox for adding an extern modifier to an operation. Abstract modifiers are disabled if the property is selected.

4. Attribute Properties:

- **type**: bool, byte, char, decimal, double, float, int, long, object, sbyte, short, string, uint, ulong, ushort, void.
- **new**: checkbox for adding a new modifier to an attribute.
- **visibility**: public, protected internal, protected, internal, private.
- **static**: checkbox for adding a static modifier to an attribute.
- **readonly**: checkbox for adding a readonly modifier to an attribute.

CORBA IDL inspector

CORBA IDL inspectors have these special fields:

1. Link Properties: In addition to the common client and supplier fields are the following fields:

- **label**: text area to enter the label of the link.
- **label direction**: drop-down list of possible directions of the label (default, forward, reverse).

2. Association Class Properties:

- **client role / supplier role**: text areas to specify the appropriate roles.
- **client cardinality / supplier cardinality**: drop-down lists of possible cardinality values (zero or one, required, zero or more, one or more).
- **client qualifier / supplier qualifier**: text areas to specify the appropriate qualifiers.
- **directed**: drop-down list of the link directions (automatic, directed, undirected. Default is undirected).
- **type**: drop-down list of the link types (association, aggregation, composition).

C++ inspector

C++ inspectors have these special fields:

1. Class Properties:

- **C++ Properties**: Recognition of C++ properties is defined in the View Management page of the Options dialog.

2. Link Properties: In addition to the common client and supplier fields are two more fields:
- visibility: public, protected, private.
 - virtual: checkbox for adding a virtual modifier to the link.

Using Together with C++

There are two kinds of C++ source code files: header files and implementation files. When you place a new class node on a diagram, Together creates both a header and an implementation file.

TIP: When you click on a class node in a diagram, Together opens the header file in the Editor. When you SHIFT+click on the node, Together opens the implementation file instead.

Limitations on C++ projects

Two C++ language characteristics place limitations on C++ projects in Together: 1) fully qualified class names in C++ need not correspond to the actual physical locations of the classes; and 2) C++ uses a preprocessor.

The lack of correspondence between class names and source code files can impact memory demand. In order for Together to retrieve a class by name, all known classes must be visible when the project opens. Together must process all available files at the outset, thus increasing memory demand for C++ projects. The memory demand becomes crucial when the project uses many libraries. The size of these libraries may be significant (for example, MFC, VCL, and OWL) even when the project itself makes use of only a small portion of the libraries.

Problems with the preprocessor usage fall into two areas:

1. Macro definitions. The object model depends on resolving macro definitions. Since macros are used in course of conditional compilation, they define portions of the source code that may or may not appear in the object model.

Each macro is considered global, and thus falls into the global table. This violates C++ semantics that assume the use of macro definitions in the context of compilation unit. However, with most Together projects, you can avoid this violation by proper management of the project structure and configuration options.

2. #include directives. Together projects are folder oriented. Together processes both header files and implementation (definition) files that reside in project folders. The order in which the files are processed is arbitrary. When Together encounters an include directive, it switches to processing of that file. Together processes each file only once.

The task of creating and configuring C++ projects is important, especially for existing source code. The project must achieve a proper balance between the completeness of the model at run-time and resources used.

Defining C++ project structure

The files in the project path and in the search/classpath determine the structure of a project. Together always parses files in the project path. Together parses a file in the search/classpath only if files in the project path (directly or indirectly via a chain of include directives) `#include` it.

Setting the search path in large C++ projects

In the large-scale projects, you should add to the search path all of the sources that are considered external (standard) with respect to the entire project structure. This helps restrict the number of files handled within a project.

Together uses the standard convention to discriminate the project internal and external headers. External files are included by means of `#include <filename>` directives (with angle braces). Internal files are included via `#include "filename"` directives (with quotes).

Skipping standard includes

External include directives often contain information irrelevant for object modeling. By default, Together skips external files in order to save memory. You can switch this behavior on or off.

To change the “Skip External Files” option:

1. Choose **Options | Project** or **Options | Default** from the main menu.
2. Expand **Source Code** on the left of the resulting dialog box.
3. Expand C++.
4. Check **Skip Standards Include** on or off as desired.
5. Click **Ok** to save and quit.

Using the preinclude file

Together has a *preinclude* file, named `preinclude.inc`, to address the problems of include directives and macro definitions. Together processes `preinclude.inc` before it processes any other C++ file available to the project. It treats `preinclude.inc` almost the same as other C++ files, except that it ignores the file’s symbol declarations.

The global preinclude file that ships with Together resides in the `/lib` folder of your installation. You can create a “local project” preinclude file, also named `preinclude.inc`, in the project root. Together merges information from both global and local project preinclude files.

Managing include directives

The contents of the preinclude file consist of include directives and macro definitions. With this file, you can:

- Specify which files from an external library are necessary.
- Determine the order of processing files.

To specify which files from an external library are relevant to your project, place an include directive for each file into `preinclude.inc`. Use the following form (with quotes rather than braces):

```
#include "filename"
```

Together processes all files thus included (if found relative to search path specified) independent of the setting of the Skip Standard Includes option. Classes declared in those files become available to the project. ().

NOTE: Together binds definitions and declarations of members in the classes by signature. For seamless binding, the implementation files must be visible in the project and properly parsed. Therefore, `preinclude.inc` should contain include directives for the implementation files.

The order of include directives in `preinclude.inc` determines the order in which Together processes files. This can be critical for correct processing of project sources or sources of external libraries. The `preinclude.inc` that ships with Together has an example of this technique. This file configures the usage of Standard Template Library (`$TGH$/lib/pi_*.inc` for MS Visual C++ and Borland C++).

Managing C++ macro definitions

The `preinclude` file described in the previous section is also a C++ macro definition support file. You can use this file to manage C++ macro definitions in addition to include directives.

In order for Together to parse project files properly, it must know all the macro definitions in advance. These macros may be defined in external libraries, in internal headers, or in headers that are external to the project but still part of the source code that you want to be represented in diagrams.

If you do not include these library headers with the project resources, Together cannot parse them to perform the substitution of the macros in the headers. This can lead either to error messages or to missing diagram elements. The latter occurs if some constructs (for example, class member declarations) that are part of the object model have been defined with macros. Without macro substitutions for these constructs, Together cannot properly recognize them and display them in diagrams.

If you include *all* library sources into a Together project, the project may become unnecessarily large. This has the potential of dramatically decreasing parsing speed.

You can solve this dilemma of library headers by editing the `preinclude` file. Simply add macro definitions to `preinclude.inc` to limit which macro substitutions Together uses when parsing the project.

Writing macro definitions

The purpose of the macro definitions in `preinclude.inc` is to have Together parse the sources without error messages and display all the desired information on the diagram. You have a choice for each macro definition.

- If the constructs in the library macro do not correspond to any desired diagram elements, define the macro to nothing. Together simply skips such macro declarations.
- If some constructs in the library macro do correspond to desired diagram elements, write your macro definition to include the items that you want displayed on the

diagrams. This may mean using the original definition in the library header, or it may mean simply extracting the items that are needed for the diagram.

NOTE: Substitutions to macro definitions made in `preinclude.inc` do not affect the compilation of your sources in any way. Your compiler will use the actual library headers.

The default `preinclude.inc` file has the following examples of macro definitions. The first part is a *wrapper macro*, which is used for managing groups of macros.

```
// Used to control Microsoft C++ and MFC related macro definitions
// #define TS_PREINCLUDE_MICROSOFT_VISUAL_C
```

The second part shows the expansion options for un-commenting the wrapper:

```
#if defined(TS_PREINCLUDE_MICROSOFT_VISUAL_C)
// To show result of MFC macros expansion, uncomment
//#define MFC_MACROS_SHOW_EXPANDED

// To show MFC macros as functions, uncomment
//#define MFC_MACROS_SHOW

// Otherwise macros will be expanded to empty
```

Default library support

The default `preinclude.inc` addresses problems that can occur from parsing several popular C++ libraries. The file has `#defines` for the following libraries:

- Microsoft Windows
- Microsoft C++
- Microsoft Foundation Classes (MFC) library
- COM
- ATL
- Borland C++ version 5.0
- Borland C++ Builder

The default `preinclude.inc` contains additional wrapper macros to control which of these sets of defines are currently enabled via `#ifdef`. The wrapper macros:

- `TS_PREINCLUDE_MICROSOFT_WINDOWS`
- `TS_PREINCLUDE_MICROSOFT_COM`
- `TS_PREINCLUDE_MICROSOFT_VISUAL_C`
- `TS_PREINCLUDE_ATL`
- `TS_PREINCLUDE_BORLAND_CPP`
- `TS_PREINCLUDE_BORLAND_CPP_BUILDER`

It is impossible to anticipate which of the above you will actually use. It is also impossible to include something for every conceivable library or to include anything for your own company standard headers. Therefore, you may need to reset some C++ configuration options and change `preinclude.inc` before using Together with your C++ projects.

Setting wrapper macro options for default libraries

Some popular libraries may use macros with identical names but different definitions, making the potential for causing parsing problems. For example, MFC and OWL both define `BEGIN_MESSAGE_MAP` in completely different ways.

Before using Together for C++ projects, make sure that the wrapper macros for the libraries you use are defined in your C++ configuration options. Make sure also that the wrapper macros for the libraries you do not use are *not* defined. (If you fail to do this, some macro names may have multiple definitions. This can cause Together to display error messages while parsing C++ source code files.)

To define or un-define wrapper macros:

1. Choose **Tools | Options | Default** or **Tools | Options | Project** from the main menu.
2. Expand **Source Code** in the left pane.
3. Select **C++**.
4. Change the **Define** textfield on the right pane:
 1. Remove the names of wrapper macros for libraries you do not use from the string in the textfield.
 2. If the name of the wrapper macro for a library that you do use is not present in the string, add it to the string. Observe the uppercase naming convention. Separate macro names with semi-colons (;).
5. Click **Ok** to close the dialog.

Configuring Together for C++

You can change entries in `.config` files to customize Together to work with your C++ projects.

Configuring C++ header and implementation file extensions

The file `%TGH%/config/resource.config` contains definitions of Together file types and sets of extensions corresponding to the file types. The default file types and extensions for C++ implementation files are set by the following definitions:

```
# -- C++ --
resource.file.cpp_source.extension.1 = "cpp"
resource.file.cpp_source.extension.2 = "cc"
resource.file.cpp_source.extension.3 = "cxx"
resource.file.cpp_source.extension.4 = "c"
resource.file.cpp_source.extension.5 = "CPP"
resource.file.cpp_source.extension.6 = "CC"
resource.file.cpp_source.extension.7 = "CXX"
```

The default file types and extensions for C+ header files are set by the following definitions:

```
resource.file.cpp_header.extension.1 = "hpp"
resource.file.cpp_header.extension.2 = "h"
resource.file.cpp_header.extension.3 = "HPP"
resource.file.cpp_header.extension.4 = "H"
```

.If you add more extensions to this section of the configuration file, Together will recognize the corresponding files as C++ headers and implementation files.

Specifying file extensions for code generated by Together

By default, the file extensions for C++ code generated by Together are:

- "h" : header file extension
- "cpp": implementation file extension

You change these default values through the default or project options.

To change the file extensions for C++ code generated by Together:

1. Choose **Tools | Options | Default** or **Tools | Options | Project** from the main menu.
2. Expand **Source Code** then **C++** on the left.
3. Edit the textfield labeled **Extension of generated C++ header file**. Select a header extension listed in the file %TGH%/config/resource.config.
4. Edit the textfield labeled **Extension of generated C++ definition file**. Select a implementation extension listed in the file %TGH%/config/resource.config.
5. Click **Ok** to save and quit the dialog.

NOTE: You change an extension of source code generated by Together only to an extension listed in %TGH%/config/resource.config. Together will not accept any other extensions.

Incidentally, the file %TGH%/config/cpp.config sets the file extensions for C++ code generated by Together. The default values are defined in the following lines:

```
codegen.cpp.declaration_file_ext = "h"  
codegen.cpp.definition_file_ext = "cpp"
```

Configuring C++ compile and make utilities

Together provides no default C++ compiler. However, you can set up a compiler of your choice, following same procedure as described in configuration sections .

To configure the compile and make commands:

1. Choose **Tools | Options | Default** or **Tools | Options | Project** from the main menu.
2. Expand **Tools** on the left. This gives a tree-view of the available tools.
3. Select the tool on the left pane that you want to configure.

TIP: Do not overwrite default compiler/make settings. Use the empty "Tool #x" slots instead.

4. In the right pane, fill in the settings. For proper setup, follow the help commentary displayed in the Description section of the dialog box.
 1. **Language** drop-down list at the top: select C++.

2. **Tool X:** enter the name that you want to appear on the Together menus. Enter the text without the quotes or brackets (omit the [" and "]). For example, you can name the compile command C++ Compiler.
3. **Command:** enter the actual compile command, or browse for the compiler.
4. Set the remaining options as desired.
5. In the left pane, expand the selected tool. Then click **Show in menu**.
6. In the right pane, check all of the menus that should display the new tool.
7. Click **Ok** to finish.

NOTE: When configuring the tools, you must specify the appropriate menu command names and where these commands should be displayed. So you should already know how to execute compile and make utilities.

Known problems and recent corrections

The following sections summarize the problems and limitations of C++ in Together along with some recently made corrections.

Deep parsing

Deep parsing handles statements and variables in the method bodies and initializers. With C++, the depth of the parsing impacts the completeness of results in Update Package Dependencies, Autodependencies between classes, Show dependencies, Add Linked, Find Usages, and Generate Sequence Diagrams.

The following are recently made corrections:

- Initializers of local variables and members are now recognized. This includes:
 - using = as in: `Type x = a;`
 - using () as in: `Type x(a);`
- Lists of initializers in constructors are now processed. For example:
`ClassType(T1 param1, T2 param2): BaseType(param1), attr(param2)`
- Access of static members is now handled correctly.
- Classes used as the arguments in templates are now handled correctly. Consider this example:

```
class Example : public SomeTemplate <AClass>{
    SomeTemplate<BClass> *p;
    //...
}
```

If AClass is renamed to BClass, then the declaration for the pointer p changes to `SomeTemplate<BClass> *p;`

The remaining deep parsing problems include:

- The operations are matched to the operation calls by number of parameters rather than by types (types are ignored).

Recognizing member operation definitions

The only member definitions that Together recognizes are those that have *exactly* the same signature as the member declarations. For example, Together does not recognize the definition in the following:

```
--- header ---
```

```
class A {  
    void op( std::string str );  
}
```

```
þ--- definition ---
```

```
using namespace std;  
void A::op( string str ) {} // this definition will not be recognized
```

As a result of Together not recognizing this definition, three behaviors that you might expect are missing:

- Shift+click on op on the class diagram does not navigate to its definition.
- The definition file property in the inspector is empty.
- Editing the declaration of operation op does not affect the definition.

You can avoid this behavior by using either short names in both places (string) or full names in both places (std::string)

Processing of enumerations, typedefs and global symbols

Global enumerations, typedefs, and functions and variables are treated as follows:

- Global enumerations: Represented in class diagrams as class-like nodes labeled enum. These nodes do not support inheritance or operations. You can add a new enumerator to an enum by selecting **New | Attribute** from its right-click menu.
- Global typedefs: Represented in class diagrams as class nodes or rectangles.
 - If the typedef declaration simply creates another name for a known type, it appears on a class diagram as a rectangle with a typedef label on the top and the name below. The rectangle does not support members or inheritance.
 - If the typedef declaration defines a new class, it appears on the class diagram as an ordinary class node. If it defines a new struct, it appears on the class diagram as a class-like node with the label struct at the top.
- Global functions and variables: Currently not represented in diagrams.

Typedefs and enumerations that are defined in classes are not represented as separate nodes on class diagrams: Instead, they appear in separate compartments within the class nodes.

Documentation of C++ member definitions

C++ *Definition Doc-Comment* is an activatable feature for creating Javadoc comments of C++ class member definitions.

To activate the documentation feature:

1. Select **Tools | Activate/Deactivate Features** from Together's main menu.
2. Open the **Early Access** on the dialog box.
3. Check **C++ Definition Doc-Comment** on.
4. Click **Ok** to quit the dialog.
5. Restart Together to enable the feature.

Using the definition documentation feature

Activating the C++ Definition Doc-Comment feature adds a new **Definition documentation** tab to C++ class member inspectors. (You can access a member's inspector by selecting **Properties** from its right-click menu.) When you fill in the values for the Javadoc tags and the description, Together adds the documentation to the definition of the member (in the implementation file, unless the definition is inlined).

B

COMMANDS, MACROS, AND KEYBOARD SHORTCUTS

This appendix includes the following topics:

- “Command Line Parameters” on page 707
- “Parameters for the Together.exe Launcher” on page 710
- “System Macros” on page 711
- “Template Macros” on page 715
- “Keyboard shortcuts” on page 715

Command Line Parameters

This section provides the command line parameters for Together and includes the following topics:

- “Basic command line syntax” on page 708
- “Using the Windows launcher” on page 708
- “Invoking the Together main class” on page 709

Basic command line syntax

Table 72 lists the command line syntax for:

`Together_starter [Options] [ProjectFile]`.

Table 72 Command line syntax

Together_starter	Command that starts Together. On the Windows platform, Together has the following launchers: • <code>Together.exe</code> with disabled console output, • <code>TogetherCon.exe</code>, that uses the existing console, or opens a new console for output. • <code>Together.exe</code> provides dialogs for selecting parameters. Therefore, console version is preferable for automated doc generation. In some cases, when graphic output is advisable in the console version, you can use the option <code>-gui</code>. For other platforms, you can create <code>Together.sh</code>, <code>Together.cmd</code>, etc. depending on the operating system. Alternatively, this can be a complicated command that calls your Java VM, specifying parameters for it, followed by the Together main class name (<code>com.togethersoft.together.Main</code>) followed by any parameters for that class.
Options	One or more concordant options, starting with hyphen. If an option requires a value, you can use either “=” (equal) or “:” (colon) symbols after the option's name. For example you can type: <pre>-script=com.togethersoft.modules.helloworld.HelloJava</pre> or <pre>-script:com.togethersoft.modules.helloworld.HelloJava</pre> Note: Use the colon (:) symbol under Windows, since “=” is not recognized by the Windows command line interpreter.
ProjectFile	Path to a Together project file to be opened. The file name must have <code>.tpr</code> extension. Optional for invoking modules that do not need to access model information from a specific project.

Using the Windows launcher

The installer for Windows platforms installs the Windows-specific launcher `Together.exe`. The launcher invokes the registered Java virtual machine, prepares a command line for it, and passes any parameters in the input command that don't pertain to itself on through to the main Together class.

For complete parameters of `Together.exe` see “Parameters for the Together.exe Launcher” on page 710.

Invoking the Together main class

Under all supported operating systems, you can invoke the Together main class directly, specifying any desired or necessary parameters. The name of this class is `com.togethersoft.together.Main`. Since this can require a complex command that calls your Java VM, specifying parameters for it, followed by the Together main class name, followed by any parameters for it, you will probably prefer to create batch/command/shell script files for this kind of startup. You can find examples of such files in several platform-specific subdirectories in `$TGH$/bin`. Use these files directly, configuring as necessary for your system, or use them as models to create your own launcher files.

Table 73 lists the parameters for main class.

Table 73 Parameters of the main class.

<code><Project.tpr></code>	Specifies the fully qualified name of <i>Together</i> project file that will automatically open on start-up.
<code>-config.path:<path></code>	Tells Together to search <code><path></code> for configuration properties instead of the default hard-coded location. For information on usage see Creating a shared multi-user configuration
<code>-script: <module name></code>	Automatically runs a module after opening the specified project. If module name is specified without the full path, <i>Together</i> searches for the module in the default locations (depending on the file extension). For more information, see Working with modules . NOTE: when a script is launched, the simplified initialization of Together is performed. It means that all output is forwarded to the console, and GUI dialogs are disabled. If you wish to launch a script with full initialization of Together and GUI support, make sure to use <code>-gui</code> option in the command line.
<code>-version</code>	Types Together version and build number, and exits. *
<code>-help</code>	Types parameters of Together's main class, and exits. *
* This option is only used if console is enabled.	

Command-line examples

This section provides command line examples for the standalone formatter, documentation generation, and running modules at start-up.

Standalone formatter

You can format the source codes of your projects externally, using the Standalone Formatter. The command files for standalone formatter reside in the relevant folders under `%TGH%\bin`. For the Windows platform, the `Formatter.bat` file resides in the `win32` folder; `Formatter.sh` for the UNIX platform resides in the `unix` folder.

Run `%TGH%\bin\win32\Formatter.bat -help` to see the list of parameters.

Documentation generation

If you have a nightly or other periodic automated build process, you can update your documentation as part of it by having your process script call Together's HTML or RTF documentation generation facility through the command line interface. Together provides the possibility of generating documentation without actually opening Together window. Refer to the section Automated Doc Generation for details and examples.

Running modules at start-up

The following command line starts Together and executes the `Hello_world` module, compiling it if necessary. (Note that the `scriptloader.config` file must be configured in accordance to your Java environment). The project file name parameter is optional in this case, since the `Hello_world` module does not access project information. Note that Java files and Java classes should be specified with the proper case of letters (Java is the case-sensitive language).

```
cd %TOGETHER_HOME%\bin <Enter>
```

```
TogetherCon -script:com.togethersoft.modules.helloworld.HelloJava  
samples\java\CashSales\CashSales.tpr
```

Parameters for the Together.exe Launcher

The `Together.exe` launcher is provided for use under Win32 operating systems. Parameters of this executable file may be used when invoking it from the command prompt.

When invoking `Together.exe` from the Windows command line (or batch files), specify its parameters before any other parameters. You can see the complete list of `Together.exe` parameters by running it from the Windows Command prompt using the `-h` option.

Parameters for virtual machine preferences

Together includes the parameters `-builtin` and `-sun13`. These enable you to specify the preferred virtual machine (VM) in cases where more than one is installed. By default, the Windows launcher first looks for the Microsoft JVM and uses it to run Together. Specifying `-sun` tells the launcher to prefer the Sun JVM (assuming it is present). If the preferred VM is not found on your computer, then another one is used. If no VM is found, the launcher returns an error message. Table 74 lists valid options for the following:

```
Together.exe [options] [-c[class_name]] [parameters]
```

Table 74 Options for `Together.exe`

Option	Description
<code>-?, -h, -help</code>	Print this usage message and exit
<code>-con, -nocon</code>	Show/don't show console (default without new console)
<code>-cmd</code>	Print launched command line
<code>-nowarn</code>	Do not show warning messages

Table 74 Options for Together.exe (continued)

-full	Load all *.zip/jar from TOGETHER\lib
-profile	Display time for common operation
-verbosegc	Print when garbage collection occurs
-noclassgc	Disable class garbage collection
-ms<number>	Initial java heap size (default 64m)
-mx<number>	Maximum java heap size (default 512m)
-D<name>=<value>	Set system property (or -d:<name>=<value>)
-Xbootclasspath<path>	Set bootclasspath to <path> (JDK 1.3)
-classpath<path>	Set classpath to <path>
-cp <path>	Prepend <path> to classpath
-cp:p <path>	Prepend <path> to classpath
-cp:a <path>	Append <path> to classpath
-nojit	Disable JIT compiler
-verify, -noverify	Verify/don't verify classes when loaded
-verifyremote	Verify network loaded classes (default by VM)
-builtin	Prefer built-in Java VM (Sun JDK) to other Java VM
-sun13	Prefer Sun JDK 1.3 VM to other Java VM. Important: Together supports only JDK 1.3.
-nosystemcheck	Do not check system memory size
-c<class_name>	Class name to run (default com.togethersoft.together.Main)

The rest of command line after -c or unknown parameter prefix is class_name or its parameters. By default, JDK 1.3 (installed with Together on Windows platforms) is used. For example:

Together -sun13 -ms16m param1 param2 param3

System Macros

System macros are shorthand notations for lengthy path specifications that you can use for configuring properties, scripting or other tasks. Together knows how to expand the shorthand and make the proper reference.

Use macro references when completing customization tasks in the Tools tab of the Options dialog. Table 75 uses **bold** text to indicate references to the Tool tab.

Table 75 System macros

Macro	Description

Table 75 System macros (continued)

\$TGH\$	Contains the full path to your Together's installation. For example (c:\Together5).
\$TOGETHER_HOMES\$	Identical to \$TGH\$
\$SYSTEMJVM\$	Contains the call of the installed Java VM, along with the value of the system's CLASSPATH environment variable. For example: c:\jdk1.2\bin\java.exe -cp %CLASSPATH% (for Sun VM) c:\WINNT\jview.exe /cp %CLASSPATH% (for Microsoft VM) You can use this macro in the “Java VM” option and add additional directories/archive files after a semicolon (in Windows) or a colon (in UNIX). For example: \$SYSTEMJVM\$c:\MyClassesDir (Windows)
\$CLASSPATH\$	Contains the value of the system's CLASSPATH environment variable.
\$CLASSPATH_JVM\$	Contains the path to the rt.jar file.
\$CLASSPATH_PROJECT\$	Contains the paths to all the packages in the project.
\$SOURCEPATH\$	Contains the paths to all the writable packages in the project.
\$DESTINATION\$	Contains the value of the Destination option in the Tools tab of the Options dialog (this value is defined in the “build.destination” property located in the tool.config file)
\$MAINCLASS\$	Contains the name of a class in the project, defining the “public static void main (String[])” method. If there is no such class, this macro contains an empty line (“”).
\$LINENUMBER\$	Contains the line number of the selection in the file containing the selected element. For example, for selected operation this macro will contain the line in the file with the operation's class.
\$CLASS_NAME\$	Contains the name of the selected class (fully qualified name in Java).
\$PROJECT_DIR\$	Contains the full path to the project's directory. For example, if the project is located in the c:\together\myprojects\CoolProject directory, then this macro will contain the value "c:\together\myprojects\CoolProject"
\$PROJECT_NAME\$	Contains the name (without extension) of the project file. For example, if the project file is c:\together\myprojects\CoolProject\myproj.tpr, then this macro will contain "myproj".
\$PROJECT_EXT\$	Contains the extension of the project file. For example, if the project file is c:\together\myprojects\CoolProject\myproj.tpr, then this macro will contain "tpr".
\$PROJECT_FULLNAME\$	Contains the name (with extension) of the project file. For example, if the project file is c:\together\myprojects\CoolProject\myproj.tpr, then this macro will contain "myproj.tpr ".

Table 75 System macros (continued)

\$PROJECT_SPEC\$	Contains the full name of the project file (path, name and extension). For example, if the project file is c:\together\myprojects\CoolProject\myproj.tpr, then this macro will contain the value: "c:\together\myprojects\CoolProject\myproj.tpr".
\$FILELIST\$	Contains the "@somefile", where somefile is the name of the automatically generated file with the list of the files (each file on a new line) contained in the selection. (If a user clicks on a class, somefile will contain only the file with the selected class, but if a diagram is clicked, somefile will contain all the files that represent classes in the diagram, and all the classes in subpackages shown on this diagram, etc.)
\$FILE_DIR\$	Contains the full path to the selected file's directory. For example, if the selected file is located in the c:\together\myprojects\CoolProject directory, then this macro will contain "c:\together\myprojects\CoolProject".
\$FILE_NAME\$	Contains the name (without extension) of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "MyClass".
\$FILE_EXT\$	Contains the extension of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "java".
\$FILE_FULLNAME\$	Contains the name (with extension) of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "MyClass.java".
\$FILE_SPEC\$	Contains the full name of the selected file (path, name and extension). For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "c:\together\myprojects\CoolProject\MyClass.java".
\$DEF_FILE_DIR\$	Contains the full path to the selected definition file's directory (C++ only). For example, if the selected definition file is located in the c:\together\myprojects\CoolProject directory, then this macro will contain "c:\together\myprojects\CoolProject". Definition files has .cpp extension
\$DEF_FILE_NAME\$	Contains the name (without extension) of the selected definition file (C++ only). For example, if the selected definition file is c:\together\myprojects\CoolProject\guest.cpp, then this macro will contain "guest".
\$DEF_FILE_EXT\$	Contains the extension of the selected definition file (C++ only). For example, if the selected definition file is c:\together\myprojects\CoolProject\guest.cpp, then this macro will contain "cpp".
\$DEF_FILE_FULLNAME\$	Contains the name (with extension) of the selected definition file. For example, if the selected definition file is c:\together\myprojects\CoolProject\guest.cpp, then this macro will contain "guest.cpp".

Table 75 System macros (continued)

<code>\$DEF_FILE_SPEC\$</code>	Contains the full name of the selected definition file (path, name and extension) (C++ only). For example, if the selected file is <code>c:\together\myprojects\CoolProject\guest.cpp</code> , then this macro will contain " <code>c:\together\myprojects\CoolProject\guest.cpp</code> ".
<code>\$PROMPT\$</code>	Displays a dialog with a text input field and, when you press OK, returns the entered value. If you press CANCEL, the command is cancelled. Variants: <code>\$PROMPT:name=placeYourLabelHere\$</code> Same as <code>\$PROMPT\$</code>, but you can specify the name (a string after the equal sign) of the label above the input field. <code>\$PROMPT:name=placeYourLabelHere, default=placeDefaultValueHere\$</code> Same as above, but allows you specify the default value of the input field. If you press CANCEL, the default value will be returned. Note: If there are several <code>\$PROMPT\$</code> macros, then only one dialog will be displayed, containing fields specified in these macros, for example: <code>\$PROMPT\$PROMPT:name=MyLabel\$_someString_\$PROMPT\$</code> will display a dialog with three input fields, and after you press OK, a string will return containing : <code>AB_someString_C</code>, where A,B,C are the entered values.

Referencing configuration properties as macros

Besides using the predefined macros listed in Table 75, you can also use any of the properties defined in the `*.config` files. In this case, you need to write `$(nameoftheproperty$)`. Note that you must use a colon after the first dollar sign (`$`). If you use a property named `build.someproperty`, you can write just `$(someproperty$)`.

Following are examples:

- `$(build.classpath$)` - include the value of the property `build.classpath` (the value of this property is in the Tools tab ("Classpath") of the Options dialog)
- `$(classpath$)` - you can use short names for properties `build.*`, same as above.
- `$(vcs.option.cvs.executable$)` - include the value contained in the property `vcs.option.cvs.executable`

Template Macros

The following macros, surrounded by % characters, are used in the template values for both forward (code generation) and reverse (parser) engineering.

Table 76 Template macros

Macro	Definition	Language
%Class_Name%	The Choose Pattern dialog does not include this macro. For classes, it is substituted with the name of the class to which this template is applied (<i>such as</i> for class constructors). For members, it is substituted with the name of the class where a member is created by this template. Note: Can not be used in prologue/epilogue properties.	All languages
%FILE_NAME%	Name of the class source code file. Note: Used for generating prologue/epilogue only.	All languages
%FILE_EXT%	Extension of the class source code file. Note: Used for generating prologue/epilogue only.	All languages
%Name%	Name of a generated class / attribute / operation, editable in the Choose Pattern dialog.	All languages
%Dst%	Name of the destination class of a generated link, editable in the Choose Pattern dialog.	All languages
%Type%	Type of attribute or return type of operation, editable in the Choose Pattern dialog.	All languages
%Header_File%	The header file path.	C++
%Any%	Matches any token.	All languages

When working with the template macros, you need to consider the following:

- The case of letters in the file macros (%FILE_NAME%, etc.) controls the case of letters in the generated file name. For example, %FILE_NAME% %FILE_EXT% is expanded to CLASS1_HPP, while %File_Name% %file_ext% is expanded to Class1_hpp.
- The file macros (%FILE_NAME%, etc.) are resolved meaningfully. Thus, if due to the context they should be resolved to valid identifiers, you should control this on your own. For example, if you use .h++ file extensions, then the statement in the default file prologue: #ifndef %FILE_NAME% %FILE_EXT% can cause compilation errors.
- Code templates replace Blueprints from the pre-4.0 versions. If you are still using Blueprints, it is strongly recommended to convert to code templates, as they are more efficient.

Keyboard shortcuts

This section includes keyboard shortcuts for the follow:

- Main menu
- Diagrams

- Editor/Compiler/Debugger
- Miscellaneous shortcuts

Main Menu

Together provides a keyboard interface for frequently needed tasks. The standard keystrokes are documented in the following tables. The files `action.config` and `menu.config` in your installation store the data that controls the presentation and actions of the keyboard interface. It is possible to modify these files to customize the keyboard interface. However, such customization is recommended for advanced users only.

Table 77 Explorer shortcuts

Expand node	Right arrow, ENTER, double click
Collapse node	Left arrow, ENTER, double click

Table 78 File shortcuts

New project	CTRL+SHIFT+N
New diagram	CTRL+N
Open project	CTRL+SHIFT+O
Save	CTRL+S
Save all	CTRL+SHIFT+S
Close	CTRL+W
Close all	CTRL+SHIFT+W
Print Diagram	CTRL+P
Exit	ALT+X

Table 79 Edit shortcuts

Undo	CTRL+Z, ALT+BACK_SPACE
Redo	CTRL+Y, CTRL+SHIFT+Z
Cut	CTRL+X, SHIFT+DELETE
Copy	CTRL+C, CTRL+INSERT
Paste	CTRL+V, SHIFT+INSERT
Delete	DELETE
Go to line	CTRL+G

Table 80 Search shortcuts

Search on Diagrams	CTRL+SHIFT+D
Search / Replace in Files	CTRL+SHIFT+F
Search by Query	CTRL+SHIFT+Q
Search for Usages	CTRL+SHIFT+U

Table 81 View shortcuts

Toggle explorer pane	CTRL+ALT+Z
Toggle editor pane	CTRL+ALT+E
Toggle designer pane	CTRL+ALT+D
Toggle message pane	CTRL+ALT+M
Full screen view	F12
Next Pane	CTRL+F12
Previous Pane	CTRL+SHIFT+F12
Next Tab	ALT+RIGHT
Previous Tab	ALT+LEFT

Table 82 Help shortcuts

Contents	SHIFT+F1

Designer shortcuts

Table 83 Designer shortcuts

Deselect element	ESC
Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Invoke right-click menu of the selected element	SHIFT+Right Click
Auto layout all elements	CTRL+K
Update	F5
Paste shortcut	CTRL+SHIFT+V
Navigation between elements	UP, DOWN, LEFT, RIGHT
Select first member in the selected class	PAGE DOWN
When member is selected, select its class	PAGE UP
Diagram scrolling up/down/left/right	CTRL+UP, CTRL+DOWN, CTRL+LEFT, CTRL+RIGHT
Diagram scrolling page up/down	CTRL+PAGE UP, CTRL+PAGE DOWN
Diagram scrolling page left/right	CTRL+HOME, CTRL+END
Open Inspector	ALT+ENTER, ALT+double click
Move focus from docked inspector	ALT+ENTER
Delete	DELETE

Table 83 Designer shortcuts (continued)

Rename	F2

Table 84 Zoom shortcuts

Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Activate Zoom Lens	CTRL+SPACE
Zoom in	+
Zoom in with the toolbar zoom icon	click
Zoom out	-
Zoom out with the toolbar zoom icon	ALT + click
Fit in Window	*
Zoom 1:1	/

Table 85 Class diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 86 Class shortcuts

New attribute	CTRL+A
New operation	CTRL+O
New member by pattern	CTRL+T
New property	CTRL+B

Table 87 Package diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 88 Attribute, Operation, Property, Statechart, and Activity shortcuts

New attribute	INSERT, CTRL+A
New Operation	INSERT, CTRL+O
New Property	INSERT, CTRL+B
New internal statechart transition	CTRL+T
New internal activity transition	CTRL+T

Table 89 Toolbox shortcuts

Scroll up/down	Page Up/Page Down, Up/Down arrows
Scroll to the first button	Home
Scroll to the last button	End
Open the next tool page	SHIFT+Up
Open the previous tool page	SHIFT+Down
Scroll up/down	Page Up/Page Down, Up/Down arrows

Editor/Compiler/Debugger shortcuts

Table 90 Compile and Run/Debug shortcuts

Make	F7
Rebuild	CTRL+SHIFT+F7
Run configuration dialog	SHIFT+F10
Run application with the current parameters	CTRL+F5
Run application with the parameters dialog	CTRL+SHIFT+F5
Debug application with the current parameters	SHIFT+F9
Debug application with the parameters dialog	CTRL+SHIFT+F9
Attach to remote process	SHIFT+F5

Table 91 Editor shortcuts

Import Assistant	ALT+ENTER
Insert snippet	CTRL+J
Fast expand snippet	Space or tab
Invoke Code Sense	CTRL+SPACE
Invoke Surround with	CTRL+SHIFT+R
Insert Bookmark	CTRL+M
Invoke Bookmark Dialog	CTRL+D
Insert Numeric Bookmark	CTRL+SHIFT+number
Navigate to Numeric Bookmark	CTRL+number
Navigate to the next declaration	ALT+UP
Navigate to the previous declaration	ALT+DOWN
Navigate to the next bracket	CTRL+}
Navigate to the previous bracket	CTRL+{
Swap case of a word	CTRL+F3
Swap capitalization of the first letter	CTRL+SHIFT+F3
Comment/uncomment selection	CTRL+/
View method parameters (parameter tooltip)	CTRL+F8
Renaming (refactoring)	SHIFT+F2

NOTE: Editor keyboard shortcuts can be configured in the Text Editor Options (*Text Editor - Keyboard - <scheme>*)

Miscellaneous

Table 92 File chooser shortcuts

Invoke the right-click menu in the file chooser dialog	SHIFT+F10

Table 93 Version control shortcuts

System	CTRL+Q

Table 94 Navigation shortcuts

Switching between open panes	CTRL+F12
Switching between open panes back	CTRL+SHIFT+F12
Switching between the tabs of the selected component	ALT+ RIGHT/LEFT ARROW
Open next message (that can open)	ALT+F10
Open previous message (that can open)	ALT+F9

A

- accelerators, defining 291
- activity diagram 122, 177
 - shortcuts 718
- addToCompartment function 666
- Apache SOAP 621, 622, 638
 - deployment expert 639
 - generating proxy Web service client 635
- Apache Tomcat 621
- API
 - documentation 691
 - IDE 675
 - overview 674
 - RWI 675
 - SCI 675
- applet
 - deploying 540
 - running and debugging 538
- application client 564
 - creating “one-click” 567
- application client diagram 463, 564
 - creating 565
 - creating using Object Gallery 565
 - deploying 568
 - overview 564
 - security support 465
 - using 567
 - visual design elements 566
- array
 - evaluating 268
- assertion
 - inserting in visual script 412
- attribute 681
 - adding using the XML editor 246
 - breakpoints, setting 267
- audits and metrics 438
 - additional sources 459
 - automatic correction of audit violations 450
 - language specific, table of 452
 - output for 442
 - printing results 450
 - real-time audits 211
 - running from command line 457
 - running in Together 440
 - saving results 443

See also metrics

sets of, creating and using 451

UI Builder 299

authentication and authorization

See security

AWT 398

B

bar graph (metrics results) 447

BEA

See WebLogic

bookmarks

classifying global 236

deleting 237

editing 234

global 235

local 237

reordering 236

bootstrap loader 418

BorderLayout 282

BoxLayout 282

breakpoints 238

attribute, modifying 267

debugging 265

properties, modifying 267

setting 266

browse symbol 242

business process diagram 122

notation 191

C

C++ 698

SHIFT+Click to open files 698

C++ projects

compile and make utilities 703

file extensions 703

include directives 698

macro definitions 698

member documentation 705

preinclude file 699

Cactus 404

CardLayout 282

checkbox menu items 292

child element 246

Choose Pattern command 436

Choose Pattern dialog 424, 430, 434, 435

class diagram 122, 154

- association, showing 159
- attributes and operations, rearranging 161
- compartment controls, setting 161
- importing other diagrams 159
- inner class, defining 159
- Java Beans, showing 156
- members and properties, creating 160
- search paths, showing classes of 158
- shortcuts 718
- shortcuts, setting-up 156
- class shortcuts 718
- classes
 - UI 275
 - creating new 275
- ClearCase
 - advanced version control 322
 - interface, setting up 319
 - operation semantics 320
 - results and errors, reporting 323
 - Unified Change Management 324
- clipboard assertion 413
- CMP
 - EJB 514
- code completion feature 239
- code sense 231, 239
 - advanced 240
- code template
 - editing 426
 - properties 424
- code templates 423
 - browsing code 425
 - browsing using Directory tab 425
 - custom 427
 - custom, creating manually 429
 - custom, creating using the expert 427
 - default properties 425
 - diagram elements, creating from 431
 - displaying custom names 430
 - editing 426
 - grouping 429
 - macros, user-defined 430
 - patterns, using with 435
 - properties of 424
 - source code, formatting 427
 - using macros 425
- collaboration diagram 162
 - creating 165
 - refining 166
 - sequence diagram, converting to 166
- collection suites 407
- command line
 - parameters 707
 - syntax 708
- compile 258
- compile-make-run 258
 - alternative compiler, using 261
 - compile and make tools, executing 258
 - makefile, generating a 262
 - programs, running 261
 - standard compiler, using 259
 - using Together's 258
- component diagram 122, 179
 - contents 180
 - techniques for 179, 181
- components 56
 - menu bar 288
 - showing in container 288
 - popup menu 289
 - UI 282
 - Object Gallery 276
 - user interface 279
- config file 661, 663
 - referencing configuration properties as macros 714
 - sample 667
- configuration files 661
 - language specific 695
- configuring
 - overview of levels 79
 - See options
 - UI Builder 274
- Connector Architecture 569
- Connector Service Provider Interface
 - See Connector SPI
- Connector SPI 569
- container transaction elements 510
 - creating 511
 - definition 510
 - deploying 513
 - EJB Properties elements, creating 513

- context help
 - using 244
- Creation Scenario field 97
- customizing Together
 - advanced 661
 - toolbox 300
- CVS 306
- D**
- database modeling 112
 - activating in Together 112
- deadlock state 271
- debugging 263
 - arrays, evaluating 268
 - breakpoints 265
 - breakpoints, setting 266
 - monitors 270
 - objects, evaluating and modifying 268
 - program execution, controlling 264
 - remote debugging, starting 271
 - remote process, attaching 271
 - using the integrated debugger 263
 - variables, evaluating and modifying 268
 - watches, adding 269
- declarative security role 576
- deep parsing 694
- def file 684
- default level 79
- default target
 - setting for a test 418
- deployment
 - view in modeling 122
- deployment descriptor 461, 462, 463, 464, 465
 - CheckMustUnderstands 625
 - limitations 568
 - message 625
 - resource adapter diagram 573
 - service-urn 624
- deployment diagram 122, 182
 - techniques for 183
- Deployment Expert
 - EJB references 525
- deployment expert
 - Apache SOAP 639
 - container-transaction elements 513
 - resource adapter diagram 573
 - Web services, running for 638
 - WebLogic 642
 - WebSphere 645
- design
 - view in modeling 121
- Designer pane
 - Menu Designer view 272, 277
 - UI Designer view 272, 277
- diagram level 79
- diagrams
 - archived file, restore from 109
 - autodependency links, filtering out 131
 - bidirectional links, converting 134
 - cloning 126
 - configuring options 126
 - creating custom types 662
 - creating using main menu 125
 - default of a project 95
 - dependency links 131
 - drag-and-drop support 133
 - elements, copying and cloning 133
 - elements, drawing 127
 - elements, moving and copying 132
 - elements, selecting 132
 - graphics, improving 126
 - grid, using the 128
 - Hyperlinking feature 125
 - link to self, drawing 131
 - links, drawing 130
 - links, labeling 134
 - list of supported 122
 - multiple elements, adding 129
 - node elements and links 132
 - node elements, drawing 128
 - node elements, resizing 133
 - overview for creating 124
 - relationship links, drawing 129
 - renaming 126
 - right-click menus 132
 - See also* J2EE-supported diagrams
 - See also* real-time modeling
 - See also* UML extension diagrams
 - See also* UML modeling
 - See specific types of diagrams*
 - undo/redo 128

- disabled 248
- distribution graph (metrics results) 448
- DocGen
 - See documentation generator
- documentation generation
 - automating 330, 710
 - for user interface components 300
 - options 331
 - RTF options 329
 - starting 327
- documentation generator 334
- documentation template 334
 - body sections 335
 - calls to stock sections 336, 345
 - calls to template sections 336, 346, 362
 - controls 348
 - current model element 336
 - data controls 352
 - designer 334
 - DG functions 372
 - DG variables 369, 370
 - element iterator scopes 343
 - element iterators 336, 342
 - element property iterators 336, 344
 - enabling condition 341
 - folder sections 336, 345
 - footers 348
 - formula controls 353
 - frameset templates 358
 - headers 348
 - hypertext links 356, 363
 - image mapping 365
 - images 350
 - JavaDoc link references 366
 - labels 350
 - metamodel 336
 - panels 350
 - root object metatype 336
 - static sections 336, 348
 - stock sections 345
 - text controls 354
 - variables 353
- documentation templates
 - DG functions 372
- drag-and-drop support 133

- E
- EAI 569
- early access folder 679
- e-commerce
 - J2EE support 461
- editing 230
 - bookmarks 234
 - breakpoints 238
 - browse symbol 242
 - code completion feature 239
 - code templates 426
 - commands, edit menu 234
 - commands, editor right-click menu 234
 - configuring the editor 231
 - deleting bookmarks 237
 - editor features, advanced 239
 - Editor pane, showing and hiding 250
 - external editor 251
 - help database, creating 243
 - import assistant 241
 - inplace, how to enable 666
 - no open project 251
 - open project 251
 - parameter tooltip feature 239
 - rectangular blocks 249
 - snippets 241
 - split pane 249
 - surround with... feature 239
 - Tag Library Helper 249
 - UI classes 275
 - UI component properties 279, 280
 - using the editor 232
 - XML editor 244
 - XML, JSP, and HTML files 248
- editor
 - advanced features 239
 - override/implement methods 239
 - surround with...feature 239
 - toggle comment 239
 - watches, adding 269
- editor, built-in 230
- editor, external 251
 - configuring 251
 - using 252
- EIS 569

EJB

- assembler diagram relates to J2EE deployment expert, how 507
 - Cactus support in testing framework 404
 - code, customizing default 477
 - configuring using EJB inspectors 478
 - correction 506
 - creating a “one-click” 472
 - creating a project using existing EJB code 471
 - creating using the Object Gallery dialog 473
 - entity 472
 - local interface 475
 - message-driven 472
 - resource adapter and connector architecture 570
 - security 574
 - See also* EJB references
 - session 472
 - shortcuts, creating 504
 - Show options 473
 - verification 506
- EJB assembler diagram 462, 511, 513, 516, 518, 519, 520, 525
- and J2EE Deployment Expert 507
 - creating 468
 - key elements and properties 471
 - notation 471
 - security support 465
 - tips 508
- EJB inspector
- mandatory subtabs 478
 - optional subtabs 478
- EJB modules
- creating 467
 - creating a project using existing EJB code 471
 - creating EJBs 471
 - designing and developing 466
 - EJB assembler diagram, creating 468
 - Object Gallery, creating by 468
 - security support 576
 - visual assembly for deployment 466
 - visual design elements 469, 472
- EJB modules

See also EJB 466

- EJB Properties element 513
- EJB references
- container-transactions 513
 - creating 515
 - deploying 525
 - environment references, creating 521
 - environment variable, changing 524
 - inspector 517
 - linking between two EJBs 519
 - local, creating 517
 - resource references, creating 520
 - resource-environment references, creating 521
 - security 521
 - supported types 515
 - working with 518
- element
- options, defining 666
 - shape of, defining 665
 - unique name 666
- element, Together 46
- enterprise application
- security support 583
- enterprise application diagram 463
- and J2EE deployment expert 562
 - creating 559
 - creating using Object Gallery 559
 - importing files 562
 - shortcuts, creating 561
 - visual design elements 560
- Enterprise Application Integration
- See* EAI
- enterprise applications 558
- visual assembly 558
- Enterprise Information Systems
- See* EAI
- entity bean
- See* entity EJB
- entity bean
- See* entity EJB 481
- entity EJB 472, 481
- bean-managed, creating 491
 - code template 426
 - container-managed, creating 492

- environment references 486
- inspector 482
- local references 485
- methods 488
- primary key 483
- properties, general 483
- references 484
- relationship 490
- resource references 485
- resource-environment references 486
- security role references 486
- entity relationship diagram 122, 185
 - attributes 188
 - cardinality 190
 - entities 188
 - logical and physical view 187
 - notation 186
 - relationship links 189
- environment references 523
- event handlers 287
- event sheet diagram 206
- Explorer pane 51
 - Components tab 56
 - Copying and pasting 58
 - Diagrams tab 55
 - Directory tab 52
 - Favorites tab 54
 - finding locations 59
 - Model tab 52
 - Modules tab 56
 - opening diagrams and files 53
 - quick project access 58
 - Server tab 52
 - Test tab 57
 - toolbars 54
 - UI Builder tab 57
 - using right-click menus 58
 - XML tab 58
- export 111
 - database 112
 - DDL 112
 - IDL 114
 - JDBC supported types 112
 - XMI 116

F

- filter
 - for web application diagram 551
 - UI component properties 279
- find and replace 252
- FlowLayout 282
- focus pane 46
- frames 269
 - using 270

G

- generating documentation
 - See documentation generator
- getter 680
- go-to-line search 256
- graphics
 - improving in diagrams 126
- GridBagLayout 282
- GridLayout 282
- GUI Builder. See UI Builder

H

- Hello World example 677
- help database
 - creating 243
- HTML editor 249
- HTML files
 - editing 248
 - viewing 249
- HttpUnit 405

I

- IBM
 - See WebSphere
 - Test Registry 649
- icons
 - in menus 296
- IDE 675
- IDE Profile (UI Builder option)
 - choosing 274
 - options 274
- IdeScript interface 676
- IdeStartup interface 676
- implementation
 - view in modeling 122
- import 111
 - database 114
 - existing files to an enterprise application dia-

- gram 562
 - JDBC supported types 112
 - JUnit tests 407
 - XMI 115
- import assistant 241
- inspector 46, 63
 - EJB 478
 - EJB Environment Reference tab 523
 - EJB reference 517
 - entity EJB 482
 - UI components 279, 280
 - web application diagram 531
- inspectors 695
 - C# 696
 - C++ 697
 - CORBA IDL 697
 - test suite 418
 - Visual Basic 695
- instances, running multiple 104
- interface
 - resource managing 569
 - security, managing 570
 - sharing home and remote 479
 - transactions managing 570
- interface, definition of 676
- interface, local (for EJB) 475
- items, menu. *See* menus, designing, new menu items 291
- J**
- J2EE Module Import command 109
- J2EE support 461
 - See also* references support
- J2EE-supported diagrams 462
 - application client diagram 463
 - EJB assembler diagram 462
 - enterprise application diagram 463
 - resource adapter diagram 464
 - Taglib diagram 463
 - web application diagram 462
- JAR 564
 - enterprise application diagram 562
- JSP
 - collection, creating 545
 - creating 544
 - creating using Object Gallery 545

- debugging 546
 - deploying 547
- JSP editor 248
- JSP files
 - editing 248
- JUnit 404
 - importing tests 407
 - See also* testing framework
- JUnit tests 409
- JUnitX 404
- K**
- keyboard shortcuts 715
 - designer 717
 - editor/compiler/debugger 720
 - for the editor 231
 - main menu 716
- keyboard shortcuts, defining 291
- Kiviat graph (metrics results) 446
- L**
- launcher
 - Together.exe, invoking 710
- layout managers 281
 - BorderLayout* 281, 282
 - BoxLayout* 282
 - CardLayout* 281, 282
 - classes 281
 - FlowLayout* 281, 282
 - GridBagLayout* 281, 282
 - GridLayout* 281, 282
 - learning about 285
 - null* 282
 - OverlayLayout* 282
- listeners 552
 - creating 553
 - deploying 554
 - event types 552
- Live Source 692
- look and feel 285
- M**
- macros
 - for code templates 430
 - system 711
 - template 715
- main
 - menu 48

- window 46
- main class, Together 709
- make 258
- makefile
 - generating 262
- managers, layout 281
- manifest file
 - See Manifest.mf file, def file
- Manifest.mf file 684
- menu bar
 - creating 288
 - showing in container 288
- menus
 - accelerators 291
 - colors 297
 - component identifiers
 - renaming 298
 - designing 272, 277, 288, 297
 - new menu items 291
 - visually 290
 - fonts 297
 - keyboard shortcuts 291
 - menu bar (UI Builder) 288
 - menu items
 - adding icons 296
 - checkbox 291, 292
 - enabling, disabling 291, 295
 - moving 295
 - properties 295
 - radio-button 291, 293
 - separators 294
 - showing, hiding 291, 295
 - menu text 291
 - mnemonics 291
 - nested 294
 - popup (UI Builder) 289
- message-driven bean
 - See message-driven EJB
- message-driven EJB 472, 501
 - inspector 501
- method 680
- metrics
 - comparing results 444
 - graphs 446
 - reports 443

- saving and loading results 444
 - See also audits and metrics
 - UI Builder 299
- mnemonics, in menus 291
- model
 - existing code, create from 107
- modeling
 - deployment view 122
 - design view 121
 - implementation view 122
 - list of supported diagrams 122
 - overview 121
 - patterns 122
 - process view 122
 - See also diagrams
 - See UML modeling
 - use case view 121
- module
 - definition 675
 - testing framework 399
- modules 56
 - activating and deactivating 78
 - compiling 689
 - declaring 684
 - declaring in def 687
 - declaring in Manifest.mf 684
 - evaluating results 689
 - extending Together 678
 - guidelines for developing 680
 - Hello World example 677
 - interfaces implemented by 679
 - locating 676
 - naming conventions 680
 - registering 676
 - running 679
 - running at start-up 710
 - source code 683
 - starting 677
 - storing compiled class 689
 - tab, description of contents 679
 - troubleshooting 690
 - tutorial for writing and deploying 682
 - types of 678
 - viewing 679
- modules

- See* EJB modules 466
- monitors 270
- N**
- New Project dialog 98
- New Project Expert 96
- non-deadlock state 271
- Notation 471
- null* layout manager 282
- O**
- Object Gallery
 - EJB modules, creating 468
 - EJB, creating a 473
 - User Interface 275
- Object Gallery dialog 468
- objects
 - evaluating and modifying 268
- operator site 654
- options
 - checkboxes, value of 83
 - context-sensitive help 83
 - reference guide 83
 - resizing the Options dialog 83
 - UI Builder 274
 - viewing and editing 82
- Options dialog 672
- OverlayLayout 282
- P**
- package diagram 157
 - shortcuts 718
- package prefix
 - specifying in a project 97
- pane 46
 - expand 47
 - hiding title bars 48
 - resize 47
 - undocking 48
 - view or hide 46
- parameter function 666
- parameter tooltip feature 239
- parameters
 - command line 707
 - Together main class 709
 - virtual machine preferences 710
- pattern
 - creating resource adapter diagram 571
- patterns 431
 - and modeling 122
 - behavior 433
 - choosing for member 437
 - classes and links, creating 435
 - code templates, using with 435
 - creating 433
 - creating members by 436
 - descriptions, providing 435
 - elements, generated 432
 - languages supported 432
 - refactoring with 437
 - user interface 275
- peer element 246
- Perforce 317
- performance tuning
 - large projects 105
- popup menu
 - creating 289
 - displaying at runtime 290
- process
 - view in modeling 122
- profile
 - IDE 274
- programmatic security role 576
- project
 - creating a 95
 - creating using existing EJB code 471
 - default diagram 95
 - multiple instances, running 104
 - New Project dialog, using 98
 - New Project Expert, using 96
 - performance tuning 105
 - properties, editing 99
 - resource options, setting 103
 - resources, adding 101
 - resources, removing 102
 - root directory, primary 94
 - tips for large projects 105
 - tpr file 95
 - understanding basics 94
 - version control, setting-up 104
 - views with referenced content 105
- project level 79
- properties

- editors, UI components 280
- exposure levels, UI components 279
- of UI components 279
- properties file 661
- property inspector
 - See Inspector
- PVCS 317
- PVCS Version Manager 317
- Q**
- query-based search 254
- R**
- radio-button menu items 293
- real-time modeling 195
 - concurrent state 206
 - event sheet diagram 206
 - functional requirement 201
 - interaction diagram 208
 - maximum response time 203
 - minimum response time 203
 - object interaction group 209
 - ops simulation unit 213
 - process delay link 211
 - quality of service requirement 201
 - rate of occurrence 203
 - receiver polling reply message 210
 - sender waiting reply message 210
 - simple asynchronous messages 209
 - subsystems 197
 - system architecture diagram 203
 - system context diagrams 199
 - system requirements and architecture 196
 - time stamp message 211
 - trtelement 201
 - use case diagrams 198
- recorder 411
- recorder, for testing 398
- refactoring
 - encapsulating attributes 391
 - extracting interfaces, superclasses 392
 - extracting operations 393
 - move class, interface 388
 - patterns 437
 - pull up operation, attribute, member 388
 - push down operation, attribute, member 388
 - renaming 390

- references support
 - security support 465
- remote debugging, starting 271
- resource adapter
 - Connector Architecture 569
- resource adapter diagram 464
 - creating 570
 - creating using a pattern 571
 - deploying 573
 - security support 465
 - using 573
 - visual design elements 570
- resource adapters 569
- resource files 661
- resources
 - adding to a project 101
 - removing from a project 102
 - setting options 103
- reverse engineering 107
 - into an existing project 109
 - J2EE archive files 109
 - J2EE Module Import command 109
 - with the New Project Expert 107
- right-click menu 46
 - editor commands 234
 - in diagrams 132
- robustness analysis diagram 122, 192
 - key elements and properties 193
- roles 68, 73
 - Business Modeler 69, 74
 - Designer 69, 74
 - Developer 69, 74
 - Programmer 69, 74
- root directory
 - of a project 94
- run 261
- RWI 675
- RWI properties 337
- S**
- sample folder 679
- schemes 231
- SCI 675
- script, definition of 676
- search facilities 252
 - diagrams, searching 254

- find and replace 252
- find and replace in files 253
- go to line 256
- query-based search 254
- usages, searching for 257
- search for usages 257
- security
 - authentication and authorization 580
 - declarative security role 576
 - EJB modules 576
 - EJB references support 521
 - enterprise application 583
 - J2EE platform support 574
 - process overview 575
 - programmatic security role 576, 577
 - references support 574
 - roles 574
 - Web applications, in 579
- security constraint 579
- security role 579
- See keyboard shortcuts 715
- separators, menu 294
- sequence diagram 122, 162
 - collaboration diagram, converting to 166
 - composite pattern 169, 170
 - creating from a class 171
 - creating using the expert 165
 - message sends, creating 171
 - Options dialog 165
 - project and class, creating 171
 - refining 166
 - sequence automation, using 169
 - Singleton pattern 169
 - source code, generating 171
- servlet
 - Cactus support in testing framework 404
 - creating using Object Gallery 541
 - deploying 543
 - running and debugging 543
- servlets 540
 - creating 540
- session bean
 - See *session EJB*
- session EJB 472, 494
 - inspector 494
 - methods 497
 - properties 495
 - stateful, creating 500
 - stateless, creating 499
 - web service, using as a 501
- setGraphicObject function 666
- setLayoutConstraints function 666
- setter 681
- shortcuts
 - EJB, creating for 504
 - in the Explorer 55
 - keyboard, in menu design 291
- Show options
 - adding 672
 - display text, changing 671
 - removing 672
- simultaneous round-trip engineering 107
- snippets 241
- Source Code Control interface 318
- Star Team 318
- startup module 687
- statechart diagram 122, 173
 - entry/exit actions, specifying 175
 - internal transitions, creating 175
 - self-transition, drawing a 175
 - shortcuts 718
 - substates, nested 175
 - transition sources and targets, showing multiple 176
- stereotype
 - UML diagram support of 150
- stylesheet
 - HTML reports from testing framework 410
- submenu. See menus, nested
- Swing 398
- syntax
 - UI Builder
 - menu definition 291
- system context diagrams 199
- system folder 679
- system macros 711
- T**
- tag library
 - See taglib
- taglib 548

- Cactus support in testing framework 404
- creating 549
- Tag Library Helper 249, 550
 - using 550
- taglib diagram 463, 548
 - elements 548, 549
 - properties 549
- template
 - default properties 425
 - using macros 425
- template macros 715
- templates
 - creating using UDDI browser 657
 - See code templates
 - test cases, creating 408
- test assets, definition of 398
- test case
 - See testing framework
- test cases
 - writing 406
- test plan, definition of 398
- test results, definition of 398
- test server
 - known problems 422
 - network security 422
 - running 419, 420
 - setting-up bootstrap loader 418
 - system requirements 419
 - troubleshooting 421
- testing framework 397
 - activating 399
 - bootstrap loader 418
 - CashSales test project 399
 - collection suites 407
 - default target, setting 418
 - JUnit step options 404
 - JUnit tests, importing 407
 - JUnit tests, running 409
 - options, configuring 402
 - options, recording 404
 - overview 397
 - process description 398
 - recording options 413
 - report of test results 410
 - stubs, generating 406
 - stylesheet 410
 - templates, using to create test cases 408
 - test cases, writing 406
 - test collection, creating 410
 - test collection, running 410
 - test project, creating 400
 - test server, setting up 418
 - test suite options 403
 - test suite, creating 407
 - Tests tab 399
 - unit test builder 408
 - unit test builder options 404
 - unit tests, developing 405
 - visual script, editing 414
 - visual script, recording 411
 - visual step options 403
- Tests tab 399
- text (in menus) 291
- threads 269
 - using 270
- Together diagrams
 - See UML Extension Diagrams
- Together.exe launcher 710
- Together_starter command 708
- Tomcat 641
 - debugging JSP 547
- toolbox
 - customizing 300
 - UI Builder 279
- toolbox shortcuts 719
- toolbox, Designer 46
- tpr file 95
- troubleshooting
 - modules 690
- trtelement 201
- U
- UDDI Browser 648
 - functionality 648
 - how to use 648
 - searching for Web objects 649
 - templates, creating 657
 - Web service, publishing 653
- UI Builder 272
 - activating 273
 - adding UI components 279

- audits and metrics 299
- component properties 279, 280
- components
 - alignment 286
 - appearance order 285
 - spacing 286
- containers 282
 - adding components to 282
- deactivating 273, 274
- documentation generation 300
- editable classes
 - creating 275
 - definition of 275
- event handlers 287
- layout managers 281
- Menu Designer view 272, 277
- menu items 291
- menus
 - designing 288
- options 274
- overview 272
- tips and tricks 285
- toolbox 279, 300
- UI Designer view 272, 277
- UML extension diagrams 185
 - business process diagram 190
 - entity relationship diagram 185
 - robustness analysis diagram 192
- UML modeling
 - activity diagram 177
 - class diagram 154
 - collaboration diagram 162
 - component diagram 179
 - deployment diagram 182
 - list of supported diagrams 150
 - notation in diagrams 150
 - sequence diagram 162
 - statechart diagram 173
 - stereotypes 150
 - use case diagram 151
- Unified Change Management 324
- unit test builder 404, 408
- unit testing 397
 - See testing framework
- unit tests, developing 405
- unittestbuilderconfig.xml file 408
- Universal Description, Discovery and Integration
 - See UDDI browser
- use case
 - view in modeling 121
- use case diagram 122, 151
 - browse-through sequences 152
 - showing other diagrams 153
- user interface
 - advanced customization 661
 - designing. See UI Builder 272
 - diagram configuration file 663
 - icons and icon references, creating 662
 - modifying elements of a visual test 414
 - toolbar icons, defining 664
 - treeview icons, defining 663
 - viewmaps, defining 665
- V**
- variables
 - evaluating and modifying 268
 - structured context, displaying 268
- version control 304
 - configuring Together for 306
 - Continuous/CM support 316
 - interacting with 313
 - multi-user development support 304
 - overview 305
 - Perforce support 317
 - product-specific notes 314
 - projects, enabling for existing 312
 - projects, enabling for new 311
 - PVCS support 314
 - PVCS Version Manager support 317
 - SCC-compliant version system 309
 - setting-up for projects 104
 - Star Team support 318
 - VCS 306, 308
- view management
 - customizing Show options 671
- virtual machine 710
- visual script
 - recording 411
- visual tests 398
 - options 403
 - results 416

- running 415
 - See testing framework
- W**
- WAR**
 - enterprise application diagram 562
- watches 269
 - display format, changing 270
 - display range, changing 270
 - values, changing 270
- Web application
 - security 580
 - security support 579
- web application
 - See applet
 - See listeners
- web application diagram 462, 526
 - and J2EE Deployment Expert 556
 - container transactions, creating 536
 - creating 527
 - creating using Object Gallery 527
 - design elements 529
 - EJB properties, creating 536
 - EJB shortcuts 536
 - elements and properties 530
 - filters, creating 551
 - filters, deploying 552
 - notation 530
 - references, creating 536
 - security support 465
 - See servlets
 - tips 556
 - working with 535
- web applications
 - designing and developing 526
 - visual assembly 526
- Web resource collection 579
- Web service 621, 638
 - Apache SOAP 622
 - application servers, supported 621
 - client for Apache SOAP, generating 635
 - client for WebLogic, generating 636
 - client with WSDL file, generating 634
 - client, generating for BEA WebLogic 635
 - creating using patterns 632
 - deployment expert 638
 - deployment expert for Apache SOAP 639
 - deployment expert for WebSphere 645
 - deployment properties 638
 - properties 623
 - properties, server-specific 624
 - properties, type mapping 625
 - publishing 653
 - relationship to the Internet 654
 - run-time registering 641
 - WebLogic 629
 - WSDL file, generating 633
- web service
 - session EJB 501
- WebLogic 621, 629, 638
 - generating Web service client by pattern 636
 - message-style Web service 630
 - RPC-style Web service 629
- WebSphere 621, 632, 638
 - deployment expert 645
- Windows
 - launcher 708
- workspaces
 - and roles 68, 73
 - default workspace 67, 71
 - managing 67, 72
 - roles 68, 73
 - saving 68, 73
- WSDL 633, 648
 - generating using a Web service 633
 - subtab on Search tab 652
- WYSIWYG 272
- X**
- XML 573
- XML editor 244
 - attribute, adding 246
 - child element, adding 246
 - peer element, adding 246
 - unit test builder configuration file 409
 - XML file structure 246, 247
 - XML file, creating an 245
 - XML file, opening an 245
- Z**
- zoom shortcuts 718