



User Guide for Together[®] ControlCenter[™] and Together[®] Solo



VERSION 6.1

Borland[®] **Together[®] ControlCenter[™] and Together[®] Solo**

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249
www.borland.com

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

Copyright © 2002-2003 TogetherSoft Corporation, a wholly owned subsidiary of Borland Software Corporation. All rights reserved. All patents pending. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Other product names are trademarks or registered trademarks of their respective owners.

Updated May 15, 2003

Contents

Preface 23

Audience	23
Together Documentation Set	23
Documentation for Together Solo	24
Finding out more	26
Third-Party Books	26
On the Web	26

Getting Started 27

Introducing Together 29

Together concepts	30
Projects: creating a context	30
Modular architecture: turning features on and off	30
LiveSource: keeping code and models in sync	31
Flexible user interface: roles and workspaces	31
Roles	31
Workspaces	31
What's in the Together Installation	32

Tour of Together 33

Getting started with a sample project	33
Opening a project.	33
Opening a diagram.	35
Observing correlation between model and code	35
Activating special features	36
Navigating the Together main window	37
Arranging panes, Inspector, and Designer toolbox	37
View or hide a pane	38
Resizing or expanding panes to full screen	39
Undocking and moving panes	39
Hiding pane title bars	39
The main menu, toolbar, and status bar	40
Main menu	40
Main toolbar.	41
Status bar	42
Explorer pane	42
Directory tab	43
Server tab	44
Model tab	44
Opening diagrams and source code files.	45
Model, Favorites, and Diagram tab toolbar	46
Favorites tab	47
Diagrams tab	47
Modules tab	48

Components tab	49
Special Features tabs	49
User Interface Builder tab	49
Test tab	50
XML tab	50
Explorer pane operations.	50
Right-click menus	51
Quick project access	51
Copying and pasting	51
Finding the locations of nodes, files, and folders	52
Designer pane	52
Opening and closing diagrams	52
Designer pane toolbar	53
Designer tools	53
Editor pane	54
Message pane	55
Property Inspectors	56
Overview of Inspectors.	57
Inspector tabset	58
Properties tab.	59
Hyperlink tab	59
View tab.	59
Description tab	60
Javadoc tab	60
Requirements tab	60
Bean/C++ Properties tab	60
Object Gallery	61

Setting Your Personal Preferences 65

Setting up workspaces	65
Accessing different workspaces	65
Managing workspaces	66
Saving workspaces	67
Role-based workspaces.	67
User roles	68
Changing the configured role	68
Using view management.	69
Controlling the level of detail shown in diagrams	69
Controlling how members display in diagrams	69
Showing and hiding subpackage contents in diagrams	69
Controlling how JavaBean/C++ properties display in class diagrams	70
Showing and hiding referenced classes in dia-	

grams	70	Project file	96
Showing dependencies between classes and interfaces.	70	The <default> diagram	96
Controlling the display of sequence diagrams	70	Creating new projects	97
Banning destinations in class diagrams	71	Preparing to create a project	97
Showing or hiding aggregations of diagram and EJB elements	71	New projects without existing code	97
		New projects with existing code	97
		Using the New Project Expert	98
		Using the New Project dialog	100
Configuring Options 73		Editing project properties	101
Activating and deactivating modules	73	Adding Resources	103
Overview of configuration levels	74	Removing Resources	105
Setting options in default mode	74	Setting Resource Options.	105
Setting options in advanced mode.	76	Creating a new JDK configuration	106
Viewing and editing options	78	Running multiple project instances.	107
Values indicated by checkboxes.	79	Setting-up version control for projects	108
Accessing context-sensitive help for nodes and options	79	Setting up large projects	108
Resizing the Options dialog	79	Creating views with referenced content	109
Encoding of the configuration options	79	Performance tuning.	109
Reference guide to options	80	Reverse Engineering 111	
Default and project level options	81	Reverse engineering techniques.	111
Diagram level options	87	Creating a new project with existing code	111
Common Configuration Tasks	87	Incorporating code into an existing project. . . .	113
How to make Association links display a directional arrow	88	Recognizing links during reverse engineering	114
How to change the default source file header for the generated code	88	Reverse engineering archive files	115
How to customize the default settings for C++	88	Importing and Exporting Information 117	
How to create and use custom snippets for source code and text	89	Overview of import and export operations	117
How to configure Stereotypes.	89	Importing and exporting JDBC database information	118
How to customize Inspector properties	89	Activating database import and export.	118
How to hide and show elements	89	Generating DDL (exporting to DDL)	119
How to set options to control the formatting of your source code.	89	Importing database meta-information	120
How to enable mouse-wheel support (Windows only).	90	IDL Export and LiveSource	121
How to set up Together and projects to interact with version control	90	Exporting IDL from class diagrams	121
Configuring Together for improved performance	91	LiveSource support for IDL	122
		Importing and exporting XMI models	122
		Importing XMI models	122
		Exporting XMI models	123
		User configurable translations	124
		Generating makefiles 125	
		Configuring the makefile destination.	125
		Generating a makefile	126
Creating and Managing Projects 93		Modeling with Together 127	
Working with Projects 95		Introduction to Modeling 129	
Understanding Project Basics.	95	Overview of modeling	129
Primary root directory.	96		

Diagrams supported in Together	130
Working with Diagrams 133	
Creating diagrams in projects	133
Using the main menu or toolbar to create diagrams.	134
Using the hyperlink feature to create diagrams.	135
Cloning diagrams.	135
Renaming diagrams	135
Configuring diagram options	136
Options for improving graphics	136
3D look	136
Anti-aliasing	136
Drawing diagram elements.	136
Undo/Redo	137
Using the grid.	137
Drawing elements	138
Configuring element symbol size.	138
Adding multiple elements	139
Drawing relationship links	140
Drawing links with bending points.	141
Drawing a link to self.	142
Drawing dependency links	142
Filtering out autodependency links.	142
Placing shortcuts on diagrams	143
Working with elements and links	143
Selecting elements	143
Right-click menus	144
Drag-drop moving and copying elements.	144
Full drag-and-drop support	145
Copying and cloning elements	145
Resizing node elements	145
Converting bidirectional links	146
Labeling links	147
Changing links	147
Viewing diagrams and managing diagram layout.	148
Viewing	148
Zooming.	149
Using the automated layout features	149
Creating your own manual layout	149
Tips and tricks.	150
Editing diagrams.	150
Opening and closing diagrams	150
Editing properties	151
Opening the Inspector	152
Modifying properties	152
Applying changes	152
In-place editing	152
Hyperlinking diagrams	153
Why use hyperlinking?.	154
How to create hyperlinks.	154
Hyperlinking to a new diagram	155
Hyperlinking to an existing diagram or diagram element.	155
Hyperlinking to a URL or file	155
Viewing hyperlinks	156
Browsing hyperlinked resources	156
Removing hyperlinks	156
Annotating diagrams.	157
Using Notes	157
Note Links	157
Inspector documentation tabs	158
Standalone design elements	158
Creating standalone design elements.	158
Using standalone design elements	159
Saving and copying diagram images	159
Copy - paste within Together	159
Copy image	160
Save image	160
Printing diagrams and source code	160
Setting Print options	160
How to print diagrams	161
How to print text	161
Using auto-layout for printing.	161
Printing generated documentation	162
Troubleshooting	162
Tips and tricks	162
UML Modeling 165	
Notation and stereotypes of UML diagrams.	165
Use case diagrams	166
Diagram elements.	166
Creating browse-through sequences of use case diagrams	167
Adding extension points	168
Class diagrams	168
Diagram elements	169
Physical (package) and logical class diagrams.	170
Working with logical class diagrams	170
Working with package diagrams	171
Opening packages	171

Deleting packages	171	Statechart diagrams	184
Renaming packages	171	Diagram elements	185
Viewing packages and their content . . .	171	Drawing a self-transition	185
Moving and copying elements from packages	171	Creating internal transitions	186
Showing classes on search paths	172	Specifying entry/exit actions for a state . .	186
Showing JavaBeans	172	Creating nested substates	186
Defining inner classes	172	Showing multiple transition sources or targets .	187
Creating associations, aggregations, and compo-		Modeling complex states	187
sitions	173	Activity diagrams	188
Editing members and properties	174	Diagram elements	188
Adding and editing members	174	Working with activity diagrams	189
Adding and editing properties	174	Component diagrams	190
Rearranging the order of attributes or opera-		Working with component diagrams	191
tions	175	Deployment diagrams	192
Setting compartment controls	175	Diagram elements	192
Sequence and collaboration diagrams	175	Working with deployment diagrams	193
Diagram elements	177	UML Extension Diagrams 195	
Converting between sequence and collaboration		Entity relationship diagrams	195
diagrams	178	Notation	196
Creating sequence and collaboration diagrams .		Logical and physical diagram view	197
178		Diagram elements	198
Generating sequence diagrams from opera-		Entities	198
tions	178	Attributes	198
Setting options to control generation of se-		Relationship links	199
quence diagrams	179	Business process diagrams	200
Specifying classes for objects	179	Diagram elements	200
Working with messages	180	Notation	201
Messages in collaboration diagrams . . .	180	Robustness analysis diagrams	201
Messages in sequence diagrams	180	Diagram elements	202
Creating a message-to-self	180	Key elements and properties	203
Creating a message link that corresponds		Real-Time Modeling 205	
to an operation call	180	Introduction to Together Real-Time	205
Changing activation times	181	Capabilities of the real-time feature	205
Reordering message links	181	Getting started	206
Branching messages	181	Real-time development process	206
Annotating message links	182	Gathering system requirements	207
Nesting messages	182	Deciding system architecture	207
Creating statement blocks	183	Analyzing subsystems	207
Working with lifelines	183	Designing subsystems	207
Adjusting the default lifeline of objects	183	Implementing subsystems	207
Adjusting the size of object lifelines . .	183	Special real-time diagrams	207
Changing the order of objects in a se-		Use case diagrams	208
quence diagram	183	Use case stereotypes	209
Marking a destroyed object with an "X" .		Activity	209
183		Exception	209
Generating implementation source code . . .	184		

Use case links	210	Format of the XML structure diagram	232
Extend	210	Working with DTD-specific components	233
Actors.	210	XML modeling step by step	234
System context diagrams	210	Creating the XML structure diagram	234
System context diagram elements	210	Creating elements, links, and attributes.	234
Real-time inspector properties	211	Creating and using complex types and data	
System requirements.	211	types.	236
Creating a new system requirement	212	Creating re-usable (global) attributes	237
Properties of functional requirements	213	Creating groups	238
Properties of quality of service require-		Import and export operations	238
ments	214	DTD/XSD Import-Export	238
System architecture diagrams.	214	XML Serialization.	239
Subsystems	215		
Subsystem Properties.	216		
Responsibilities	216		
Statechart diagrams	217		
Event sheet diagrams	217		
Interaction diagrams.	219		
Interaction diagram elements	220		
Object groups	220		
Asynchronous messages.	220		
simple asynchronous	220		
sender waiting reply	221		
receiver polling reply	221		
time stamp	222		
process delay link	222		
Real-time audits	222		
Simulation	223		
Preparing a simulation.	224		
Event properties	224		
The ops simulation unit	224		
Rates of occurrence	224		
Interprocess messages	225		
Preemption	225		
Priority.	225		
Running a Simulation	225		
Simulation controls.	226		
Simulation chart settings.	226		
XML export	227		
Simulation results	227		
XML Modeling 229			
XML structure diagrams	230		
Content of XML structure diagrams	230		
Creating an XML structure diagram	231		
Creating design elements on the XML structure			
diagram	231		
		Programming with Together 241	
		Editing 243	
		Configuring the Editor	243
		Using the Editor.	245
		Edit menu commands.	246
		Editor right-click menu commands	246
		Bookmarks.	247
		Global Bookmarks.	248
		Setting and removing global bookmarks	248
		Viewing, editing, and classifying global book-	
		marks	248
		Editing global bookmark descriptions.	249
		Reordering bookmarks	249
		Classifying global bookmarks	249
		Deleting bookmarks.	250
		Navigating with global bookmarks	250
		Local bookmarks	250
		Setting and removing local bookmarks	250
		Navigating with local bookmarks	251
		Breakpoints	251
		Setting and removing breakpoints.	251
		Working with breakpoints	252
		Advanced Editor features	252
		Code completion	253
		Using Code Sense and parameter tool tips	253
		Example 1	253
		Example 2	254
		Example 3	254
		Using Advanced Code Sense.	254
		Type casting	254
		Example:	254
		Instantiating new objects	255

Example:	255	Query-based search.275
Suggesting commonly used names	255	Go to line.277
Example:	255	Search for usages278
Import Assistant	255	Compiling and Running 281	
Snippets	255	Overview of compiling facilities281
Defining Snippets.	256	Executing the compile and make tools282
Deleting Snippets	256	Configuring and using the standard compiler. .282	
Using Snippets	256	Compiler output.283
Browse Symbol	257	Cross-compilation283
Adding library classes to the sourcepath. .257		Configuring and using an alternative compiler .	284
Context Help	258	Running programs in Together285
Creating a help database	258	Run/debug configurations.285
Example:	259	Debugging 287	
Using context help	259	Overview of Debugging287
Integrated XML Editor	259	Starting a debugger session288
Working with the XML Editor	260	About the Run/Debug tab.288
Developing XML file structure on the XML		Controlling program execution289
tab	261	Using breakpoints290
Developing XML file structure using the table		Setting breakpoints291
view	263	Modifying breakpoint properties292
Editing XML, JSP, and HTML files with XML		Evaluating and modifying variables and expres-	
support disabled	263	sions293
Code Sense in JSP Editor.	263	Displaying structured context294
Viewing HTML files	264	Evaluating arrays294
Tag Library Helper	264	Using watches, threads and frames.295
Tips and Tricks.	264	Using watches295
Bracket Highlighting.	264	Changing the display range296
Swapping case	264	Changing values.296
Selecting rectangular blocks.	265	Changing the display format.296
Splitting the Editor pane.	265	Using threads and frames297
Showing and hiding the Editor pane.	267	Working with monitors297
Using the Editor with an open project	267	Monitors in deadlock and non-deadlock states .	297
Using the Editor with no open project.	267	Attaching to a remote process297
Working with Java files with non- java exten-		Configuring Together to support JDK 1.4 299	
sions	268	Specifying rt.jar299
Defining new extensions.	268	For a new project299
Working with classes with non-java exten-		For an existing project300
sions	268	Enabling Java 1.4 syntax support300
External Editor	269	Configuring the compiler300
Configuring the external editor.	269	Configure external tool.300
Using the external editor	269	Configuring the Builder301
Using search facilities 271		Configuring Ant Runner301
Overview of search facilities	271	Setting Java 1.4 variables for Ant Runner. . .301	
Find and replace	272	Activating Ant Runner301
Find and replace in files.	272		
Search on diagrams	274		

Configuring compiler tool for Ant Runner	302
Using the UI Builder 303	
Overview of the UI Builder	303
Component Explorer	304
Activating the UI Builder	305
Deactivating the UI Builder	305
Setting UI Builder options	306
Choosing a profile	306
Customizing a profile	306
Profile options	306
Creating visually editable UI classes	307
What is a visually editable class?	307
Creating UI classes	308
Using the UI Designer	310
UI Designer view	310
Menu Designer view	312
Adding components from the Toolbox	313
Editing component properties	314
Exposure levels of properties	314
Editing properties using Customizers	314
Editing properties using Property Editors	316
Changing the layout manager for UI components	317
The layout property	317
Using layout setup	317
Supported layout managers	318
Containers with different layouts	319
NullLayout	320
BorderLayout	322
CardLayout	323
GridBagLayout	324
Where to learn more about layout managers	327
Other tips and tricks for working on UI components	327
Selecting multiple components	327
Changing the look and feel	327
Creating event handlers for UI components	327
Changing alignment, spacing, and distribution	328
Changing the order of components	328
Designing menus	330
Creating a new main menu	330
Creating a new menu bar component	331
Adding commands to the menu	331
Setting a menu bar into the container frame	332
Understanding the code	334
Creating a new popup menu component	335
Displaying a popup menu at runtime	335
Defining menus visually	336
Defining new menu items	337
Menu definition syntax and syntax helpers	338
Using the menu definition syntax helpers	338
Defining a checkbox menu item	339
Changing the default state of a checkbox	340
Defining a radio-button menu item	340
Changing the default state of a radio-button	340
Creating nested menus	341
Inserting separators	341
Creating separators using in-place editing	341
Creating separators using the right-click menu of the menu item	341
Creating separators using the Toolbox	341
Moving menu items	341
Defining menu item properties	342
Disabling and hiding a menu using the right-click menu	342
Adding icons to menus	343
Icon properties	343
Defining menu colors and font	344
Renaming menu component identifiers visually	344
Renaming component identifiers in source code	344
Defining menu items using the UI Component Explorer	345
UI Builder audits and metrics	345
UI audits	345
UI metrics	346
Generating user interface documentation	346
Customizing the Toolbox component palette	347
Creating a component archive	347
Invoking the Toolbox customization dialog	347
Customizing toolbox categories	348
Reordering existing component categories	348
Creating new component categories	348

Removing component categories	348	Together's IDE367
Restoring the default component categories	348	Continuus/CM369
Adding custom bean components	349	Known issues.369
Adding components to a category	349	Perforce.369
Removing components.	349	Known issues.369
Changing component display names.	350	Problems and workarounds370
Changing component icons	350	PVCS Version Manager.370
Java features supported in designable classes	350	Known issues.370
Using Version Control 355		StarTeam370
Multi-user development	355	Known issues.370
Overview of version control support	356	Using Together with ClearCase371
Using Together with a Version Control System 356		Setting up the interface371
Getting started with version control	356	ClearCase options372
Configuring Together for version control	357	Basic Version Control373
Enabling version control support	357	ClearCase Move373
Choosing which VCS to use	358	Add374
Configuring system-specific version control options	359	Update374
Setting up CVS options.	359	Check-in374
Setting up CVS Local or LAN-based repository.	360	Uncheckout.374
Setting up CVS Pserver connection	360	Checkout375
Setting up direct connection to the server	361	System.375
Setting up an SCC-compliant versioning system	361	Advanced Version Control.375
Coroutine classes	362	History376
Setting SCC Version Control options	362	Remove376
Switching among different SCC providers	362	Difference.376
Tuning SCC support for Visual SourceSafe	363	Details.376
Enabling version control for projects.	363	Refresh376
New projects	363	Reporting Results and Errors377
Existing projects.	364	Unified Change Management377
Together files to include in version control	365		
Interacting with version control	365		
Examining differences	366		
External visual diff tools	366		
Product-specific VCS notes	366		
PVCS command line tools.	366		
PVCS Dimensions	366		
Problems and workarounds	367		
Configuring PVCS Dimensions to work with			

Generating Documentation 379

Generating Documentation for Together Projects 381

Generating documentation for an open project .381
Generating HTML documentation382
Generate HTML documentation generation parameters382
Printing documentation383
Print Documentation generation parameters .383
Generating documentation using a template.384
Documentation generation parameters . .385
RTF documentation options386
Using a custom formatting template (.dot file) with DocGen.387
Generating reports with the Rational® Requi-

sitePro® plug-in	388	and footers	412
Setting up the reporting tool.	388	Setting area properties	413
Running reports.	389	Creating controls and setting control properties	413
Documentation Generation Tips	389	Labels, images, and panels	413
This section provides tips that may be used		Labels	413
when generating documentation for your		Images.	414
projects.	389	Panels	415
Adding package-level documentation	389	Data controls.	415
Line breaks in PDF documentation.	390	Data Control Sources	415
Automating documentation generation.	390	Displaying custom properties	416
Launching the documentation generator from		Formula and text controls	417
the command line	391	Formula controls.	417
Documentation generation options	392	Include text controls.	418
Options for generating documentation using		Managing the display of output.	418
a template	392	Moving and resizing controls	418
Options for generating HTML documenta-		Text format settings	419
tion	393	Aligning controls	420
Designing Custom Documentation Templates 395		Hyperlinking controls to element documentation .	420
Template Organization	395	Making a target from a static section, footer,	
Body sections and the DocGen engine.	396	or header	420
Body section types	397	Linking a control to a target	421
Metamodel types.	398	Creating Multi-Frame HTML Documentation 423	
Creating new templates	400	Multi-frame HTML document template basics .	423
Setting template properties	401	Frameset document template organization. .	424
Designing the template body	402	Creating frameset templates	424
Creating, arranging, and resizing sections .	402	Defining the frameset structure	424
Setting section properties	403	Specifying frameset properties	425
Output Style	403	Specifying frame properties	426
Other options (Settings tab for folder sec-		Designing the frameset template body	426
tions)	403	Frameset template body organization	426
Enabling Conditions	403	Frameset template processing	427
Accessing model elements through iterator sections		Call to template section properties	427
404		Naming the generated document	427
Element iterators	404	Naming the output directory.	428
Element iterator scopes.	405	Creating hypertext links (advanced)	429
Sorting	406	Assigning a target frame to a link reference .	429
Element property iterators	406	Targeting an element's "specific" documenta-	
Folder sections	407	tion	429
Reusing templates and stock sections	408	Image mapping diagram elements	430
Calls to stock sections	408	Creating compound link references.	431
Creating calls to stock sections	408	Javadoc link references	431
Creating and editing stock sections.	408	Converting JDRefs into hyperlinks	431
Calls to template sections	409	Converting {@link} tags	432
Documentation Template Controls 411		Converting the value of an element's prop-	
Using static sections, headers, and footers	411	erty.	432
Creating and deleting static sections, headers,			

Documentation Generator and Template Designer Reference 435

Documentation generator variables	435
Documentation generator functions	438

Contemporary Software Practices 451

Refactoring 453

Enabling Refactoring	453
Showing code structure	454
Moving classes, interfaces, attributes, and operations	456
Moving classes and interfaces	456
Moving attributes and operations in the class hierarchy.	456
Renaming	458
Encapsulating attributes	459
Extracting interfaces, superclasses, and operations 460	
Extracting interfaces and superclasses	460
Extracting operations	461
Summary of refactoring commands	465

Using the Testing Framework 469

Overview of the testing framework	470
Before you begin	471
Activating the testing framework module.	471
Accessing the sample test project CashSales	471
Creating a test project	472
Configuring options	474
Testing framework options	475
Options for visual steps	475
Options for test suites	476
Options for JUnit steps.	476
Options for building unit tests	476
Support for JUnit	476
Support for JUnitX	477
Support for Cactus	477
Support for HttpUnit.	477
Setting up version control.	477
Browsing test servers and working with profiles	478
Developing unit tests for source code	479
Setting up filter options for the test builder	480
Generating stubs for test cases	480
Writing test cases.	481
Creating a test suite	482
Importing JUnit tests.	483

Creating and configuring collection suites	483
Using templates to create test cases	483
Elements of the unit test builder configuration file	484
Using the XML editor	485
Testing the default ClassLoader of an application 485	
Working with unit tests	485
Running JUnit tests	485
Stopping the test server.	486
Regenerating test cases	486
Developing visual tests for a user interface	486
Creating a test collection	487
Recording a visual script	487
Setting options for recording a visual script	489
Editing a visual test	490
Working with visual tests	491
Running visual tests.	491
Creating an assertion	493
Viewing test results.	495
Working with test results in the Message pane table.	495
Navigating from test results to diagrams, test plan, or code	496
Generating a report in HTML	496
Viewing test results in XML format	497
Distributed testing	497
Setting up the bootstrap loader for a test server	498
System Requirements	498
Installing the bootstrap loader	498
Setting run configuration options	499
Testing your application with Together	499
Running a stand-alone test server	500
Troubleshooting	501
Network security	501
Known problems and limitations	502

Templates and Patterns 503

Code templates	503
Template properties.	504
Using template macros	505
Default properties in templates	505
Browsing available code templates	506
Editing code templates	507
Applying source formatting	508
Custom code templates	508

Working with the Code Template Expert . . .	508
Creating custom code templates manually . . .	510
Groups of templates	510
Displaying custom template names	511
User-defined macros	512
Creating templates with custom file extensions	512
Defining the source for extension of the newly	
generated files.	512
Defining the custom template extension . . .	513
Creating files with custom template exten-	
sions	513
Patterns	513
What are patterns?	513
Working with various languages	514
Elements generated by patterns.	514
Pattern behavior	515
Creating patterns	515
Providing pattern descriptions	517
Using templates and patterns	517
Creating classes and links by patterns	518
Creating a stub implementation.	519
Creating members by pattern	519
Choosing a pattern for a Member.	520
Refactoring with patterns	521
Audits and Metrics 523	
Overview of audits and metrics analysis	523
How should metrics be used?.	524
Running audits and metrics in Together	525
Available audits and metrics	526
How to perform metrics analysis.	526
Metrics for audit violations	526
How to perform audits	527
Options for processing audits and metrics . .	527
Output for audits and metrics	528
Sorting results	528
Updating results	528
Exporting results	529
Creating linked HTML reports for metrics .	
529	
Saving and loading metrics results	530
Comparing metrics results.	531
Graphic output of metrics results.	532
Kiviat graph	532
Bar graph.	533
Distribution graph	535
Printing results.	536
Printing results from the table	536
Printing graphic results	537
Automatic correction of audit violations . .	537
Creating and using saved sets of metrics or au-	
dits	538
Language-specific metrics and audits	539
Audits and metrics supported for Java	539
Audits and metrics supported for C++	544
Metrics supported for Visual Basic 6	545
Metrics supported for C# and Visual Basic .NET	
546	
Running audits and metrics from the command line	
547	
Usage	547
Options	548
Extending the QA module.	549
Additional information sources	550
J2EE Development 551	
J2EE Support 553	
Supported J2EE technologies	553
Special J2EE diagrams	554
Creating a J2EE diagram	555
EJB assembler diagram	556
Web application diagram.	556
Enterprise application diagram	556
Taglib diagram.	557
Application client diagram.	557
Resource adapter diagram	557
References support	558
Security support.	558
Creating EJB Components 561	
Setting the EJB properties for a project	561
Creating EJB components	562
Setting component editing and display options .	
562	
Setting component synchronization options .563	
Creating EJB components from the class diagram	
566	
Creating EJB components using the Object Gal-	
lery	566
Applying an EJB pattern to an existing class .567	
Importing entity EJBs from a database	567
Reverse engineering EJB source code	569
Using EJB Inspectors to set EJB properties	569

Configuring EJB inspectors	569	Sharing home and remote interfaces588
EJB Inspector tabs	569	Creating a second implementation class .588	
General tab	570	Deleting implementation classes with shared	
References tab	571	interfaces589
Deployment properties tab	572	Customizing the default EJB code.589
Fields tab.	574	Assembling EJB Modules 591	
Methods tab	575	Visual assembly of EJB applications591
Relationships tab	576	Creating and modifying EJB modules592
Setting additional deployment properties . . .	576	Creating an EJB module592
SunEE AS Properties Inspector tab.	576	Placing elements on an EJB assembler diagram .	593
Borland Enterprise Server 5.2 Inspector tab .	577	Setting properties of an EJB assembler diagram .	593
Borland Enterprise Server 5.2 Inspector tab		Displaying EJB shortcuts on EJB assembler dia-	
for session EJBs	577	grams.594
Borland Enterprise Server 5.2 Inspector tab		Creating EJB shortcuts594
for entity EJBs	577	Displaying EJBs on EJB assembler diagrams .595	
Borland Enterprise Server 5.2 Inspector tab		Showing and hiding EJB component parts . .596	
for message-driven EJBs	578	How to copy and paste EJB shortcuts.596
WebLogic Inspector tabs	578	EJB verification and correction597
WebLogic 6.0 and WebLogic 6.1 Inspector		Verification overview597
tabs	578	Using verification and correction597
WebLogic 7.0 Inspector tabs.	578	Deploying an EJB module598
WebLogic 7.0 Inspector tab for session		Container Transactions and	
EJBs	578	EJB References 601	
WebLogic 7.0 Inspector tabs for entity EJBs		Container transaction elements601
579		Container transaction attribute types601
WebLogic 7.0 Inspector tabs for CMP enti-		Creating container transaction elements . . .602	
ty EJBs	579	EJB references604
WebLogic 7.0 Inspector tabs for message-		EJB reference diagram elements604
driven EJBs.	581	EJB properties elements.604
Working with EJBs.	581	Reference attributes in the client.605
Compiling EJBs.	581	Supported EJB reference types606
Entity beans.	582	Reference attribute inspectors606
Default entity bean code	582	EJB reference diagram elements607
Setting the persistence	583	EJB resource and resource environment refer-	
Creating primary keys	583	ence diagram elements609
Creating container-managed relationships .	583	Creating EJB environment references.610
Session beans	585	Designing and Developing Web Applications 613	
Default session bean code	585	Working with Web application diagrams613
Making the session EJB stateful or stateless .	585	Creating a Web application diagram614
Message-driven beans	586	Web application diagram elements615
Setting the project properties for message-		Servlet element616
driven beans.	586	Web application diagram properties617
Default message-driven bean code	587	General tab617
Setting deployment properties for message-		Login Config tab618
driven beans.	587		

App event listeners tab	619
Creating shortcuts to applets, EJBs, EJB properties, container transactions, and references	619
Developing applets	620
Creating an applet	620
Running and debugging an applet	620
Deploying an applet	621
Working with servlets	622
Creating a servlet	622
Running and debugging a servlet	624
Putting a servlet into a Web application	624
Working with JSPs	624
Placing a JSP on a Web application diagram	624
Debugging JSPs	626
Working with tag libraries	627
Creating tag handlers on class diagrams	627
TagLib diagrams	628
Diagram elements	628
Taglib diagram properties	628
Properties of the handlers	629
Creating and using tag libraries	629
Working with filters	631
Creating filters and filter mappings	631
Setting filter and filter mapping properties	632
Deploying filters	632
Working with listeners	632
Supported event types	632
Creating a listener	633
Associating listeners with Web applications	633
Deploying a Web application	634
Designing and Developing Enterprise Applications 637	
Visual assembly of enterprise applications for deployment	637
Creating enterprise application diagrams	638
Enterprise application diagram elements	639
Properties of an Enterprise Application diagram	639
Creating diagram shortcuts	639
Importing existing archive files	641
Deploying an enterprise application	642
Designing and Developing Application Clients 645	
Modeling enterprise application clients	645
Creating application client diagrams	646
Visual design elements	648
Properties of an application client diagram	648
Creating “one-click” application clients	648
Using an application client diagram	649
Designing and Developing Resource Adapters 651	
J2EE Connector Architecture	651
Creating a resource adapter diagram	652
Resource adapter diagram elements	652
Creating a resource adapter diagram using a pattern	653
Using a resource adapter diagram	655
J2EE Platform Security Support 657	
Security support in J2EE diagrams	657
Security elements in J2EE diagrams	658
Security roles	658
Linking security elements	659
Security in EJB modules	659
Declarative security and method permissions	660
Programmatic security and security role references	661
Security in Web applications	663
Security roles and principals	663
Constraints and Web resource collections	664
Security in enterprise applications	666
Security in resource adapters	666
J2EE Deployment 669	
Deployment process	669
Supported application servers	670
Integrating other application servers	670
Generic server options	671
Installations and resources required for deployment	671
Starting application servers from Together	672
J2EE Deployment Expert features	674
Diagrams for deployment	674
Specifying parameters	674
Running the J2EE Deployment Expert	675
Main scenarios for using the J2EE Deployment Expert	676
Configuration files	677
Server-specific deployment information	677
Sun EE reference implementation application server	677
Borland Enterprise Server 5.2	678
IBM WebSphere Application Server	678
BEA WebLogic Application Server	678

J2EE Deployment Expert options common to all servers	679	Encoding in deployment descriptors	700
First page: process options	679	Web Services 703	
Sun EE and Borland Enterprise Server 5.2 process options	682	Creating Web Services 705	
IBM WebSphere process options	682	Supported platforms	705
BEA WebLogic process options	682	Creating Web services from existing code	706
Sun EE common properties	684	Creating a Web service from a class.	706
Borland Enterprise Server 5.2 common properties	684	Properties tab of the Web service inspector.	706
IBM WebSphere common properties	684	Server-specific Properties tab of the Web service inspector.	708
BEA WebLogic common properties	684	Type Mapping Properties tab of the Web service inspector	709
Verify / Correct Sources page	684	Creating a Web service from an EJB.	712
Simple JSP Client Generation page	685	RPC-style Web service	712
Options for WebSphere 3.5	686	Message-style Web service	712
Options for WebLogic 5.1	686	Creating Web services by using patterns	713
Options for Sun EE, Borland Enterprise Server 5.2, WebSphere 4.0, WebLogic 6.0, 6.1, 7.0	687	Exposing methods to clients	713
Command Line File with Instructions for Deployment page	687	Generating WSDL files from Web services.	714
Server-specific J2EE Deployment Expert pages	688	Generating Web service clients	715
Sun EE Deployment Properties page.	688	Generating a proxy Web service client from a WSDL file	715
Borland Enterprise Server 5.2 Run-time Deploy properties page	688	Using patterns to generate proxy Web service clients	716
BEA WebLogic Run-time Deploy properties page	689	Apache SOAP	716
IBM WebSphere 3.5 pages.	691	BEA WebLogic	717
The IBM WebSphere 3.5 deployment options differ significantly from those for other servers.	691	Deploying a Web Service 719	
EJB Deployment Properties page	691	Setting deployment properties	719
Servlet Deployment Properties page	692	Running the Web Services Deployment Expert	720
Client Properties page	694	Deploying to Apache-SOAP.	720
IBM WebSphere 4.0 pages.	695	Process options for Apache-SOAP	721
Advanced Single Edition Properties page	695	Common Properties for Apache-SOAP.	721
Deployment Properties page	695	Run-time Registering Web Services for Apache-SOAP	721
Making transitions among application servers.	696	Deploying to BEA WebLogic 6.1, 7.0	722
Transitions among specifications.	696	Process options for BEA WebLogic 6.1, 7.0.	722
Automatic verification and correction of EJBs	697	Common Properties for BEA WebLogic 6.1, 7.0	722
Manual correction of EJBs	697	Run-time Registering Web Services for BEA WebLogic 6.1, 7.0	723
Moving from EJB 2.0 to EJB 1.0	697	Web Services Deployment Expert for IBM WebSphere Application Server 4.0	723
Moving from EJB 1.0 to EJB 1.1	698	Process options for IBM WebSphere 4.0	723
Transitions among application servers	698	Common Properties for IBM WebSphere 4.0	723
Developing and deploying clients for various servers	699	Deployment Properties for IBM WebSphere 4.0	724

UDDI Browser 725	
UDDI Browser overview	725
Searching for Web objects.	726
Search options	726
UDDI browser options	727
Operator configuration options	727
Search Options.	728
Search results	728
Properties display.	729
WSDL file display	729
Generating a WSDL client	730
Saving WSDL files	730
Publishing Web services	731
Publishing prerequisites.	731
Logging on to the operator site	732
Publishing a Web service on an operator site	732
Managing registered objects	734
Using templates	734
Extending Together 737	
Advanced Customizations 739	
Customizing the user interface	739
Creating custom diagram types	740
Step 1: Creating icons and icon references for the	
UI.	741
Editing the Together.bat file	741
Referencing images in the config file .	741
Step 2: Creating the diagram configuration file.	741
Step 3: Defining tree-view icons	742
Step 4: Defining toolbar icons	742
Step 5: Defining viewmaps	744
Defining graphical presentation	744
Defining the shape of an element	745
Defining the element's options	745
Enable display of the element's unique	
name	745
Add the element's name to the compart-	
ment	745
Enable in-place editing	746
Define label properties	746
Define links	746
Sample configuration file	746
Customizing Property Inspectors	747
Adding custom pages and fields to the Inspector	748
API-based Inspector customization	748
More API documentation	750
Customizing Inspector by means of the Custom	
Properties Module	750
User-defined inspector customization with	
the Inspector Property Builder	751
Customizing View Management <i>Show</i> options .	752
Changing the display text of a Show option .	752
Removing a Show option from the Options dia-	
log.	753
Adding a Show option to the Options dialog	753
Together Open API 755	
Overview of the Together Open API	755
Understanding modules	757
Locating modules	757
Registering a module	758
Starting modules	758
Examples for Hello World	758
Extending Together using modules.	759
Types of modules	760
Interfaces implemented by the modules	760
Viewing modules	760
Running modules	761
Guidelines for developing modules	761
Applying naming conventions.	762
Documenting the module	763
Order of Tags	764
Deploying the module	764
Tutorial: Writing and deploying modules	764
Source code for the module	765
Declaring a module	766
Declaring a module in a Manifest.mf file .	766
Declaring a module in a .def file	769
Specifying whether a module is a startup	
module	769
Defining a visible name for the module	769
Specifying the location of the Modules tab	770
Adding the module name to menus for	
classes, interfaces, and members	770
Adding the module name to menus of ele-	
ments with specific shapetype	770
Rules for the manifest file.	771
Compiling the module	771
Storing the compiled class	772

Evaluating the results	772
Troubleshooting your module	773
Accessing more API documentation.	774

Configuring Together for Multiple Users 775

Overview of a shared multi-user configuration	775
Setting-up multiple workstation installations .	776
Creating the start-up pointer file	777
Launching via the command line and pointer file	778
Adding new configuration levels to predefined lev-	
els	778
Adding new configuration directories to a	
shared location	779
Adding new properties directories to a local in-	
stallation.	779
Creating the path.config file	779
Viewing added configuration levels	780
Modifying default configuration levels	780

Language Support 783

Language Support	783
Switching languages on or off	784
Design level modeling	784
Essential features	785
Language-specific configuration files	786
Language-specific Inspectors and usages	787
Visual Basic 6 Inspectors	787
Visual Basic.Net Inspectors	788
C# Inspectors	789
CORBA IDL Inspectors	790
CORBA IDL tips	792
C++ Inspectors	792
Visual Basic 6 class members	793
Visual Basic.Net class members	794
Renaming properties.	794
Using Together with C++.	795
Limitations on C++ projects.	795
Defining C++ project structure	795
Setting the search path in large C++ projects	
796	
Skipping standard includes	796
Adding Standard Library Paths.	796
Using the preinclude file	797
Managing include directives	797
Managing C++ macro definitions.	798
Writing macro definitions	799

Default library support	799
Setting wrapper macro options for default	
libraries	800
Configuring Together for C++.	800
Configuring C++ header and implementation	
file extensions	801
Specifying file extensions for code generated	
by Together.	801
Configuring C++ compile and make utilities	
802	
Working with C++ language features.	802
Header and implementation files	803
Templates, structures, unions, and overload-	
ed operators	803
Interfaces	804
Enumerations, typedefs and global symbols .	
804	
Properties.	805
Known problems and recent corrections	806
Deep parsing	806
Recognizing member operation definitions. .	
806	
Together patterns for member operations	
807	
Documentation of C++ member definitions .807	
Using the definition documentation feature .	
807	
Accessing the defDocComment module	
through the API	808
Enabling HTML, PDF, and RTF documen-	
tation of member definitions	809
Using the sample documentation template	
809	

Commands, Macros, and Keyboard Shortcuts 811

Command Line Parameters	811
Basic command line syntax.	811
Using the Windows launcher	813
Invoking the Together main class	813
Command-line examples	814
Standalone formatter	814
Documentation generation	815
Running modules at start-up	815
Parameters for the Together.exe Launcher.	815
Parameters for virtual machine preferences .815	
System Macros	816
Referencing configuration properties as mac-	
ros	819

Template Macros	820	Editor/Compiler/Debugger shortcuts826
Keyboard shortcuts	821	Miscellaneous827
Main Window and Main Menu.	821		
Designer shortcuts	824		

Index 829

PREFACE

The *User Guide* contains instructional and conceptual information for using Together ControlCenter to build enterprise level applications. The guide assumes that you have installed Together on your computer. If not, refer to the Readme.

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.).
- 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
Readme	Late-breaking information including: <ul style="list-style-type: none">• Licensing notes• System requirements• Installing and starting Together• Known problems• Significant fixes	Your Together installation under \$TGH\$/readme.html

Table 1 Together documentation set (continued)

Item	Description	Location
<i>User Guide</i>	Comprehensive information most relevant to the user, including: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • Together's main menu under Help Manuals • Your Together installation under \$TGH\$/doc/userGuide.pdf • http://info.borland.com/techpubs/together/
Online Help	A subset of the information in the <i>User Guide</i> , including: <ul style="list-style-type: none"> • Main window components • Brief instructions for using Together features with references to the <i>User Guide</i> for more details • Dialog help 	Together's main menu under Help Application Help Contents
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> .	<ul style="list-style-type: none"> • 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://info.borland.com/techpubs/together/

Documentation for Together Solo

This guide includes instructions for using the features supported in Together Solo. However, note that not all of the information in this guide applies to Together Solo. Following is a list of features supported by Solo:

- Working with projects
 - Reverse engineering
 - Import/export XMI models
 - LiveSource support for Java, C++, Visual Basic 6, Visual Basic .NET, C#

- Multi-level configuration options
- Modeling
 - UML diagrams
 - Language-neutral diagrams
 - Real-Time modeling
 - View management features
- Programming
 - Programmer's editor
 - Compile-Make-Run
 - Java debugger
 - UI Builder
- Version control support
- Documentation generation
- Refactoring
- Templates and patterns
- Support for external tools and third-party integrations

These features are documented in the chapters of the *User Guide* listed below. Other chapters do not apply to Solo.

Chapter 1, "Introducing Together" on page 29

Chapter 2, "Tour of Together" on page 33

Chapter 3, "Setting Your Personal Preferences" on page 65

Chapter 4, "Configuring Options" on page 73

Chapter 5, "Working with Projects" on page 95

Chapter 6, "Reverse Engineering" on page 111

Chapter 7, "Importing and Exporting Information" on page 117 (only "Importing and exporting XMI models" applies)

Chapter 8, "Generating makefiles" on page 125

Chapter 9, "Introduction to Modeling" on page 129

Chapter 10, "Working with Diagrams" on page 133

Chapter 11, "UML Modeling" on page 165

Chapter 13, "Real-Time Modeling" on page 205

Chapter 15, "Editing" on page 243

Chapter 16, “Using search facilities” on page 271
Chapter 17, “Compiling and Running” on page 281
Chapter 18, “Debugging” on page 287
Chapter 19, “Configuring Together to support JDK 1.4” on page 299
Chapter 20, “Using the UI Builder” on page 303
Chapter 21, “Using Version Control” on page 355
Chapter 22, “Generating Documentation for Together Projects” on page 381
Chapter 27, “Refactoring” on page 453
Chapter 29, “Templates and Patterns” on page 503
Appendix 47, “Language Support” on page 783
Appendix 48, “Commands, Macros, and Keyboard Shortcuts” on page 811

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/>

1

GETTING STARTED

- Chapter 1, “Introducing Together”
- Chapter 2, “Tour of Together”
- Chapter 3, “Setting Your Personal Preferences”
- Chapter 4, “Configuring Options”

Introducing Together

Together ControlCenter includes the features you need 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.

Modeling support includes all the standard UML diagrams, plus additional diagrams for other special types of modeling. Class and sequence diagrams generate source code automatically and keep it in sync. While you may choose to create models without generating source code, or code without diagrams, the main advantage of Together is its ability to fully synchronize diagrams and source code using LiveSource™, also known as *simultaneous round-trip engineering*.

Source code editing is supported for multiple programming languages with a syntax-savvy editor. Coding support also includes code patterns and refactoring, along with a Java debugger. Audits and metrics for all supported languages, in addition to a testing framework for Java, help you control the quality of code.

Other major features include version control integration, generating source code documentation, and import and export of databases and models, as well as full J2EE support for e-commerce development and a user interface designer for Java. Extensive integrations allow you to use external tools such as compilers and many third-party products from within Together, in addition to internal tools.

This chapter provides important background information for working with Together. It includes the following topics:

- “Together concepts” on page 30

- [“What’s in the Together Installation” on page 32](#)

Together concepts

As you begin to work with Together ControlCenter, it is important to understand some of the main ideas behind Together. If you have worked with previous versions of Together, you will find that the concepts are familiar.

Projects: creating a context

Source code files and models you create in Together are kept in a *project*, which is represented by a project file with the `.tpr` extension. Outside of a project, you can open any text-based files for editing in the Together Editor (either when no project is open, or when you want to edit files that do not belong to the current project). However, modeling, version control integration, generating documentation, and all other Together features require the context of a project. The scope of a project usually corresponds to an application your team is developing.

A project has a primary root directory where project files are stored, and any existing files in that directory become part of the project. Together reverse engineers any existing source code to create a model, if necessary. Each project has an extensive set of properties such as the location of resources required by the project, version control configuration, the default programming language, and the set of features to be loaded with the project.

Modular architecture: turning features on and off

Together consists of a large set of modules representing available features. Because a given project will not require all the available features, some feature modules can be turned on or off as necessary. This means that unneeded features are not loaded, simplifying the user interface by displaying only those commands and features you need for your project.

Features are activated on several levels:

- Your selected *user role* determines which features are available for all projects.
- The *Activate/Deactivate Features dialog* enables you to set which features to load for the current project.
- Features activated *on demand* are loaded automatically when they are needed.

Each feature module has a configuration file that stores property settings that apply to the feature. You can use the Options dialog to make configuration settings.

LiveSource: keeping code and models in sync

One of the main Together features is LiveSource™ (also known as *simultaneous round-trip engineering*), the ability to immediately synchronize diagrams with their implementation code. Round-trip engineering is the combination of reverse engineering (drawing models from code) and forward engineering (generating code from visual models). Simultaneous round-trip engineering means that when you change a code-generating diagram, Together immediately updates the corresponding source code, and when you change the code, Together updates the visual model. In this way, diagrams are always synchronized with the source code that implements them.

You can customize forward and reverse engineering and/or source code formatting. See [Chapter 4, “Configuring Options”](#) for more information.

LiveSource only applies to diagram types that generate source code: class diagrams, sequence and collaboration diagrams, and J2EE-related diagrams.

LiveSource is also turned off for certain user roles. The Programmer role provides code editing without generating diagrams, and the Designer role provides modeling without generating code.

Flexible user interface: roles and workspaces

The Together user interface (including menus, toolbars, and panes) changes according to how you are working with Together.

The menus, toolbars, and panes available depend on several factors:

- your *user role*
- the *project context* (whether a project is open, and which feature modules are activated)
- the selected *workspace* in the project

Roles

During installation, you choose a user role (*Business Modeler, Designer, Developer, or Programmer*). When Together starts up it shows only those menu commands, toolbars, and panes that are appropriate for your chosen role. The role also determines the default workspace. The user role is a global configuration setting that you can reset in the Options dialog. For information about setting the user role, see [“Role-based workspaces” on page 67](#).

Workspaces

A *workspace* is an arrangement of panes that you can save and reuse as you find convenient. Workspaces are saved with the project. For information about workspaces, see [“Setting up workspaces” on page 65](#).

What's in the Together Installation

Before you begin working with Together, take a moment to examine the file structure and some of the key files in the Together installation. This section does not describe all the files and directories, but summarizes the highlights.

`TGH` is the Together home directory, the root of your installation as you specified during installation. The root directory contains readme files with important release information.

`TGH/bin` contains the launcher files `Together.sh` (for all UNIX-based operating systems) and `Together.exe`, `TogetherCon.exe` (for Windows systems). The two Windows executables are identical, except `TogetherCon.exe` provides console output and `Together.exe` does not. Both Windows executables use `Together.bat` as the configuration file. The batch file sets necessary environment variables to default values and launches Together using the Sun JDK installed with Together. `TGH/bin` is also the directory where license files reside.

`TGH/bin/<OS>` subfolders contain additional tools, such as CVS clients. Each subfolder contains tools specific to the corresponding operating system. The **unix** and **win32** folders contain launchers for launching individual Together features without Together. [Table 2](#) lists the launcher files and what to use them for.

Table 2 Tool launchers in the `/bin/` directory

<code>/win32/</code>	<code>/unix/</code>	Purpose
<code>Formatter.bat</code>	<code>Formatter.sh</code>	Format source code according to specified parameters without starting Together.
<code>showhelp.bat</code>	<code>showhelp.sh</code>	Launch Together online help without starting Together.
<code>UMLDoc.exe</code>	n/a	Generate HTML documentation for a project without starting Together.

The **win32** subfolder also contains the following Windows executable files:

`MsDev_run.exe` (used for launching Visual Studio 5 or 6 as an external editor)

`oistart.exe` (starts Windows tools for associated file types)

`startIde.exe` (starts external browsers)

`TGH/bundled` contains third-party tools that are distributed with Together.

`TGH/config` contains configuration files for Together features. While it is possible to edit the config files manually, you should not find it necessary. Configuration properties can be modified using the Options dialog.

`TGH/modules` contains feature modules. Each module may contain its own configuration file and resources, though you probably will not find it necessary to manually edit config files or module resources. The document `modulesHandsOn.html` in this folder explains how to add your own custom module.

Tour of Together

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

- “Getting started with a sample project” on page 33
- “Navigating the Together main window” on page 37
- “Explorer pane” on page 42
- “Designer pane” on page 52
- “Editor pane” on page 54
- “Property Inspectors” on page 56
- “Object Gallery” on page 61

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:

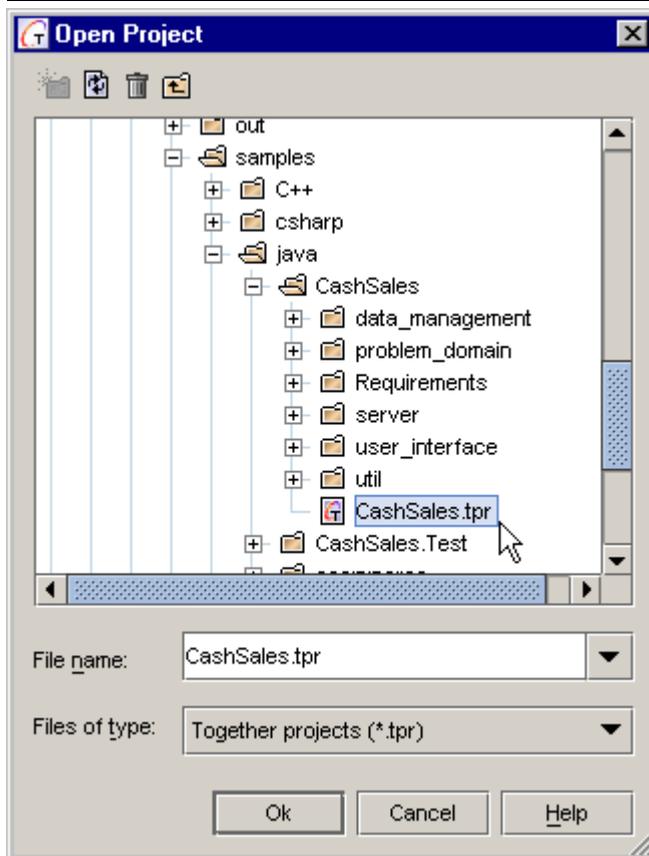
1. Select **File | Open Project**.

2. Pick the desired project from the resulting dialog box. Together project file names end in `.tpr`, as shown in [Figure 1](#). For the CashSales project, select `TGH/samples/java/CashSales/CashSales.tpr`.
3. Click **OK** to quit the dialog and open the project.

Notice that once you have opened a project, the project location and name is added to the list under **File | Recent Projects**. You can choose a project from the list to open it. Use this method as a shortcut for opening projects that you have already worked with. Recent Projects saves a list of all the projects you open during a Together session. The next time you start Together, the Recent Projects list contains the last 10 projects you opened.

Tip By default, each time you start Together it automatically opens the last project that was open the last time you ran Together. You can control this behavior in the Options dialog (**Tools | Options | Default Level**) under *General - Desktop options: Automatically open last opened project*.

Figure 1 Opening the sample Java CashSales project.



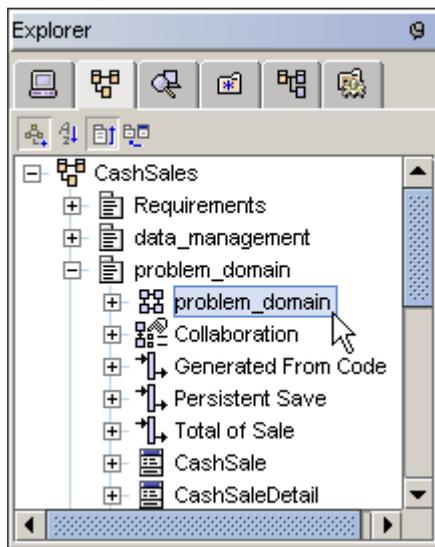
Opening a diagram

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 both the Explorer and Designer panes.

To open `CashSale.java` from the Designer pane:

1. Find the `CashSale` class node in the `problem_domain` diagram in the Designer.
2. Click **CashSales** once to select it. `CashSale.java` automatically opens displays 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 `CashSale` 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 pressing `Ctrl+Z`.

To observe how changing code also changes diagrams:

1. Find the declaration of `SALES_NEW` in the file `CashSale.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 `CashSale` 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, testing framework, and many others. Some of the features are activated by default. You must activate others that you want to use.

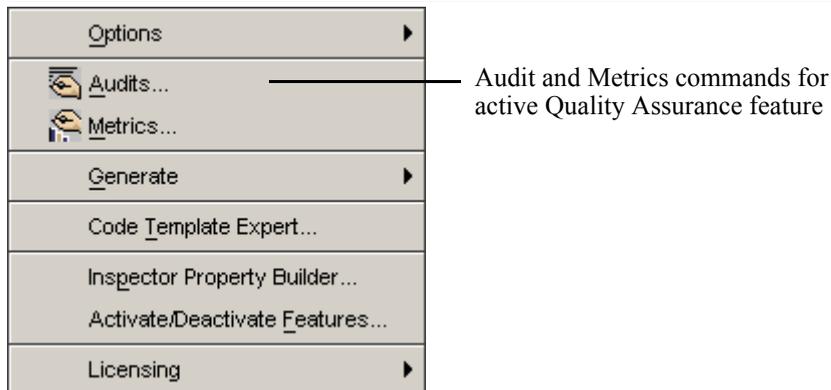
To activate the *Quality Assurance* feature for generating audits and metrics:

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.

Notice that the Message pane displays an informational message when a feature module has been activated or deactivated. For information on opening and using the Message pane, see [“Message pane” on page 55](#).

[Figure 3](#) shows the Tools menu with its new *Audits* and *Metrics* commands.

Figure 3 Tools menu with Quality Assurance activated



Navigating the Together main window

The main window is divided into four major panes.

- **Explorer:** for file system and project navigation.
- **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.
- **Editor:** for viewing and editing source code files and other text files.
- **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.

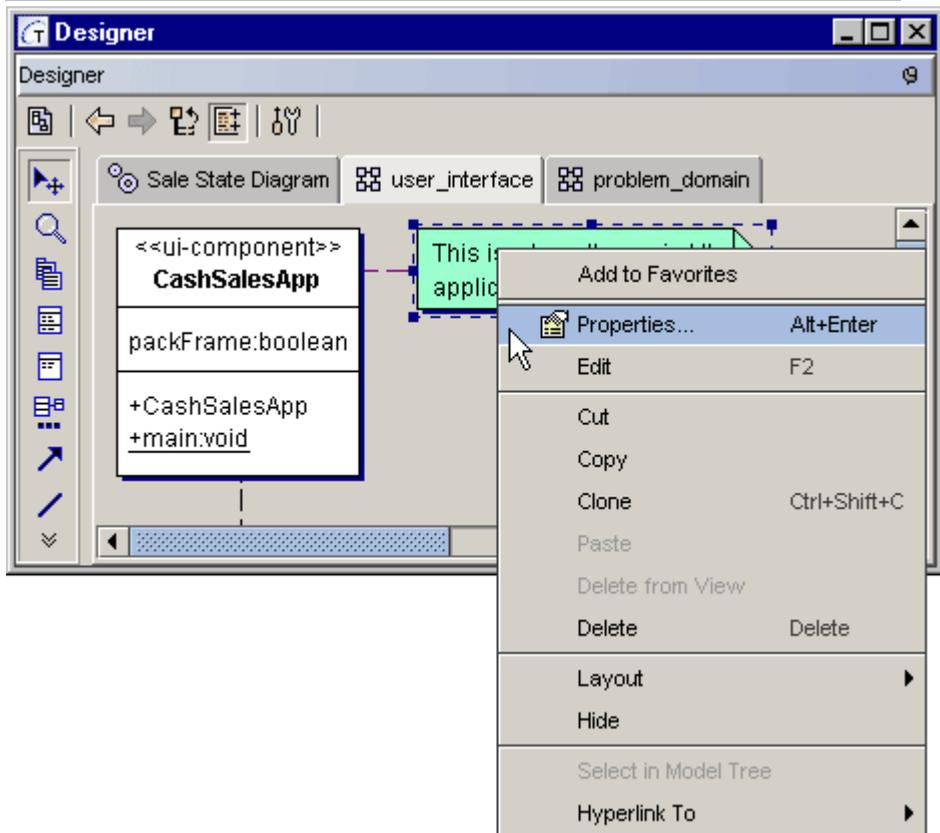
To view or hide a pane, toolbox, or Inspector:

- Click the corresponding view /hide button on the main toolbar (see “Main toolbar” on page 41), or
- On the main menu, choose **View | Main Panes | <name>**, 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



Resizing or expanding panes to full screen

To resize panes, drag their separating bars.

To expand a docked pane to full screen:

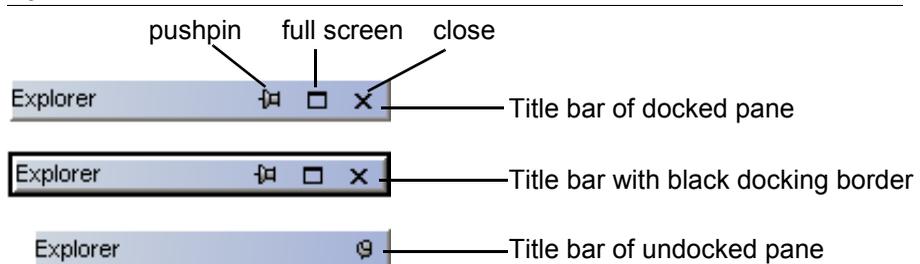
- Click its full screen expander box on the title bar (see [Figure 5](#)), *or*
- Select **Full Screen** from the right-click menu of its title bar, *or*
- Double-click the title bar, *or*
- Bring the pane into focus by clicking it *and*
 - click the **Full Screen** button on the main toolbar, *or*
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

Repeat any of 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 the 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 3](#).

Table 3 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.
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 expert. Available only when the E-Commerce feature is activated.
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 according to which features are activated.
Help	Online documentation for Together, API documentation, web links, and setup for your personal context help libraries.

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 811.

Main toolbar

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

Table 4 Main toolbar buttons

Icon	Name	Meaning
	New object	Displays the Object Gallery.
	New project	Closes the current project and displays the new project dialog.
	Open/Save/Print	Common file operations.
	Undo/Redo	Available for most operations in the Editor and Designer.
	Cut/Copy/Paste	Common edit operations.
	Find	Find in current file in Editor. (Search/Find across multiple files is available on the main menu.)
	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 Designer pane.
	Designer toolbox	Display Designer toolbox in a separate pane.
	Message pane	Toggle to view or hide 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 right-most item on the main toolbar is a menu for selecting the current workspace. For information on workspaces, see [“Setting up workspaces” on page 65](#).

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):** Shows or hides the Message pane; indicates when there is a new message.
- **General info:** Displays information about elements in diagrams as you move the mouse over them.
- **Progress:** Shows the progress of internal processes that you invoke while working in Together.
- **Editor information:** 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 detailed 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 modules or any custom ones you develop.
- View test plans for the Together activatable Testing Framework feature.
- View and modify the properties of user interface components from the Together 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 5](#). 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 5 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 logical 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. In the Directory tab, you can:

- Find files
- Cut, copy, paste, and delete files and folders (right click and choose **Cut**, **Copy**, **Paste**, or **Delete**).
- Open project files. Open text files in the Editor pane independent of a project. Open generic files in an external editor (right click and choose **Tools | External Editor**).
- Create a new project around a directory (right click the directory and choose **New Project**).

- Add a directory to the project path (right click the directory and choose **Add Root to Project**).
- Work with version control.

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:** 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:** Physical files that comprise your project as well as files relevant to your project. This shows only when there is an open 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

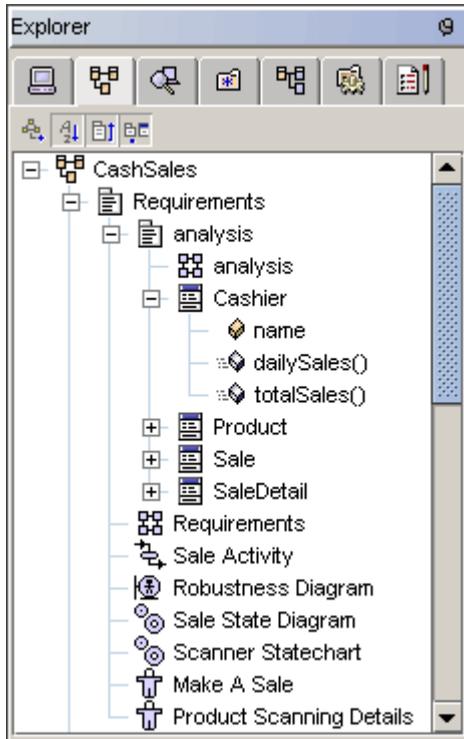
The Model tab becomes visible when you open a Together project. It displays a logical view of the elements that comprise the project's model. (See [Figure 6](#).)

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 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:

- 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*
- Choose **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 -

- 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 source code from the Model tab:

- Double click the class node in the Model tab or select **Edit** from the node right-click menu. (Double clicking a class or interface member highlights the declaration of the member as well.)

- or -

- 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 you can expand nodes on the tab to show their contents.
- **Sort node:** Controls whether nodes in the tree-view are sorted alphabetically.
- **Packages first:** Controls whether packages display first in the tree-view before other content.
- **Show fully qualified names:** Controls whether fully qualified class and interface names are displayed in tooltips. If this toggle is off, short names are displayed.

Note *Show fully qualified names* is not intended for use with Java, because the names become too long as you move deeper into the package hierarchy. Therefore, in Java projects this option displays fully qualified names only for root packages with the package prefix set.

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**.

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**.

Diagrams tab

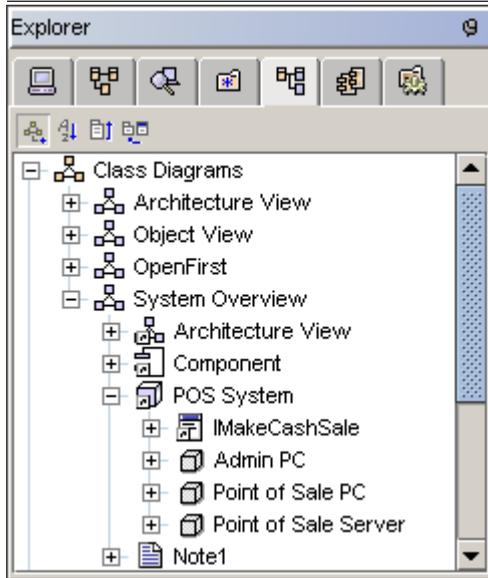
The Diagrams tab displays a list of all types of logical diagrams available in Together. The diagrams are listed according to their types. You can open the diagrams and source code files from the Favorites tab the same way as in the Model tab.

Note The Model tab does not display package diagrams. They are physical diagrams rather than logical diagrams.

When *Expandable diagram nodes* (on 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 including POS System, and IMakeCashSale. Shortcut icons have the curled arrow in the lower left corner. Note that in this example the *Expandable diagram nodes* button on the toolbar is turned on, so diagram elements are visible. If this button is turned off, you will only see the list of diagrams in the project without any diagram content.

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, choose *General* without expanding it.
3. Clear *Show Diagrams tab*.
4. Click **OK** to close the dialog box and save the setting.

Modules tab

Together is highly extensible. Using Java and the Together API, you can develop your own modules to accomplish tasks such as creating custom metrics or documentation and generating 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 file icons as shown in [Table 6](#).

Table 6 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 45, “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. From the main menu, choose **Project | Properties**.
2. Check **Include Components**.

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](#)” on page 50 describes how to copy between Explorer tabs and diagrams.

Special Features tabs

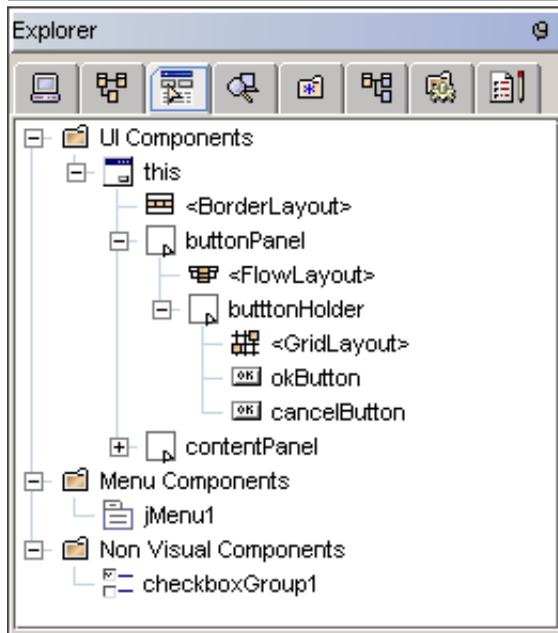
The Explorer displays an additional tab for some features that are activated.

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 52 has a discussion of Designer pane views.)

Figure 8 shows the UI Builder tab for the sample project. See [Chapter 20, “Using the UI Builder”](#) for a full discussion of how to use the UI builder.

Figure 8 UI Builder tab



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 28, “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.

See [Chapter 14, “XML Modeling”](#) for 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 the *General* node.
3. Choose the *New Project* node 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 To select several items at once, use Ctrl+click.

3. Choose **Copy** from the right-click menu.
4. Choose **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 from the Model tab of the Explorer pane (see [“Model tab” on page 44](#)). 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
- Press **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 7 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	Goes 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 8](#) shows the toolbar icons for commonly performed Editor tasks.

Table 8 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.
	Go to last change.

Table 8 Editor toolbar icons (continued)

Icon	Meaning
	Go to line.
	Go to super method.
	Run test. Available only when Testing Framework is activated.
	Displays context-sensitive help from your custom Context Help libraries.

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.
- Display 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 9](#).

Table 9 Status bar icons for the Message pane

Icon	Meaning
	Message pane is open.
	Message pane is closed, no new messages.
	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. From the main menu, choose **Tools | Options | Default Level**.
2. In the resulting Options dialog box, choose the *General* node.
3. Enter the maximum number of messages to display in the *Messages Maximum Count* field.

Note You can also set the Message pane to open automatically when a new error is detected. Choose the option called *Open Message pane on error*.

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

Property Inspectors

Property 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 Designer 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.

When the Inspector is already open, selecting an element automatically loads its properties into the Inspector.

When multiple elements are selected at the same time, the Inspector displays only those properties that are common to all the selected elements.

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 bean properties.

The property editors for the Inspector fields depend on the value types. [Table 10](#) shows the icons that the Inspectors display for some kinds of fields.

Table 10 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 tabset changes depending on the selected element and where you made the selection. (For example, elements selected from the Explorer pane do not have *View* tabs.) This section describes the basic function of each tab of an ordinary class inspector, as shown in [Figure 9](#).

In general, a tab displays two columns:

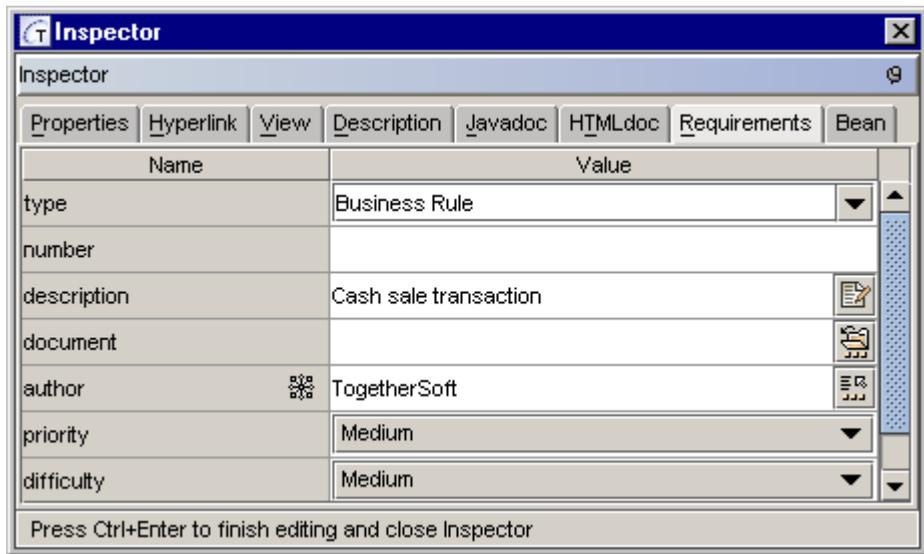
- **Name** shows the name of each property.
- **Value** displays the value or setting for the property.

To enter a value for string type properties, type a value in the edit field of the *Value* column. Some edit fields provide one of the following to help you quickly and accurately define a value:

- An arrow next to the field indicates a drop-down list of available values. You can select from the list or enter your own value in such fields.
- The asterisk symbol in the *Name* column indicates that the property can have multiple values (for example, *parameter* of an operation). Use commas to separate multiple values.
- A “browse” button next to the field indicates that an editor dialog is available. Multi-line string properties display a multi-line text editor dialog.
- A file chooser button next to the field indicates that the value must be the name of some object or element in the project. Click the button to open a dialog where you can select from available project elements.

Tip Use the keyboard shortcut `Alt+Insert` to edit the fields with file chooser buttons.

Figure 9 Properties Inspector for a class.



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 intranet or on the Internet.

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

View tab

The View tab enables you to set the foreground and background colors of an element selected in the Designer pane. When you start working with Together diagrams, the foreground and background colors are the default system 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. Click the file chooser buttons next to the fields to add values. In this case a separate tag is generated for each value in an Inspector field.
- Checking the box @deprecated creates an empty tag. Click the file chooser button next to the box to add a comment.

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 JavaBeans/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 JavaBeans/C++ Properties:

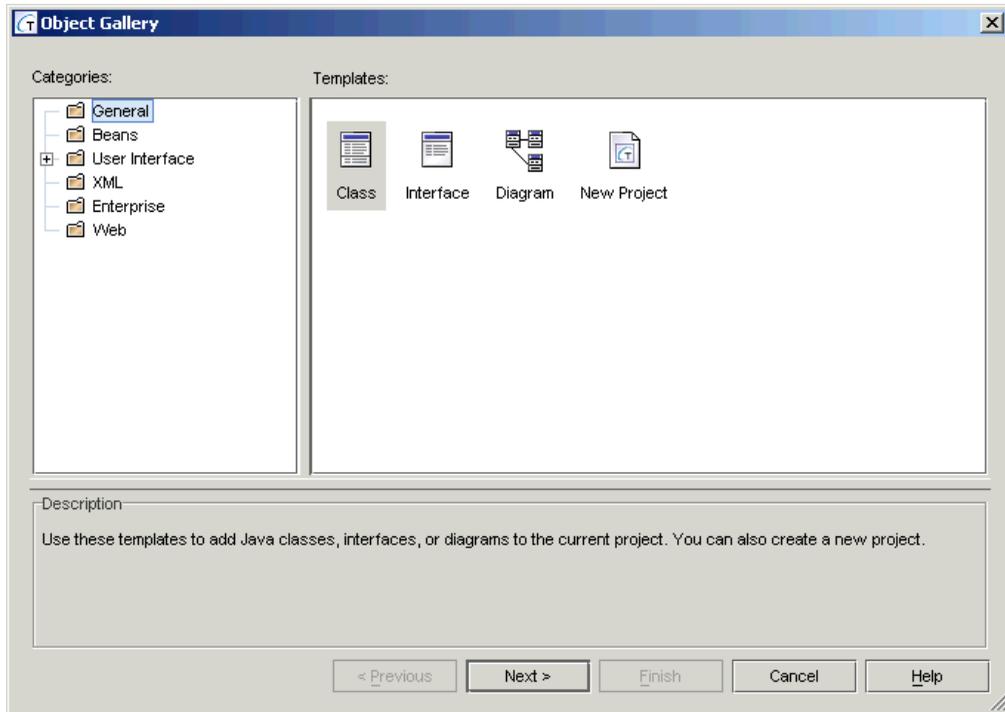
1. Choose **Tools | Options | <level>** from the main menu.
2. Expand the *View Management* node on the left.
3. Choose *JavaBeans/C++ Properties*.
4. Check the properties listed on the right side that are appropriate for your project.
5. Click OK to save and quit the dialog.

Object Gallery

The Object Gallery is a collection of “templates” for creating new files in a project. You choose the type of object you want to create, and specify the location, name, and relevant properties. This creates a skeleton file for you to work with in the current project.

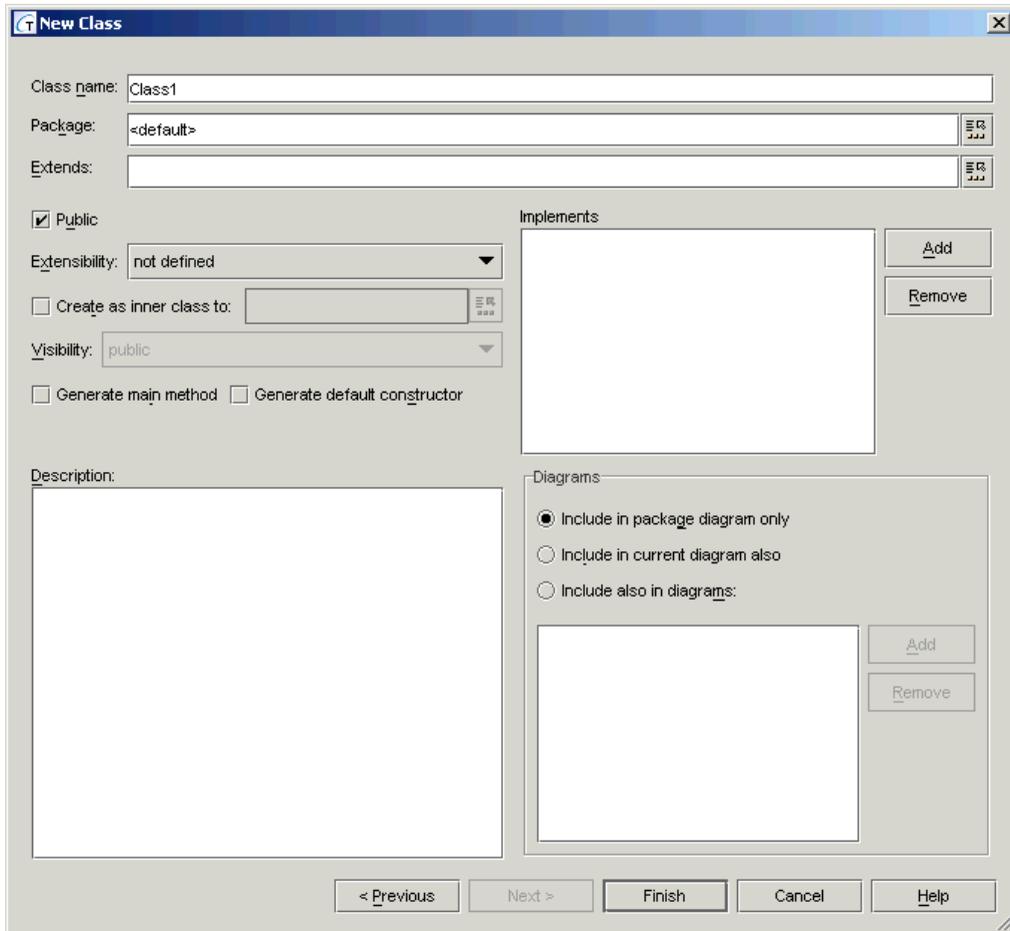
Use the Object Gallery to create diagrams, classes and interfaces, JavaBeans, user interface components to use in the UI Builder, e-commerce objects (EJBs, enterprise and web applications, JSPs, servlets, and so on), XML and HTML files, and others. The types of objects available in the Object Gallery correspond to the project language and also depend on which features are activated. For example, [Figure 10](#) shows the Object Gallery in a Java project that has UI Builder and E-Commerce features activated.

Figure 10 Object Gallery with the *General* category selected.



When you choose an object to create, the next page displays object properties for you to define. Clicking the Finish button creates the skeleton for the object, with the specified properties. [Figure 11](#) shows the page for creating a new Java class.

Figure 11 Specifying properties for a Java class in the Object Gallery.



To work with the Object Gallery:

1. Choose **File | New** to open the Object Gallery. Alternatively, click the New button on the main toolbar or press **Ctrl+N**.
2. Under *Categories* on the left side, select a category of objects to choose from.
3. Under *Templates* on the right side, select a type of object to create.
4. Click the **Next** button or double-click the object icon to go to the next page.
5. Define the object properties as desired. If the Next button is still enabled, click it to access more properties.

For help defining properties, click the **Help** button. This opens the Object Gallery topic in the online help, where you can find detailed documentation on properties for specific types of objects.

6. Click **Finish** to accept the values and create the object.

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 65
- “Using view management” on page 69

Setting up workspaces

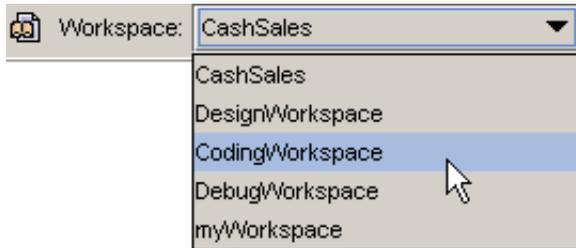
Together enables you to define, save, and recall different arrangements of main window panes. These saved, reusable pane arrangements are called *workspaces*.

Accessing different workspaces

Workspaces are project-specific. Every project has a *default workspace*, which Together creates based on the role of a user. (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 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. Choose **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. Choose **View | Workspaces | Manage Workspaces** from the main menu.
2. Choose the workspace that you want to rename or delete from the list in the resulting dialog.
3. To delete the workspace, click **Remove**.

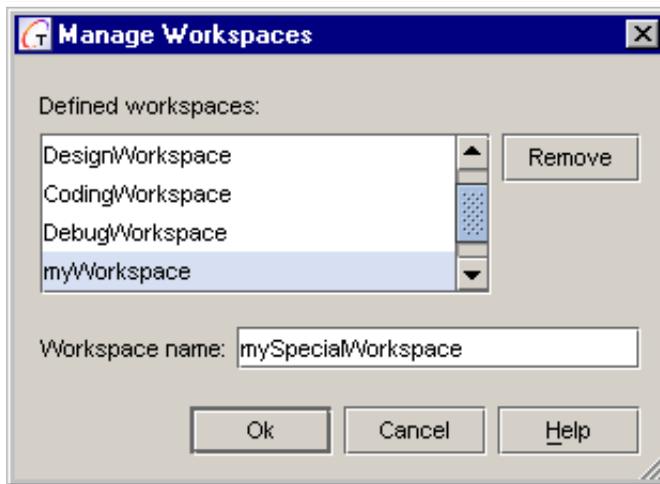
To rename the workspace, enter the new name in the *Workspace name* field.

[Figure 13](#) illustrates changing the name of *myWorkspace* to *mySpecialWorkspace*.

4. Click **Ok** to quit 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. Choose **Tools | Options | Default Level** from the main menu.
2. Choose *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 and save the settings.

With the save option checked, the current state of a workspace is automatically saved when you leave it. If the option is not checked, 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 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. *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 business modeler 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 start Together.

To change your role:

1. Choose **Tools | Options | Default Level** from the main menu.
2. Choose *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 and save the change.

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

Using view management

The Together 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. From the tree, choose *View Management*.
3. For *Diagram detail level*, choose *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. From the tree choose *View Management* 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 subpackage content as package icons with the lists of classes, interfaces, and underlying subpackages. (In the Options dialog, expand *View Management* and choose *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 *View Management*, and select *Show subpackage contents*.
3. Check the box for *Show subpackage contents* to show contents. Uncheck the box to hide contents.

Controlling how JavaBean/C++ properties display in class diagrams

To control how JavaBean/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, expand *View Management*, then choose *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. From the tree, choose *View Management*.
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, expand *View Management*, and choose *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, expand *View Management*, and choose *Sequence diagram*. You can set options for this node. You can also expand *Sequence diagram* 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 `TGH/config/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. If you are the Together administrator and need to configure options on a more global level that impacts all users, see [“Configuring Together for Multiple Users” on page 775](#).

This chapter includes the following topics:

- [“Activating and deactivating modules” on page 73](#)
- [“Overview of configuration levels” on page 74](#)
- [“Setting options in default mode” on page 74](#)
- [“Setting options in advanced mode” on page 76](#)
- [“Viewing and editing options” on page 78](#)
- [“Reference guide to options” on page 80](#)
- [“Common Configuration Tasks” on page 87](#)
- [“Configuring Together for improved performance” on page 91](#)

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 76](#).

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, see [“Configuring Together for Multiple Users” on page 775](#).

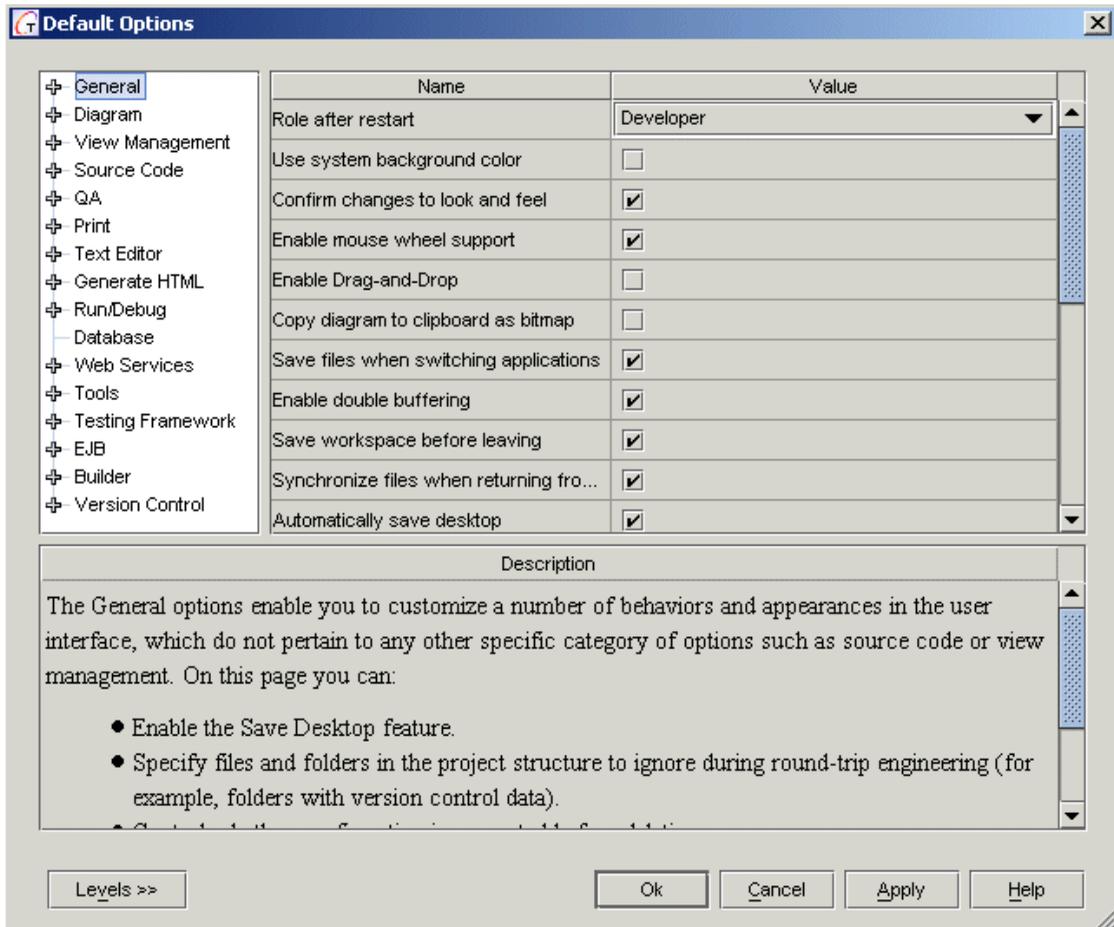
Setting options in default mode

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 78](#). For a reference to the options, see [“Reference guide to options” on page 80](#).
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:

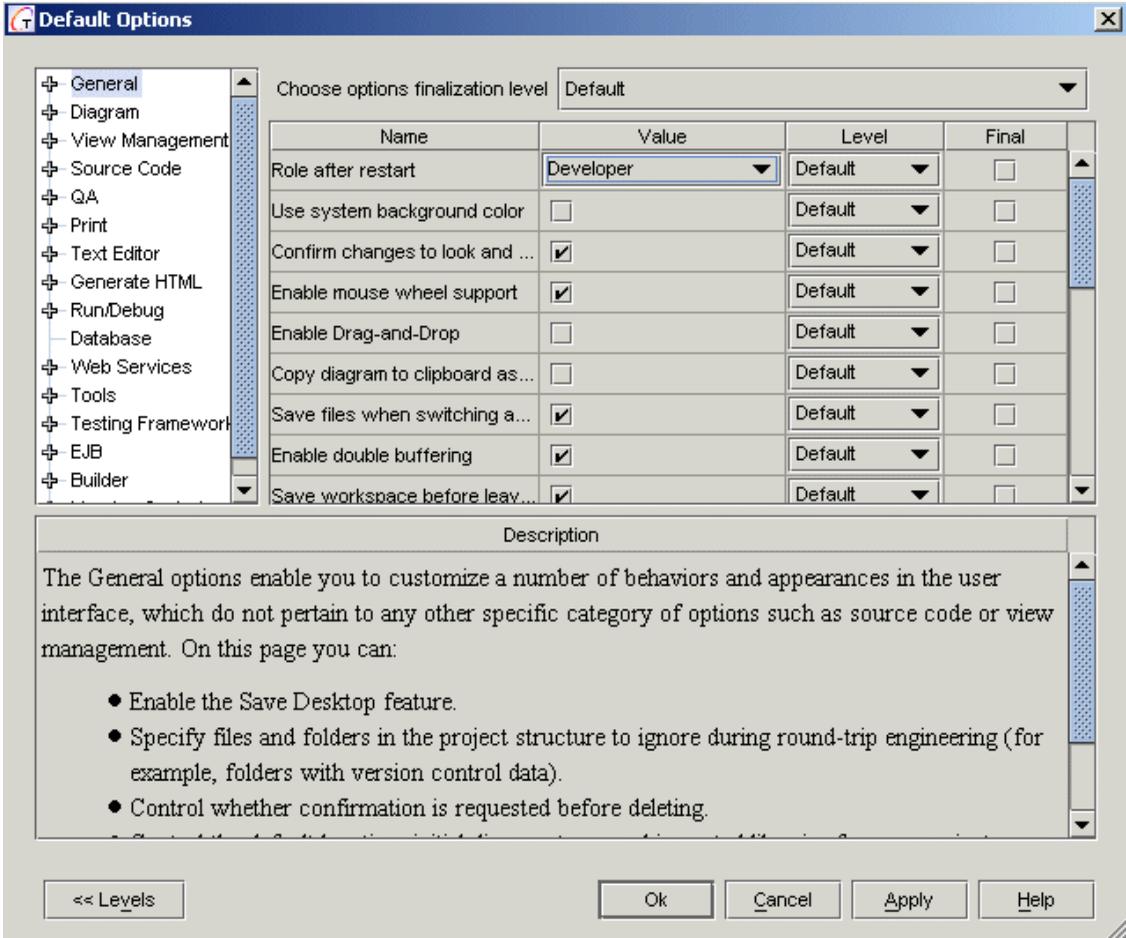
Default = Option is set at the Default level

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 78](#). For a reference to the options, see [“Reference guide to options” on page 80](#).
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 74](#) or [“Setting options in advanced mode” on page 76](#).

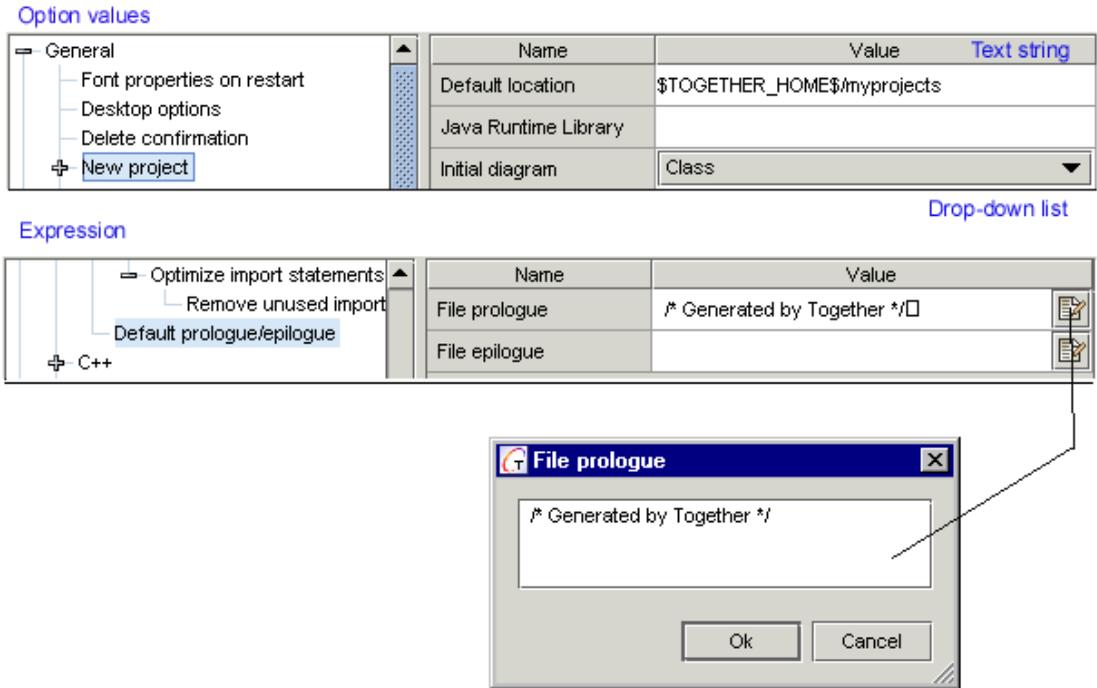
The Options dialog logically groups the options as nodes in a tree. When an option consists of groups of sub-options, you can expand the node to show all available sub-options.

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 hierarchical 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.

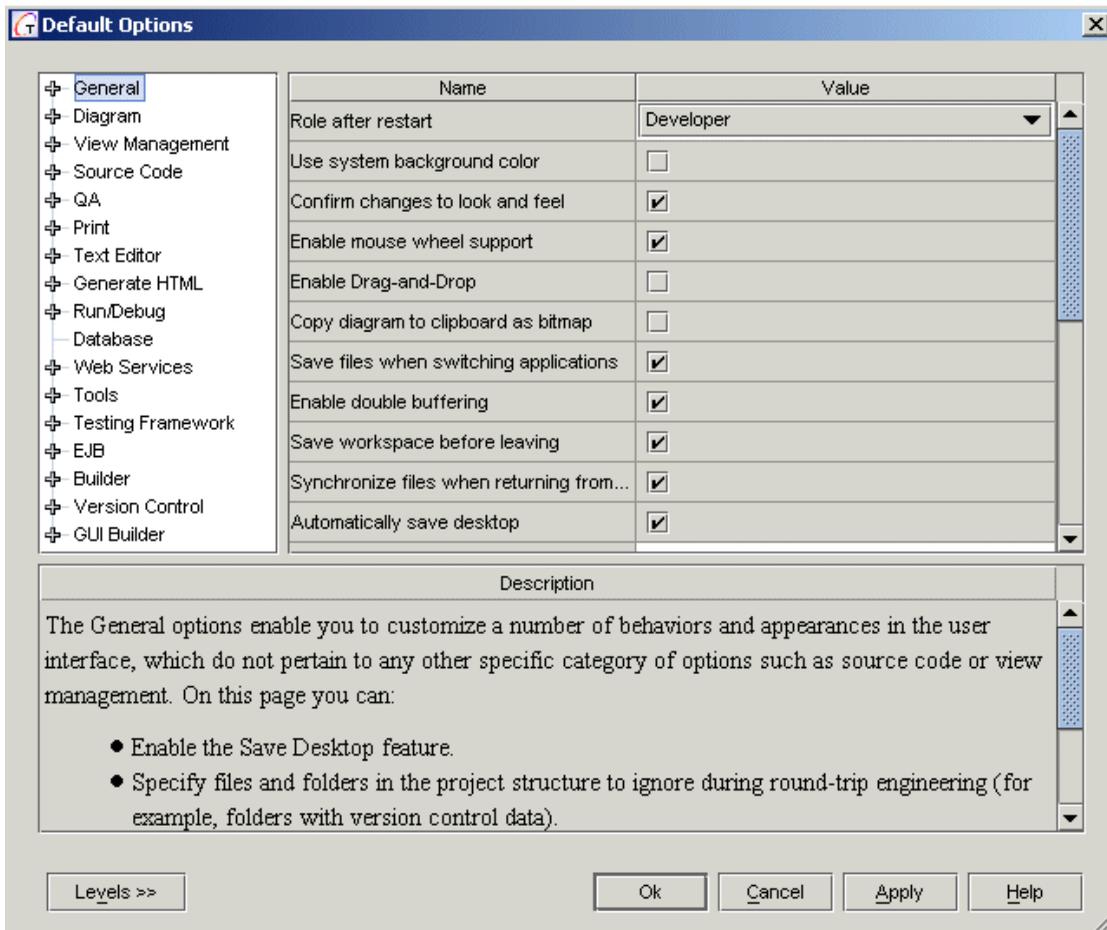
Encoding of the configuration options

Starting with the version 6.0.1, configuration files of Together are written using UTF-8 encoding. It is strongly advisable to edit configuration files via the Options dialog. However, if a configuration file needs to be edited externally, use an editor that supports UTF-8 encoding (for example, Windows 2000 Notepad), and save the file accordingly.

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 11 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 73 for more information.

Table 11 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.	<ul style="list-style-type: none"> • 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.

Table 11 Default and project level options (continued)

<p>Diagram</p>	<p>Control a number of default behaviors and display properties that apply only to diagrams.</p>	<ul style="list-style-type: none"> • 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 Designer pane.
<p>View Management</p>	<p>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 69.</p>	<ul style="list-style-type: none"> • 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.

Table 11 Default and project level options (continued)

Source Code	Control a number of default behaviors and appearances that apply to the formatting of source code during forward and reverse engineering operations.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Specify the path to saved sets of audits and metrics. • Define the applicable scope of Quality Assurance.
Print	Set a number of defaults that apply to printing diagrams, files, and generated documentation.	<ul style="list-style-type: none"> • 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.

Table 11 Default and project level options (continued)

Text Editor	<p>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.</p> <p>Note: Source code formatting is not customized on this node. Use the Source Code options.</p>	<ul style="list-style-type: none">• 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	<p>Control the inclusion/exclusion of various content in the output of the standard HTML documentation generation facility (Tools Generate HTML).</p>	<ul style="list-style-type: none">• 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. <p>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.</p>

Table 11 Default and project level options (continued)

Run/Debug	Customize your runner/debugger.	<ul style="list-style-type: none"> • Define the root directory of the JDK for running and debugging applications from Together. <p style="margin-left: 40px;">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.</p> <ul style="list-style-type: none"> • Define the location of the sources and current working directory. • Specify the project's run configurations. • Specify settings for the debugger. <p>Specify settings for JSP.</p>
Database	Customize a number of common database properties that are used in operations with databases.	<ul style="list-style-type: none"> • 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.

Table 11 Default and project level options (continued)

Web Services	<p>Set the application server that is used by default to register web services.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>	
Tools	<p>Edit various fields to point to appropriate file-based development tools (such as compilers, debuggers, IDEs, and editors).</p> <p>The Tools options contain the tool definitions for your configuration. The External editor tool is pre-defined for you.</p>	
Testing Framework	<p>Customize testing framework options.</p> <p>Note: To view this option, you need to activate the Testing Framework module.</p>	
EJB	<p>Set J2EE defaults</p>	<ul style="list-style-type: none"> • 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 11 Default and project level options (continued)

Builder	Set options for the Builder.	<ul style="list-style-type: none">• 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 and Coroutine classes must be installed (default auto-installed and configured by the Together installer for Windows).	<ul style="list-style-type: none">• 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 11](#) 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 73.

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 795](#).

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 267](#) 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 configure Stereotypes” on page 89](#) 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 89](#). 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 69](#) 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 108](#).

Configuring Together for improved performance

One way to improve performance is to turn off unnecessary features in the Activate/Deactivate Features dialog (see [“Activating and deactivating modules” on page 73](#)). This should make the project load faster and improve performance in general.

In addition, many of the config options affect the performance of Together. To improve overall performance, consider the following tips:

- Uncheck *General: Save files when switching applications* and/or *General: Synchronize files when returning from other applications*. If both options are turned on (checked), task switching is slower. Turn off one or both to improve performance.
- Uncheck *General: Enable Drag-and-Drop* (this is the default setting). The drag-and-drop behavior varies depending on your operating system and whether other performance-sensitive options are enabled, so try turning this option on and off to see whether there are any noticeable differences in performance on your system.
- Uncheck *Diagram: Update link sources when destination changes* (this is the default setting). For each link destination, there could potentially be multiple link sources to check and update. In any case, there is always more updating to process if this option is checked.
- Uncheck *Diagram: Antialias graphics* (this is the default setting). If this option is checked, it slows performance when working with diagrams.
- Uncheck any or all of the options under *View Management - JavaBeans / C++ Properties: Recognize JavaBeans, Recognize C++ Properties, Recognize VBasic Properties*.
- In *View Management - Dependencies*, set *Find between classes* to *None* (this is the default setting). If you need to set this option to *Diagram local* or *All*, set the *Check* option to *Declarations only* to improve performance. In addition, unchecking *Recognize @see tags as hyperlinks* slightly improves performance.
- In *Source Code - Live Source support*, uncheck any programming languages that you do not use.

Also consider the following options, which do not significantly affect overall performance but could make a difference, depending on your specific configuration or tasks:

- *General: Ignore files and folders*. The more files and folders included in the ignore list, the faster project sources can be processed.
- *General: Create design elements as standalone*. A large number of standalone design elements in a project may negatively impact performance. Consider turning off this option if you anticipate creating many design elements. If you will not create a lot of design elements, this option will not affect performance.

- *Diagram: Generate metafiles when saving diagrams*, *Diagram: Show controls for compartments*, and *Diagram - Grid: Show grid*. These options have a minor impact on performance when working with diagrams.
- *View Management - Show options*. The performance impact of these filters depends on the total number of options shown (checked). The more options hidden (unchecked), the better performance. Shown *Dependencies* and *Hyperlinks* have the most negative impact on performance.

The following tips apply to the performance of specific tasks:

- **Generating sequence diagrams.** In the option *View Management - Sequence diagram - Generate Sequence Diagram Options: Depth of call nesting*, set a low value (1 - 3) for the fastest parsing speed. In the same group of options, unchecking *Show full diagnostic* also slightly improves the speed of generating sequence diagrams.
- **Audits and metrics.** Uncheck *QA - Process classes: Process classes from Search/Classpath* for significantly faster audits and metrics processing (this is the default setting). You should check this option only if you are working with a C++ project in which some of the project sources are in the Classpath instead of in the model.
- **Generating HTML documentation.** Uncheck *Generate HTML: Update package dependencies* to generate documentation faster (this is the default setting).
- **Java debugging.** If you do not need to view diagrams during debugging, uncheck *Run/Debug - Debugger: Animate class diagram during debugging* (this is the default setting). This makes it slightly faster to access and navigate debugging results because Together does not open diagrams before opening results.
- **Java compiling.** Uncheck *Builder - Built-in Javac: Reflect compiling process in status bar* to compile faster (this is the default setting). Unchecking any or all of the following options also slightly improves compilation time:
 - Builder - Built-in Javac: Show javac output in full format*
 - Builder - Built-in Javac - Compiler options: Generate all debugging information*
 - Builder - Built-in Javac - Compiler options: Verbose output*
- **Version control - CVS integration.** Check *Version Control - CVS: Compress traffic* (this is the default setting). This option improves performance particularly when using a remote server.

CREATING AND MANAGING PROJECTS

- Chapter 5, “Working with Projects”
- Chapter 6, “Reverse Engineering”
- Chapter 7, “Importing and Exporting Information”
- Chapter 8, “Generating makefiles”

Working with Projects

Almost all the work in Together is done 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 95
- “Creating new projects” on page 97
- “Editing project properties” on page 101
- “Running multiple project instances” on page 107
- “Setting-up version control for projects” on page 108
- “Setting up large projects” on page 108

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 33](#).

Understanding Project Basics

To begin modeling with Together, you need to create 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 74](#)).

When creating a project, select a new or existing directory for the primary root directory. By default, Together synchronizes 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 109](#).

The scope of a project is not limited to a single root directory. Instead, you can 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 synchronization with source code.

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, but 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 4, “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 101](#) for more information.

There are two ways for creating projects.

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 98](#) or [“Using the New Project dialog” on page 100](#).

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 108 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 12](#), clicking **Next** to proceed to the next step.
3. Click **Finish**.

Table 12 Information requested by the New Project Expert

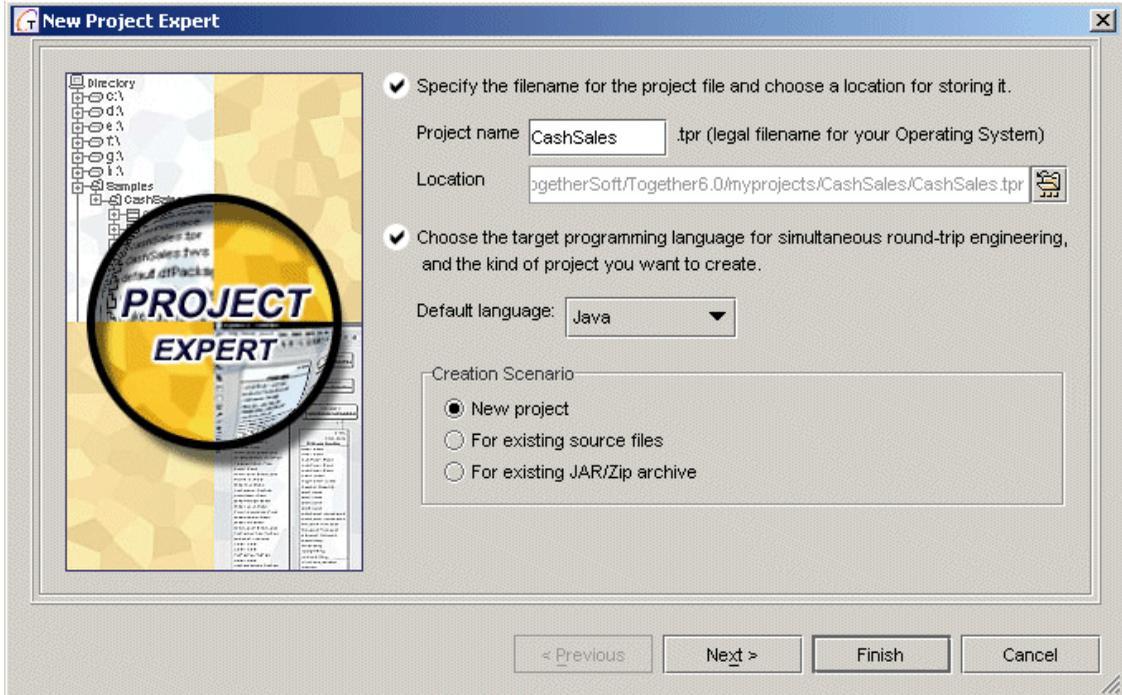
Field name	Description/Notes
Project name	Filename for the project file
Location	Location for the project file; Select the default or browse and select a new location
Default language	<p>Target programming language for simultaneous round-trip engineering and the project you want to create</p> <ul style="list-style-type: none"> • Java • Design (see note below) • Visual Basic 6 • C++ • C# • CORBA IDL • VisualBasic.Net <p>Note: The simultaneous round-trip engineering feature does not apply to Design language. That is, if you select Design as the default language, Together does not generate source code for the model.</p>
Creation Scenario	<p>New project: Creates a new project without existing code</p> <p>For existing source files: Creates a new project using existing code</p> <p>For existing JAR/Zip archive: Creates a new project with existing JAR/Zip files</p>

Table 12 Information requested by the New Project Expert

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	<ul style="list-style-type: none"> • Check or uncheck Standard libraries and/or Class path • Add or remove paths or archives

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.

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

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

Alternatively, choose **File | New** to open the Object Gallery, select the **New Project** icon, and click **Next**.

2. Complete the information required by the New Project dialog, as listed in [Table 13](#).

Table 13 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).

Table 13 Preliminary information required by the New Project dialog

Initial diagram	Type of diagram that Together will generate along with the new project; Select the type of diagram or None.
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).

3. 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.

4. 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 103](#)
- Removing resources, as detailed on [page 105](#)
- Setting resource options, as detailed on [page 105](#)
- Setting up version control for the project, as detailed on [page 108](#)

5. 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 101](#).

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.

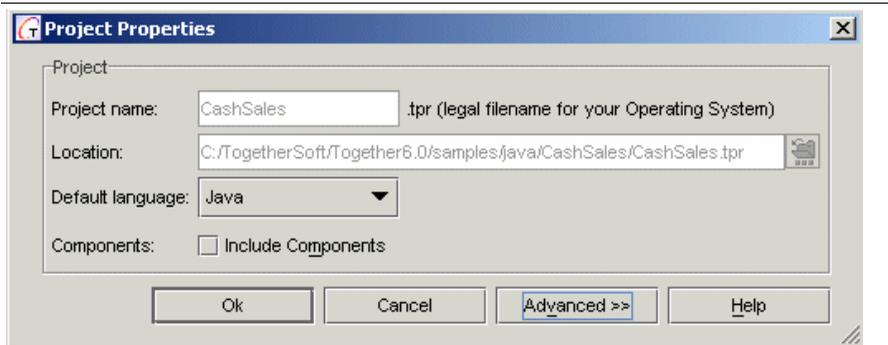
To access the Project Properties dialog, follow these steps:

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](#).

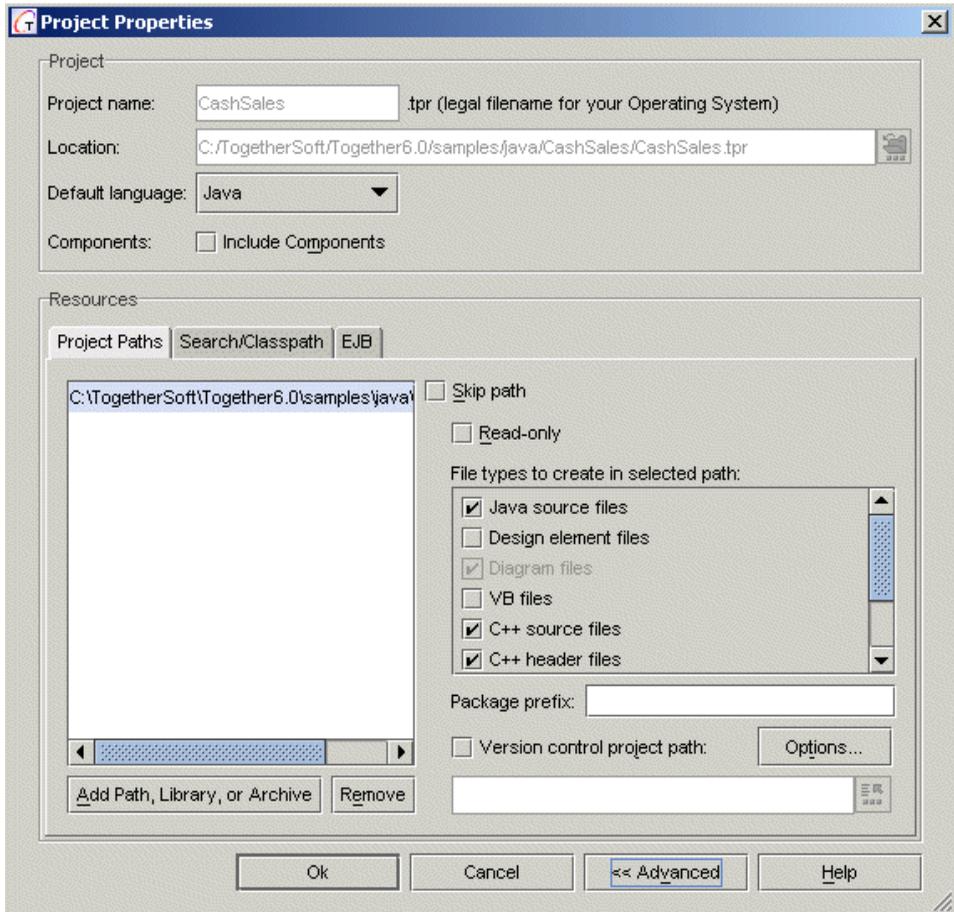
Note Project Properties shows the same fields as the New Project dialog. For a description of the fields, see [Table 13, “Preliminary information required by the New Project dialog”](#) on page 100.

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 103
 - [“Removing Resources”](#) on page 105
 - [“Setting Resource Options”](#) on page 105
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 101](#); or
- Accessed the New Project Dialog, as described in [“Using the New Project dialog” on page 100](#)

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.

For Java projects, the JDK Configuration tab contains information about the current JDK configuration. Choose the preliminary prepared JDK configuration from the drop-down list while creating a new project. To figure out how to create a new JDK configuration, see [“Creating a new JDK configuration” on page 106](#).

For J2EE projects, choose the current application server using the EJB tab.

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 105](#) 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.
- Sometimes the order of resources listed in the Search/Classpath is very important. You can move the members of the list up and down using the arrow buttons right.

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 listed in the JDK Configuration tab along with JDK configuration libraries. 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.

To reorder libraries and paths in the Search/Classpath tab, use the arrow buttons.

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 101](#); or
- Accessed the New Project Dialog, as described in [“Using the New Project dialog” on page 100](#)

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 101](#).

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 101](#); or
- Accessed the New Project Dialog, as described in [“Using the New Project dialog” on page 100](#)

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 108](#).

Creating a new JDK configuration

For Java projects, you must invoke some JDK x.x. You can create different JDK configurations and then use them if needed.

To create a new JDK configuration:

1. From the **Tools** menu, choose **Configure JDK**. The *JDK Configurations Editor* dialog opens.

Note *Configure JDK* is an activated feature. If you don't see *Configure JDK* in the list of *Tools* options, choose **Tools | Activate/Deactivate Features**, and check **JDK Configuration** in the *Activate/Deactivate Features* dialog.

2. Type the full path to the %JAVA_HOME% in the *Directory to start search* field, check *Perform search recursive* if necessary, and click **Search**.

For filling the *Directory to start search* field, you can alternatively use the file chooser.

3. If `java.exe` is successfully found out using the new path, the corresponding line appears in the list of the available JDK configurations.
4. To remove the useless JDK configuration from the list, click **Remove**.

To choose the JDK configuration as a default one:

1. Highlight the chosen line in the list of the available configurations and click **Set Default**. The default JDK home and the default JDK version are displayed. The corresponding `rt.jar` file appears in the list of the default JDK libraries.
2. Use the *Add/Remove* buttons to add/remove the additional libraries if needed.

Note The default value of the JDK configuration after installing Together is the bundled JDK located in `%TG_HOME%/jdk`.

To edit a JDK configuration, you can also use the *Edit JDK Configuration* button in the Project Properties dialog.

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 22, “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 97](#) 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 101](#).
- 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

A central feature of Together is LiveSource — the ability to immediately synchronize class diagrams with the implementation code.

LiveSource 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 the visual model. There is no intermediary repository, no batch code generation, and no risk of losing code.

The LiveSource feature applies to existing code as well as code that is being developed. Together can *reverse engineer* source code, building a model around existing code or restoring a model from archived files.

This chapter includes the following topics:

- [“Reverse engineering techniques” on page 111](#)
- [“Recognizing links during reverse engineering” on page 114](#)
- [“Reverse engineering archive files” on page 115](#)

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

To create a new project with existing code, use the New Project dialog. You can specify the use of existing source files or JAR/ZIP archives.

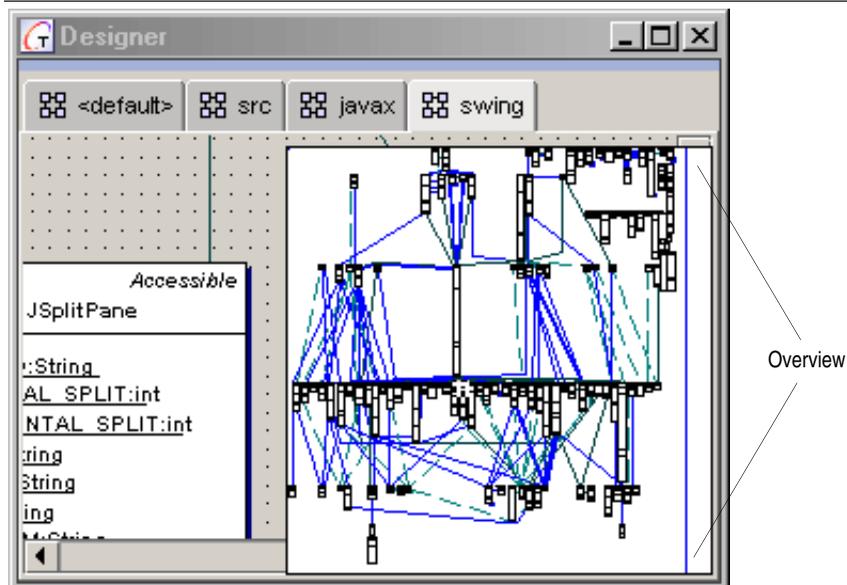
To reverse engineer code with the New Project dialog:

1. Choose **File | New Project** from the main menu. The *New Project* dialog opens.
2. In the *New Project* dialog, enter the project name and location. For the initial diagram type, select **Class**. Also, select a default language.
3. Click **Advanced**.
4. Choose the **Project Paths** tab.
5. Click the **Add Path or Archive** button and then choose the file that contain the source code that you want to use. Repeat this step to choose additional files if necessary.
6. Click **Finish**. Together uses the code of the specified files to create the new project.

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

Figure 21 shows the results from reverse engineering JDK 1.3. The Designer pane displays the swing library. The overview, as shown, 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* dialog. In that case, however, the physical class diagram files are stored in the same folders as the source files.

Alternatively, you can use the *New Project Expert* for reverse engineering.

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.

Incorporating code into an existing project

Together can reverse engineer code that you can incorporate into an existing project. The code can be source code, *.jar files, or *.zip files.

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

1. Open the project.
2. Select **Project | Project Properties** from the main menu.
3. In the Project Properties dialog, choose the **Project Paths** tab.
4. Click the **Add Path or Archive** button and then choose the file that contain the source code that you want to use. Repeat this step to choose additional files if necessary.
5. Determine if Together should treat the code as read-only (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.

6. Click **OK**. Together incorporates the codes into the specified project.

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.

Alternatively, to add existing Java code to an existing project:

1. Copy the necessary Java files to the project source path directory. If you are copying EJBs, copy Java code of their implementation classes, home and remote interfaces.
2. Correct the names of packages in the incorporated Java code.
3. Reopen your project. The design elements corresponding to the incorporated code appear on the diagrams according the packaging. *Generalization* and *Association* links between elements appear, as well.

Recognizing links during reverse engineering

Code generated from round-trip engineering is defined by codegen options and patterns (including textual templates). Such options and patterns define the creation of the implementation code of UML diagrams.

Reverse engineering implies that the source code is analyzed by the parser. As such, encountered classes are passed to the model as is. In contrast, links are created by the following source code elements:

- **Attributes.** A special engine parses the attributes of a class, selects certain tokens from the attribute string, and chooses the appropriate blueprint allowing the recognition of a link with its properties. In particular, the link type and destination class are recognized and the attribute modifiers are ignored.
- **Tags in doc comments.** Define links and optionally, their properties.
- **Links as comments.** Not visible to the compiler, as they are commented.

The configuration system of Together contains these *parser blueprints*. They are used for the automatic recognition of links in classes during reverse engineering and actually determine which elements of the source code produce links.

The blueprints are defined in the configuration files for the supported languages as follows:

```
$TGH$/config/<language>.config
```

The format of a blueprints is:

```
parser.<language>.blueprint.link.<link type> = <link expression>
```

Each attribute declaration or design comment is compared against the blueprint pattern and if recognized, forms a link of the corresponding type.

The first matched blueprint is used. Optionally, the blueprints can include additional sub-properties (tags), in the following format:

```
parser.<language>.blueprint.link.<link type>.<link property> = <property value>
```

The <link type> is not actually used, but recommended to make the description of parser blueprints meaningful.

Note Tag values, directly specified in the Inspector, override those from the blueprint.

You can define your own blueprints with their sub-properties, specifically designed for certain applications. However, the parser engine scans the entire set of existing blueprints. Thus, too large a number of custom blueprints can impact performance.

Reverse engineering archive files

Together extends reverse engineering to J2EE archive files. It is possible to restore the diagram structure of 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 Archive**. The J2EE Module Import dialog appears.
2. For *J2EE module location*, specify the fully-qualified name of the source archive file to be imported. The archive file extension should be .jar, .war, or .ear.
3. For *Extracted module location*, enter the fully-qualified path to the target folder where the new diagram will be created. Note that the destination folder should not be the parent directory of the project source path.
4. Click **Ok**. Together restores appropriate diagram.

Note The Ok button is disabled unless archive type and destination directory are properly specified. Use file chooser buttons to define the required locations in the file system.

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

Tip If you are importing a module with JSP files, mark the files as JSP in the deployment descriptor so that Together restores them as JSP files. Otherwise, Together treats them as ordinary web files.

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 117
- “Importing and exporting JDBC database information” on page 118
- “IDL Export and LiveSource” on page 121
- “Importing and exporting XMI models” on page 122
-

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 14, “XML Modeling.”](#))
- Export/Import from class diagrams and ER diagrams to DTD. (This is covered in [Chapter 14, “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
- InterBase 7
- JDataStore 6 Server
- 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, you can 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 scripts only or generating the DDL scripts and then running them in the same operation.

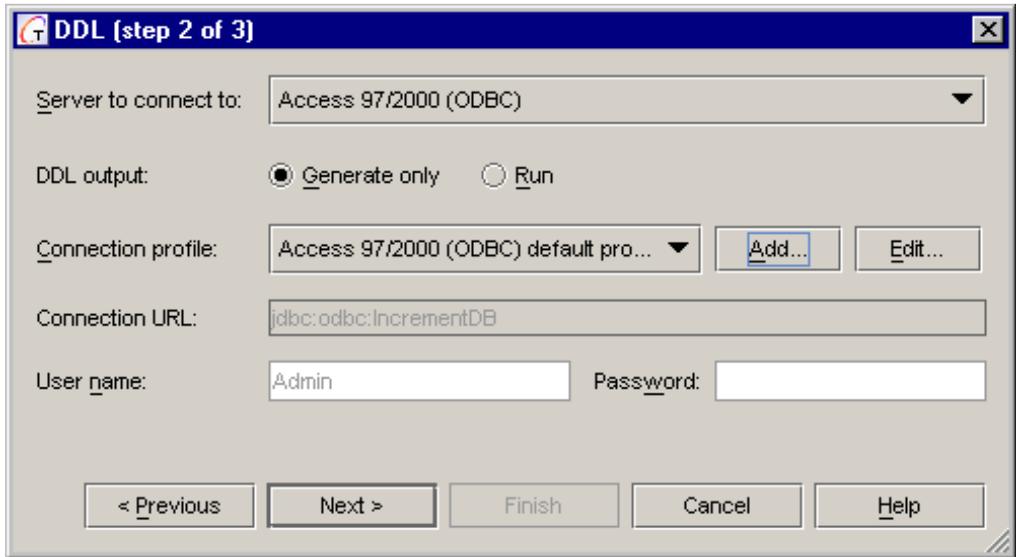
To generate DDL:

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 script 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 script. Click **Finish** to generate the DDL script.

If you select **Run** from the **DDL output**, File name field is not available. In some cases (for example, for MS SQL Server), you can choose Schema name.

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.)

Important MySQL does not support non-English characters.

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.

Tip Different kinds of import are available for the languages other than Java. For example, if the default project language is C++, then it is not possible to import to EJB; for the VB6 projects a DB schema can be imported to ER diagram only).

4. Click **Finish** to complete the import.

IDL Export and LiveSource

If a Together project contains class diagrams appropriately structured to support IDL, you can generate IDL for specific diagrams, classes, packages, or for the entire Together project. LiveSource enables you to generate IDL synchronized with your diagrams.

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 118.

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.

LiveSource support for IDL

Together supports LiveSource 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.

To create a model on the base of the existing IDL source code:

1. On the main menu, choose **File | New | New Project**.
2. In the **Project Path** tab, click the button **Add Path, Library or Archive**.
3. Select the root directory that contains the source IDL code.
4. Click **OK** to start creating the new project around the specified IDL source.

Result: the newly created project contains the model, generated on the base of the specified IDL.

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 118.](#)) 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 the 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 target location and specify the filename for the generated XMI file.
 - Select encoding of the output stream from the drop-down list.
 - 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.

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

When exporting a model to Rational Rose, keep in mind that some of the elements supported by Together are not allowed in Rose. These are, for example, objects on the Class, Activity, State, Component or Deployment diagrams, or components and interfaces on the Deployment diagram. The second option of the Export to XMI dialog (XMI 1.1 for UML 1.3 Unisys Extension, Recommended for Rose) provides the ability to drop all Rose-inconsistent elements. If such elements are encountered, appropriate messages appear in the Message pane. For example: "The following element is not exported due to Rational Rose limitations: Object aSale in ClassDiagram Object View"

User configurable translations

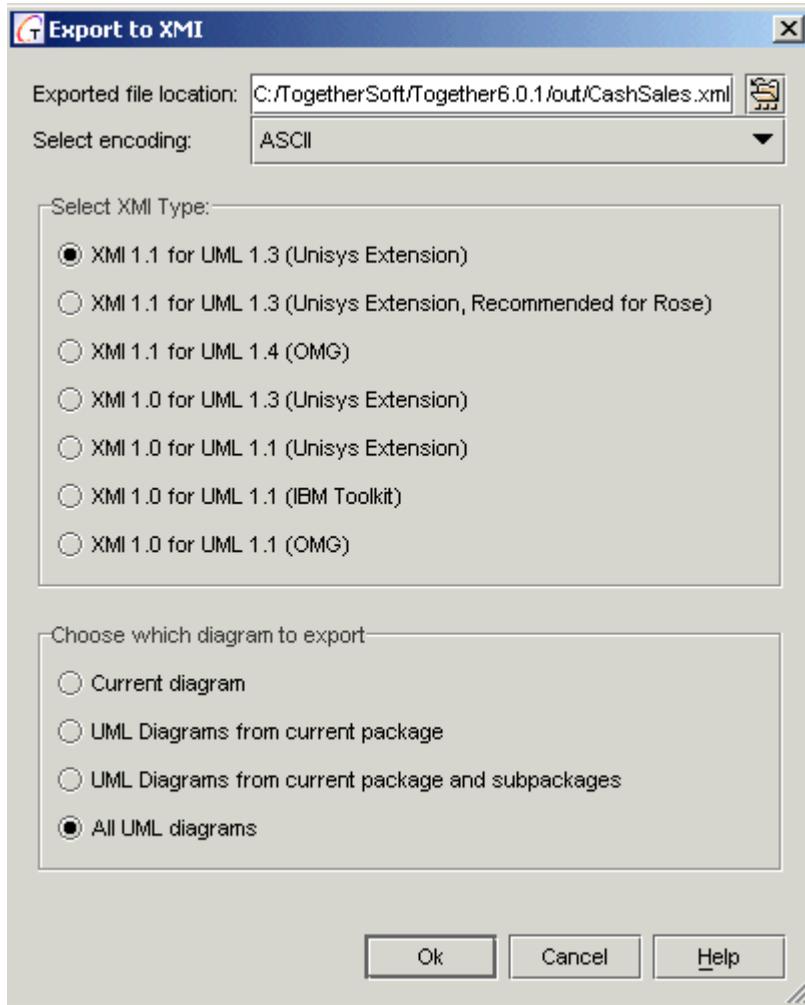
To support compatibility between Together model tags and XMI tags, the interchange module provides a list of translations that contain entries in the format:

```
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
```

Export to XMI dialog box



Generating makefiles

Together provides the possibility to generate a makefile for your project. This feature is especially valuable 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.

A makefile contains all necessary command line commands and parameters, required for running your project independently. You can generate a makefile on various levels: for a project, a package, or even a class. The default Makefile destination is specified in the *Builder* node of the Options dialog, and you can specify any other location on the required level.

This chapter includes the following topics:

- [“Configuring the makefile destination” on page 125](#)
- [“Generating a makefile” on page 126](#)

Configuring the makefile destination

To configure the makefile destination:

1. On the main menu, open the Options dialog (**Tools | Options - <level>**) and select the *Builder* node.
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.

Generating a makefile

To generate a makefile:

1. On the main menu, choose **Project | Generate Makefile**.

Note To generate a makefile for the selected object, choose **Selection | Tools | Generate Makefile** on the main menu, or the object right-click menu in the Designer pane or on 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 editing.

MODELING WITH TOGETHER

- Chapter 9, “Introduction to Modeling”
- Chapter 10, “Working with Diagrams”
- Chapter 11, “UML Modeling”
- Chapter 12, “UML Extension Diagrams”
- Chapter 13, “Real-Time Modeling”
- Chapter 14, “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 benefits inherent to modeling by fully synchronizing diagrams and 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 129](#)
- [“Diagrams supported in Together” on page 130](#)

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 14](#) 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 14](#). For instructions that apply to all diagrams such as creating, editing, and hyperlinking diagrams see [Chapter 10, “Working with Diagrams”](#).

Table 14 Diagrams supported in Together

This section...	Includes these diagrams...
-----------------	----------------------------

Table 14 Diagrams supported in Together (continued)

Chapter 11, "UML Modeling"	"Use case diagrams" on page 166
	"Class diagrams" on page 168
	"Sequence and collaboration diagrams" on page 175
	"Statechart diagrams" on page 184
	"Activity diagrams" on page 188
	"Component diagrams" on page 190
	"Deployment diagrams" on page 192
Chapter 12, "UML Extension Diagrams"	"Entity relationship diagrams" on page 195
	"Business process diagrams" on page 200
	"Robustness analysis diagrams" on page 201
Chapter 13, "Real-Time Modeling"	"Use case diagrams" on page 208
	"System context diagrams" on page 210
	"System architecture diagrams" on page 214
	"Statechart diagrams" on page 217
	"Event sheet diagrams" on page 217
Chapter 31, "J2EE Support"	"Interaction diagrams" on page 219
	"EJB assembler diagram" on page 556
	"Enterprise application diagram" on page 556
	"Taglib diagram" on page 557
	"Taglib diagram" on page 557
"Resource adapter diagram" on page 557	

Working with Diagrams

This chapter outlines basic techniques for working with diagrams. It covers the following topics:

- [“Creating diagrams in projects” on page 133](#)
- [“Drawing diagram elements” on page 136](#)
- [“Working with elements and links” on page 143](#)
- [“Viewing diagrams and managing diagram layout” on page 148](#)
- [“Editing diagrams” on page 150](#)
- [“Hyperlinking diagrams” on page 153](#)
- [“Annotating diagrams” on page 157](#)
- [“Standalone design elements” on page 158](#)
- [“Saving and copying diagram images” on page 159](#)
- [“Printing diagrams and source code” on page 160](#)

Creating diagrams in projects

Diagrams exist within the context of a project. You create or open a project before creating any new diagrams. If you create a new 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 3, “Setting Your Personal Preferences” on page 65](#)).

Together diagrams fall into two basic categories:

- UML diagrams (class, use case, sequence, and so on)

- Special Together diagrams (entity relationship, EJB assembler, business process, and others)

Some diagrams are source-generating. These are: class diagrams, sequence and collaboration diagrams, and some J2EE-related diagrams. The contents of such diagrams are synchronized with the source code. Right click a class or interface symbol on a diagram and choose Edit to open the source code in the editor. Selecting a class symbol in the diagram opens the underlying class in the special tab of the Editor, marked with the class name. If the class is read-only, the tab is also marked with the lock icon. Selecting an element within a class or interface symbol automatically navigates to the appropriate line of the source code.

When you begin a new project or add new diagrams to an existing project, you can create diagrams using one of the following:

- Main menu or toolbar
- Hyperlinking feature
- 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 diagrams

To create a new diagram in a project:

1. Select the destination directory or package on the Directory or Model tab in the Explorer.
2. Choose **File | New** on the main menu to open the Object Gallery.

Tip Alternatively, click the New button on the main toolbar.

3. In the *Templates* pane of the Object Gallery, choose the **Diagram** icon and click **Next** to open the New Diagram dialog.
4. Choose the **UML** or **Together** tab as required, and click the icon for the type of diagram you want to create.
5. In the *Diagram name* field, type a name for the diagram.
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 a symbol representing a logical link to the new diagram.
8. Click **Finish**. The new diagram opens in the Designer pane.

Using the hyperlink feature to create diagrams

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

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.
3. On the Selection menu, choose **Hyperlink To | New Diagram**. The New Diagram dialog opens.
4. Specify the type and location for the new diagram as described above. Click **Ok**.

For more information, see [“Hyperlinking diagrams” on page 153](#).

Cloning diagrams

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

To clone a diagram:

1. On 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 Model tab of the Explorer.

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, and make sure the Inspector is open.
2. Click on the background to display diagram properties in the Inspector.
3. Modify the *Name* property in the diagram Inspector.

To rename a diagram in place:

1. On the Model tab of the Explorer, select the desired diagram node.
2. Double-click 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 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.

To configure the diagram settings:

1. Choose **Diagram Options** on the diagram right-click menu. The Options dialog opens on the Diagram level.

Tip Alternatively, choose **Tools | Options | Diagram level**.

2. Change the configuration level if necessary.
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, note the options that help improve graphics.

3D look

To make your diagram look three-dimensional, open Diagram Options and check *3D look*. This draws a shadow 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 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*.

Drawing diagram elements

This section covers the basic techniques for placing diagram elements and annotations into diagrams and drawing relationship links between them.

When you create a new diagram, the Designer pane presents an empty background. You place the various elements (such as class, use case, and so on) on the background and draw relationship links between them according to the requirements of your model.

The main tools for drawing and editing diagram elements are:

- **Diagram toolbar buttons.** Use toolbar buttons to place elements on the diagram and draw links to connect them.

- **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, configure diagram-level options, and update diagrams and hyperlinks.
- **Element right-click menus.** The right-click menus of the various elements and links provide commands 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, and to create hyperlinks to diagrams, elements of diagrams and other diagrams, and to diagrams or elements and files or URLs. You can also edit annotations and source code comments.

Undo/Redo

All operations are reversible. Undo and Redo commands can affect the visual diagram, layout of elements, addition 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.

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

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 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 and select the *Grid* node.
3. Set the *Show grid* and *Snap to grid* options as desired.
4. Specify the granularity of the grid in the fields *Grid width* and *Grid height*, in pixels.

Drawing elements

To place an element on a diagram:

1. 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.
2. Click on the diagram background in the place where you want to create the new element. This creates the new element and activates the in-place editor for its name. (Note: To place multiple elements of the same type in a single operation, see [“Adding multiple elements” on page 139.](#))

Tip Alternatively, right-click the diagram background 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.

Configuring element symbol size

When an element is created, its dimensions are set automatically. The maximum and minimum allowed sizes are pre-defined in the view configuration file (`TGH/config/view.config`).

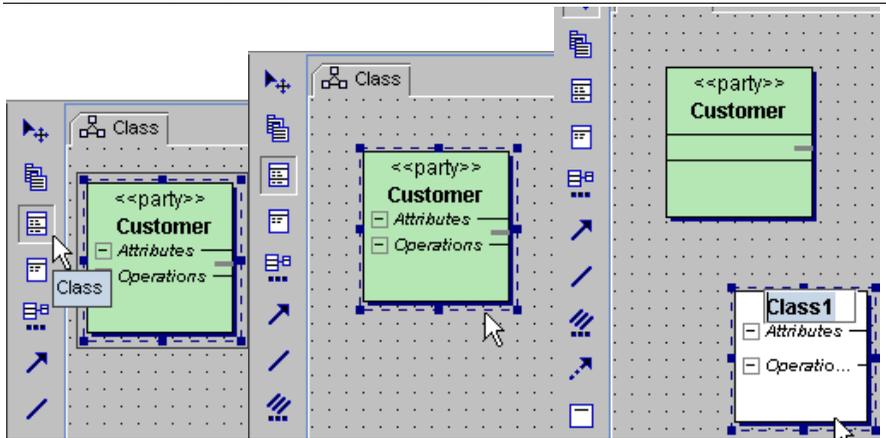
The maximum possible size of an element is configurable in the *Diagram* category of the Options dialog.

To set the maximum size of diagram elements:

1. On the main menu, choose **Tools | Options | <level>- Diagram**.
2. On the Diagram page, choose the field *Maximum width of icons*, and type in the value.
3. Click **OK** to apply the change and close Options dialog.

The size of a diagram element cannot exceed the specified maximum size. If this is the case, the elements are automatically truncated. Thus, the maximum width should not be too small.

Figure 23 Steps for placing an element on a diagram



You can edit the minimum allowed size of a node element in the `view.config` file. If a node is decreased to a smaller size, it will automatically return to the dimensions specified in the view configuration.

If necessary, you can edit the `view.config` file and specify your preferred value of the minimum dimensions.

To set the minimum size of diagram elements:

1. Open the file `TGH/config/view.config`.
2. Find the following line:

```
setLayoutConstraints(minWidth(16),minHeight(16),preferredWidth(50),maxWidth(maxWidth));
```

3. Type in the new values, and save changes.
4. In the main menu of Together, choose **Tools | Options | Reload changed config files** to apply the changes.

You can resize element symbols by dragging the borders to the desirable size, regardless of the above-mentioned setting. However, you can also return an element to its initial size.

To return an element to its initial size:

1. Select the element on the diagram.
2. On the right-click menu, choose **Layout | Actual Size**. The initial size of an element is restored.

Adding multiple elements

You can place several elements of the same type on a diagram without returning to the toolbar. Each will have a default name that can be edited in place or in the Inspector.

To create multiple elements:

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

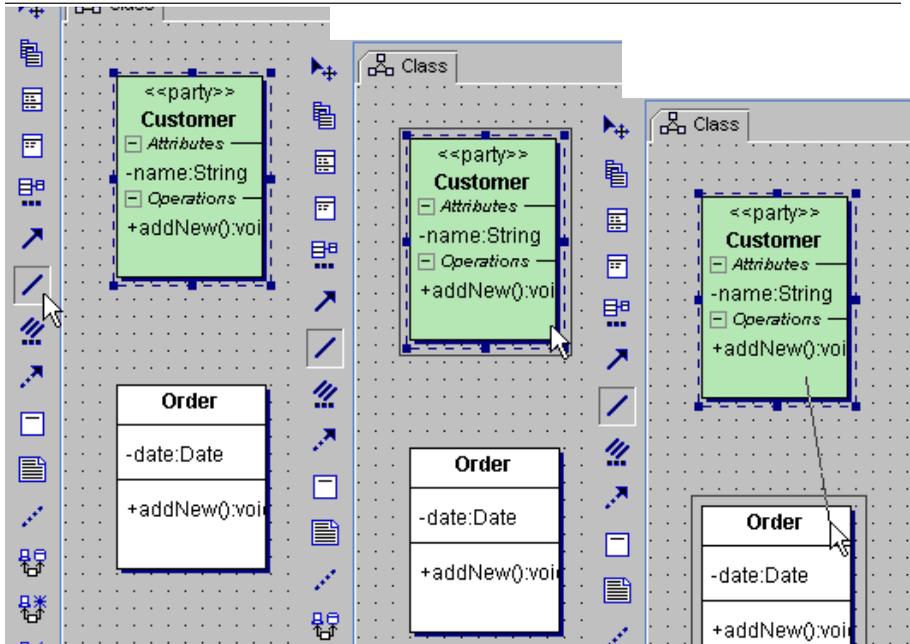
Tip After making a selection on the toolbar or doing the first of a multi-draw or multi-placement operation, cancel the operation by clicking the Select arrow on the toolbar.

Drawing relationship links

To draw a relationship link between two elements:

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 and drop when the second element is highlighted, as shown in [Figure 24](#).

Figure 24 Steps for linking elements



Tip If the destination element is out of reach, there are two ways to 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 open the Choose Destination dialog, and choose the destination element from the list. (Note: To place multiple links of the same type in a single operation, see ["Adding multiple elements" on page 139.](#))

Drawing links with bending points

If your diagram is densely populated, you can 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. Click on the destination element to terminate the link.

Tip To configure the links to bend at right angles, choose *Rectilinear* in the Diagram options (**Tools | Options | <level> - 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 from the element to itself.

Drawing dependency links

Together automatically shows 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 a dependency such as implementation, inheritance, or usage. Only actual dependencies are taken into account. For example, if a certain class is imported but never actually used, this is not considered a dependency.

This applies only to source-code based elements. Inter-package dependencies between elements such as objects, notes, search-path imported classes, use cases, and so on are not recognized.

To draw dependency links between packages:

1. Select one or more packages in the diagram.
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* (be prepared to wait - this builds links for the entire project).

Once built, the dependency links on a diagram are not updated. Obsolete or even non-existent dependencies are displayed until they are deleted or created anew. To update a link individually, select it and choose **Rescan** on the right-click menu for the link.

The process of building dependency links can be 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 complicated. Together provides a special filter for hiding the autodependency links:

`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. To change the behavior, edit the `filter.config` file.

Placing shortcuts on diagrams

You can create a shortcut to another diagram or to an element of another diagram. This is useful, for example, when you want to draw an association from a class on one diagram to a class on another. A shortcut to another diagram can reveal a relationship between the two diagrams.

To create shortcuts to other diagrams or diagram elements:

1. Right click the background of the current diagram and choose **New | Shortcuts**.
2. In the resulting dialog, locate what you want to create a shortcut to.
3. Click **Add** to add it to the list of existing or ready to add elements. Repeat this for all the shortcuts you want to make.
4. Click **Ok** to create the shortcut and close the dialog.

Tip You can also create a shortcut to an element by right clicking the element in the Model tab of the Explorer and choosing Add as Shortcut.

A shortcut to a diagram displays as a package that shows the icon of its diagram type and its diagram name. The package lists the elements of the diagram.

A shortcut to a diagram element displays with a shortcut symbol in the lower left corner. (This includes shortcuts to other packages.)

If a shortcut is to a diagram or to a package, you can open the diagram or package represented by the shortcut by double clicking. Alternatively, you can right click and choose Open or Open in New Tab.

Working with elements and links

Most manipulations with diagram elements and link involve dragging the mouse or executing right-click menu commands on the selected elements.

Selecting elements

Click on any element in the diagram to select it.

- To select multiple elements, hold down the **CTRL** key and click on them individually. Alternatively, click on the background and drag a lasso around an area to select all the elements it contains.
- To use a “rubber-band box” when working with diagrams in the UI Designer, hold down the **Shift** key and drag the mouse cursor around an area to select its elements.
- 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 the element in the diagram. Scroll the view to make it visible.

Right-click menus

Many operations on 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. Using the right-click menu of an association link, you can specify the following:
 - Type (association, aggregation, or composition)
 - Cardinality of the client by right-clicking near the client
 - Cardinality of the supplier by right-clicking near the supplier
- 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 on the background to open the right-click menu for the diagram.

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 open the right-click menu of some internal element instead. To avoid this situation, use **Shift+Right-click**. This opens the right-click menu of the desired element and preserves the current selection.

Drag-drop moving and copying elements

You can visually move elements into different packages, or members into different classes or interfaces, by dragging them on top of the destination element and dropping them. For example, you can drag classes into a package (on a class diagram), or components into a node (on a 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 elements or members around on a class diagram, the appearance of other elements changes when the dragged element crosses their boundaries. This indicates that the symbol being “crossed” has received focus and is a

potential drop target. For class, interface, and package elements, the drop focus is represented by a rectangle around the element border. Members are highlighted 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 a single diagram. This feature is set in the Options dialog - *General: Enable Drag-and-Drop*. When this option is checked, you can drag classes and members between the diagram and the Model tab of the Explorer, or between two diagram tabs. It is even possible to move elements within the Model tab from one node to another. So doing, the elements being moved are deleted from the source component and pasted to the destination component.

- Tip** This feature changes the behavior of automatic scrolling. The diagram does not scroll when an element is dragged outside the diagram workspace, because the element can be moved to a different component. If you want to scroll the diagram following the dragged element, you have to hold the element for a while near the internal border of the diagram. After a small delay, the diagram resumes scrolling. Note that the link dragging behavior is not affected.

Copying and cloning elements

You can copy and paste elements within the same diagram or between different diagrams. It is also possible to copy and paste members within the same class or between different classes, and to 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** The 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* using the Clone command on the right-click menu. Cloning is 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 manually 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 nodes 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 by using patterns.

To make a link bidirectional:

1. Select the link to be converted.
2. Choose the **Choose Pattern** command on the right-click menu for 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 that are associated 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 each link.
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 cannot be applied.

Labeling links

You can add textual labels to links to add information to the model. 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 | Client Role (Supplier Role)** on the right-click menu of the link. This opens the in-place editor for the role.
2. Enter the role description and press **Enter**.

The maximum length of the description is specified in the Options dialog (*Diagram: Maximum link label length without wrapping*). If the description is too long, the text is displayed in multiple lines, but only if this feature is enabled 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. However, oriented labels and multi-line labels are mutually exclusive. If oriented labels are selected, word wrapping is disabled.

Changing links

Once the links are created, you can 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. Move the mouse cursor to the source or destination point of the link. As the mouse cursor hovers over the target point, its shape changes to .

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 element.

Tip If the drop destination is out of range, drop the link end point on the diagram background and select the target in the dialog that opens.

To add bending points to a link:

1. Select a point on the link. As the mouse cursor hovers over the target point, its shape changes to cross-hairs.
2. Drag the selected point and drop in the required place on the diagram background.

To remove bending points:

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

To reverse a link:

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

Viewing diagrams and managing diagram layout

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

You can scroll the Designer pane horizontally and vertically using its scrollbars or the Overview button in the lower right corner of the Designer 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 Designer 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 | Diagram | 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 Designer pane.

- Use the Zoom lens button on the diagram toolbar:
 - Click the Zoom lens button and click on the Designer pane to zoom in.
 - Hold down the **Alt** key and click on the Designer 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.

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** reposition only selected elements.
- **Layout | Route Links** streamlines the links.
- **Layout | Actual Size** optimizes the element size.

Important The contents of the Layout submenu depend on the current selection.

Creating your own manual layout

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

- Select a single element and drag it to a new position.
- Select multiple elements and drag them as a group to a new position.

- Select multiple elements, cut them as a group, and paste them to a new position.
- Manually reroute links.

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, you can open one or more diagrams for editing. Each diagram opens in its own tabbed page in the Designer 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. In the Model tab of the Explorer, navigate to the diagram you want to open.
2. On the right-click menu of the diagram, choose **Open** to replace the contents of the current diagram, or **Open in New Tab** to open the diagram without closing the current one.

Tip Following are alternative ways to open diagrams:

- Double-click on the diagram node. This opens the diagram in the Designer pane, replacing the diagram currently in focus.
- 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 Following are alternative ways to close current diagrams:

- Use one of the keyboard shortcuts, 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 you select an element in a diagram or the diagram background, the properties of the element (or diagram) are available in the Inspector. The contents of Inspectors vary according to the selection. If nothing is selected, the diagram background is selected by default and diagram properties are displayed.

Inspectors have several tabbed pages representing different categories of properties. For example, the Inspector for a class 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 (for information on hyperlinking, see [“Hyperlinking diagrams” on page 153](#)).

View. Visual properties of the diagram symbol, 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.

Requirements. 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. You can modify an Inspector to add or modify stereotypes with some Java programming. For information on adding or modifying stereotypes, see [“Customizing Property Inspectors” on page 747](#).

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 (or Alt + double-click).

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.

Applying changes

Changes to property fields are applied when you:

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

Important If you click the Close button in the dialog while changes are pending to a field, *the changes are not saved*.

In-place editing

In addition to editing properties of elements in the Inspector, you can do in-place editing on the diagram itself. Every element on the diagram has an identifying string with a certain set of properties. 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, leaving the underlying engine to complete the changes.

There are several ways to enable in-place editing:

1. Double-click the selected element.

2. Choose **Rename** on the right-click menu for the element.

- or -

Press **F2** for the selected element.

This opens a highlighted text string with a cursor for modification, unless the diagram is read-only.

To apply changes and thus generate the relevant code:

1. Press **Enter** (except for Note elements)

2. Click on another element

Note When editing Note elements, keep in mind that Enter creates a new line and Ctrl+Enter applies editing and closes the in-place editor.

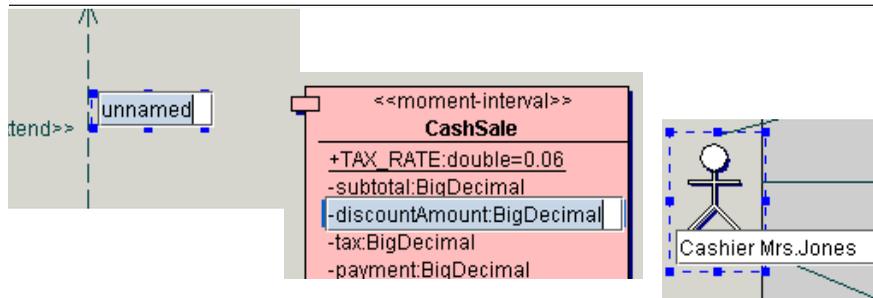
On code-generating diagrams, you must adhere to correct syntax when editing properties manually. Together responds to the wrong syntax of modifiers with error messages. But beware - a misspelled attribute type on a diagram is reproduced exactly in forward engineering.

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 remains 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 int` and if the return type of an operation is not specified, it defaults to `void`.

Most strings can be modified at any time.

Note Shapetypes and link types are not editable.

Figure 25 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.

The hyperlinking feature enables you to:

- 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 intranet or on the Internet.
- Create, view, remove, and browse hyperlinks using the right-click menu on the Hyperlink tab in the Inspector.
- Add or remove hyperlinks using the right-click menus of the link-type nodes of the Hyperlinks tab.

Note Hyperlinking is not available for some types of links between diagram elements. For example, generalization/implementation links do not allow hyperlinking. The Hyperlink to command is not provided on the right-click menu of such links. For association links, hyperlinking is allowed to files and URLs only.

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 various kinds of hyperlinks. Hyperlinks exist within the context of projects, so open a project to create or browse them. Hyperlinks are *properties*, which is why they appear in the Inspector. You can create hyperlinked diagrams using one of the following possible tools:

- Hyperlinks tab in the Inspector.
- Hyperlink To submenu on the Selection menu.
- Right-click menu of the selected element or current diagram.

Hyperlinking to a new diagram

You can 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 new diagram is hyperlinked to your originating element by default.

For instructions on hyperlinking to a new diagram, see [“Using the hyperlink feature to create diagrams” on page 135](#). The new diagram opens in the Designer pane and the Hyperlinks tab in the Inspector 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.
3. On the Selection menu, choose **Hyperlink To | Existing Element**. The Select Element dialog opens.
4. Choose the desired diagram or element in the dialog and click **Add**. To find an element, expand diagram nodes in the tree view.
5. Click **OK** to close the dialog and create the link.

Hyperlinking to a URL or file

You can create hyperlinks from your diagrams to any online resource. 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 online, you can link to it.

Note that file paths or URLs can be absolute or relative to the project.

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.
3. On the main menu, choose **Selection | Hyperlink To | File/URL**. The Choose URL dialog opens.
4. Choose the type of URL: **Project Relative** or **Absolute**.

5. Type in the complete path or browse for it. Note that if you change between relative and absolute path, then the specified path changes accordingly.
6. Click **Ok** to create the link.

Viewing hyperlinks

All the hyperlinks defined for any diagram or element are displayed on the Hyperlinks tab of the 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 are displayed 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 right-click menu on the Hyperlink tab to browse to the linked resources.

- Browsing to a linked **diagram** opens it in the Designer pane or makes it the current diagram, if already open.
- Browsing to a linked **element** opens its parent diagram, makes it current, and scrolls the diagram 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 default system Web browser and loads the URL.

To browse a hyperlink:

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

Removing hyperlinks

Hyperlinks can be removed 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 the diagram that has 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 for each diagram type has Note  and Note Link  buttons for placing notation on a 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 to the next line when you resize the note.
- Edit the properties of a note 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 on the Custom Properties tab.

Note Links

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.

To open the Inspector for a note link, select Properties from the note link right-click menu. The Inspector for a note link is similar to the note Inspector, but the Text tab is replaced by the Link tab. On the Link tab you can view both ends of the link - client and supplier.

Inspector documentation tabs

The Description tab in element Inspectors displays the description of the selected element. If the element is in a source generating diagram, the text in this tab becomes a Javadoc comment in the source code corresponding to the element. When you edit the text in this tab, the source code is updated when you press Alt+Enter or return to the diagram.

You can use the Description tab to create and edit comments for notes or elements on diagrams that do not generate code. These comments are stored with other diagram-specific information.

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

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

Standalone design elements

By default, design elements are stored inside the diagram packages. However, this simple organization makes it difficult to share elements among 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 (SDEs). 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 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 standalone design elements

To create SDEs:

1. On the main menu, choose **Tools | Options | <level>**.
2. In the *General* options group, check *Create design elements as standalone*.

When this option is checked, 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 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 Designer 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.

Tip If a diagram contains existing non-standalone design elements of the same shapetype, their names are not taken into account when new names for the new SDEs are assigned. However, all the standalone elements have different names assigned following the same rules as the names of the non-standalone design elements.

Using standalone design elements

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. You can 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**.

Important Using SDEs is restricted for the ER and XML structure diagrams. Design elements from these diagrams cannot be used in other types of diagrams. If you copy a standalone design element from an entity relationship or XML structure diagram and attempt to paste it to a diagram of any other type, the result is the error message "Cannot paste <SDE name> to <diagram name> (incorrect container)." However, you can share SDEs among diagrams of the same type.

Saving and copying diagram images

You can save or copy entire diagrams or their selected parts for further reuse. There are three possible behaviors: copying and pasting within Together, copying the image, and saving the image.

Copy - paste within Together

1. Select the required part of a diagram.
2. Choose **Edit | Copy** on the main menu or the 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 Designer pane to refresh the diagram.

Copy image

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

Choose the format for copying images: bitmap (BMP) or Windows Metafile (WMF) format. Bitmap format solves several problems with WMF format such as distorted fonts, incorrect conversion of localized fonts, and lack of Java 2D functions support. In addition, some rasterized graphics applications do not handle clipboard metafiles, but they do recognize bitmaps.

To set the format for copying diagrams, open the Options dialog and check or uncheck the option *General: Copy diagram to clipboard as bitmap*.

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

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 on the submenu. This opens the Save Diagram dialog for the selected format.

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

Printing diagrams and source code

You can print both diagrams and text. To print individual diagrams, open them in the Designer pane. To print source code, open it in the Editor pane.

Setting Print options

You can set printing options at different levels in the Options dialog.

To check Together Print options for the specified level:

1. Choose **Tools | Options | <level>** on the main menu to open the Options dialog for the default, project, or diagram level.
2. Select the *Print* node and check the settings.

For further information, see [Chapter 3, “Setting Your Personal Preferences” on page 65](#).

How to print diagrams

You can print diagrams separately or as a group, or print all diagrams in the project.

To print a single diagram:

1. Open the diagram in the Designer pane (the project must be open). If there are several open diagrams, click the tab of the one you want to print.
2. Choose **File | Print Diagram** on the main menu to open the Print Diagram dialog.
3. By default, the scope is set to *Current*. Click **Print**.

To print several diagrams at once:

1. Open the diagrams in the Designer 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. Choose *All opened* and click **Print**.

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 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 **Print**.

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 manual diagram layout, invoke Undo after running auto-layout and printing 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.

Under 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. Assuming Together is running, choose **File | Synchronize with External Changes**.
3. Print the diagram (**File | Print Diagram**).

Tips and tricks

- If you are running Together under Windows, set the True Type Fonts property for your printer 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.

UML Modeling

Together provides support for the most frequently used diagrams and notations defined by the Unified Modeling Language (UML). For thorough discussions of the UML, refer to the Object Management Group website at <http://www.omg.org>.

This chapter includes the following topics:

- “Notation and stereotypes of UML diagrams” on page 165
- “Use case diagrams” on page 166
- “Class diagrams” on page 168
- “Sequence and collaboration diagrams” on page 175
- “Statechart diagrams” on page 184
- “Activity diagrams” on page 188
- “Component diagrams” on page 190
- “Deployment diagrams” on page 192

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. You can also add color to stereotypes, although this practice is not currently specified in the UML.

Use case diagrams

Use case diagrams describe the external view of a system and its interactions with the outside world. Actors are entities outside the system that interact with it. A use case is an action or sequence of actions required by the system. A use case diagram depicts the relationship between actors and use cases within a system.

Figure 26 is an example of a use case diagram.

Figure 26 Use case diagram

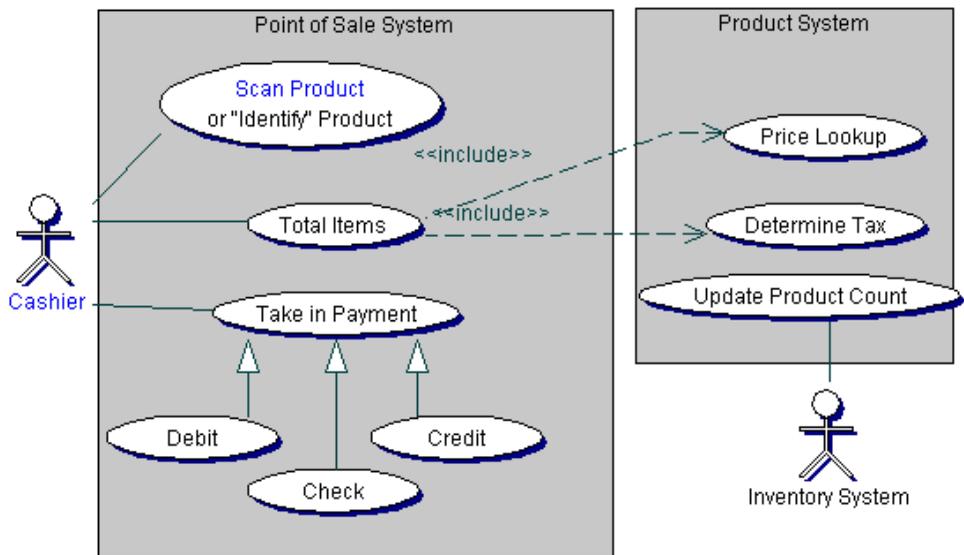


Diagram elements

Table 15 lists the elements of use case diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 15 Elements of use case diagrams

Icon	Element name
	Actor
	Use Case

Table 15 Elements of use case diagrams (continued)

Icon	Element name
	<i>Communicates</i>
	<i>Extends</i>
	<i>Includes</i>
	<i>Generalization</i>
	System boundary
	Note
	<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 [“Working with Diagrams” on page 133](#).

Creating browse-through sequences of use case diagrams

When you develop use cases, you typically begin at a high level and specify the main use cases of the system. As you proceed, you break these into finer detail. For example, a “Conduct Business” use case may have another level of detail that includes use cases such as “Enter Customers” and “Enter Sales.” It is useful to have a method of expanding or contracting the use cases to view the system at different levels of detail.

Together’s hyperlinking feature enables you to create browse-through sequences comprised of several use case (or other type) diagrams. You can link 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.

You can use hyperlinking to link diagrams and elements based on your requirements. For example, you can develop 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 153](#).

Note You can also show that two use case diagrams are related by placing a shortcut of one on the other. See [“Placing shortcuts on diagrams” on page 143](#).

Adding extension points

An extension point is a location within a use case where you can insert action sequences from other use cases. Each extension point in a use case has a unique name. The list of extension points is ordered according to 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 the diagram).

To add an extension point to a use case:

1. Right-click the use case and choose **New Extension Point**.
2. Edit the extension point in place.

The new extension point is added at the end of the list of existing extension points. If necessary, drag the extension point to change its position in the list.

Class diagrams

Class diagrams show the static structures of systems. This includes classes and interfaces and their internal structure as well as relationships between classes and inheritance structures, as shown in [Figure 27](#). A class diagram can also show objects and links between objects.

Figure 27 Class diagram

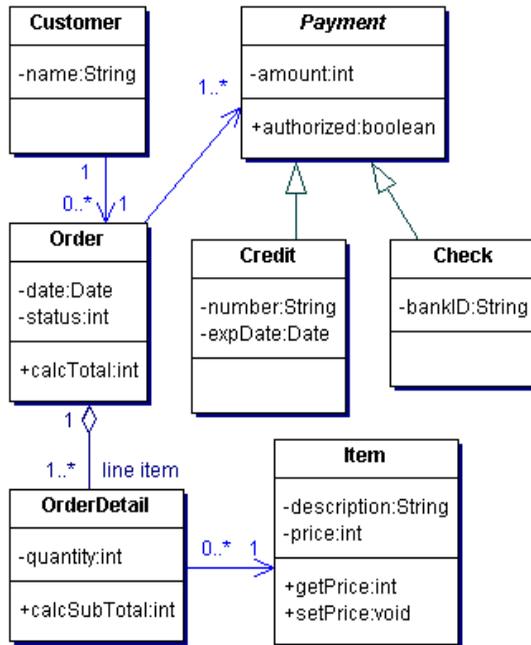


Diagram elements

Table 16 lists the elements of class diagram that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 16 Elements 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

Table 16 Elements of class diagrams (continued)

Icon	Element
	Note
	Note link
	Entity EJB
	Session EJB
	Message-driven EJB

In addition to these elements, class diagrams can contain shortcuts to the elements from other packages and diagrams.

Physical (package) and logical class diagrams

The Model tab of the Explorer uses two class diagram icons:

-  Physical class diagrams, which are referred to in this chapter as *package diagrams*. Package diagrams are stored as a text files with the extension `dfPackage`.
-  Logical class diagrams. These diagrams are stored as text files with the extension `.dfClass`.

Together creates a package diagram for the project (`<default>` diagram) and for each subdirectory under the project directory. The subdirectory paths are defined in the project properties (Project | Project Properties). You must create any logical class diagram separately.

Working with logical class diagrams

You can create a new logical class diagram in the usual manner, as described in [“Creating diagrams in projects” on page 133](#). When you place a class, interface, or package on a class diagram, Together generates the corresponding source code or subpackage in the package that you specified when you created the diagram.

The impact of changing a class, interface, or package on a logical class diagram varies according to the kind of change:

- Changing the name, adding a member, creating a new link, or applying a pattern makes the corresponding change in the actual source code.
- Dragging a class to a package on the logical class diagram either copies the class to the package or creates a shortcut to the class in the package, depending on your choice. It does not change the location of the actual corresponding class.
- Removing a class, interface, or package from a logical class diagram does not remove it from the project or from its corresponding package diagram.

Working with package diagrams

When you open a project subdirectory from the Model tab of the Explorer or from the diagram, Together reverse engineers the contents into a package diagram that shows the packages, classes, and interfaces and their interrelationships.

Opening packages

To open a package, choose one of the Open commands on the right-click menu of the package in the Model tab of the Explorer.

Deleting packages

To delete a package from a package diagram or from the Model tab of the Explorer, choose Delete from its right-click menu. Deleting a package deletes all of its contents.

Renaming packages

To rename a package, including changing the package name in all of its source files, choose Rename from its right-click menu. You can also rename by using the package's Inspector (Right-click menu | Properties).

Viewing packages and their content

By default, a package element on a diagram displays the package contents. You can use the right-click menu of a class or interface in a package to add attributes and operations directly.

To show or hide the contents of package elements:

1. Right click the diagram background and choose **Diagram Options** to open the Diagram Options dialog.
2. Select **View Management | Show subpackage contents**.
3. Check or clear *Show subpackage contents* as desired.

Moving and copying elements from packages

To copy packages, classes, or interfaces from another project package to a class diagram:

1. In the Model tab of the Explorer, navigate to the package containing the element(s) you want to show.
2. Locate the desired node in the Explorer and copy to the clipboard. Then paste from the clipboard to the desired location.
3. After completing the copy operations, choose **Update Package Dependencies** on the diagram right-click menu to update any package dependency links as desired. (For an explanation of updating package dependencies, see [“Drawing dependency links” on page 142.](#))

To move a class, interface, or package into or out of a package on the diagram, use one of the following methods:

- By drag and drop using the left button of your mouse. Use the right button to display the **Move** command.
- By copy and paste. In addition to working in the Designer pane, this method also applies to the Explorer pane.

Showing classes on search paths

Projects can have search paths whose content you may want to display in diagrams. For example, you can show entities that reside on the standard libraries or the 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 from a search path in a class diagram:

1. Open or create a class diagram.
2. Right-click on the diagram background and choose **New | Shortcuts**. The resulting dialog displays the content available for the diagram and all content residing outside the current package.
3. Select 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. For instructions on adding resources to project search paths, see [“Adding Resources” on page 103](#).

Showing JavaBeans

Together can recognize classes that match the bean pattern as JavaBeans.

To show classes as JavaBeans on a class diagram:

1. From the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. From the tree, choose **View Management | JavaBeans / C++ Properties**
3. Check the *Recognize JavaBeans* option.

Any class that satisfies the bean pattern shows a small rectangle at the top of the left side. All properties and events display in separate compartments below the operations compartment.

Defining inner classes

To create an inner class from a class that already exists, drag the existing class to the outer class and drop it. If the inner class does not exist, you can create and define it within its outer class at the same time.

To create and define a new inner class:

1. Select the outer class.
2. Choose **New | Inner Class** from the right-click menu.

Tip You can also define new inner classes from the right-click menus of class nodes in the Explorer.

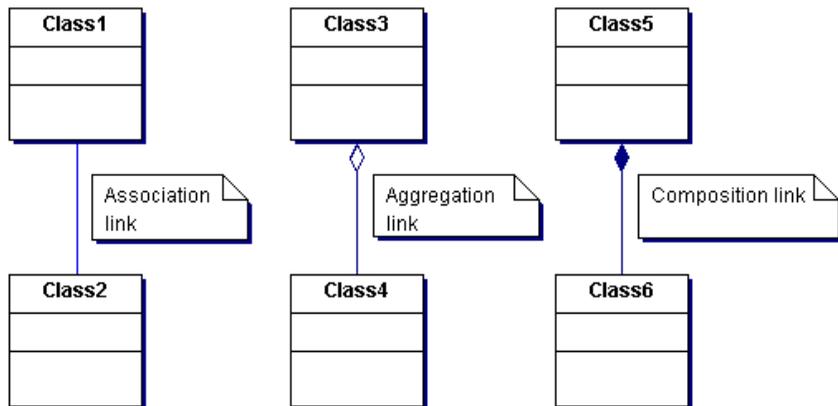
You can also use drag-drop to remove inner classes from classes in the diagram.

Creating associations, aggregations, and compositions

Use the association button  on the class diagram toolbar to draw association links between diagram elements. Right-click on an existing link to specify an aggregation or composition link. You can change the code corresponding to the link by applying the appropriate pattern to the link.

Figure 28 illustrates how Together depicts the three types of association links.

Figure 28 Types of association links



When you create an association link, Together defines an attribute in the client class (the start of the link) named `lnkClass`, where `Class` is the name of the target class. By default, such attributes are not displayed in the class elements.

The right-click menu of a link enables you to set the link type (association, aggregation, composition) and the cardinality of the client and supplier. If you right click the link near the supplier, you can change its cardinality. Right click the link near the client to change its cardinality.

Associations can be automatic, directed, or undirected. A directed link points to the supplier class (the target). An automatic link points only if the name of the corresponding attribute in the client class does not begin with `lnk`.

To set the directed property of an association link:

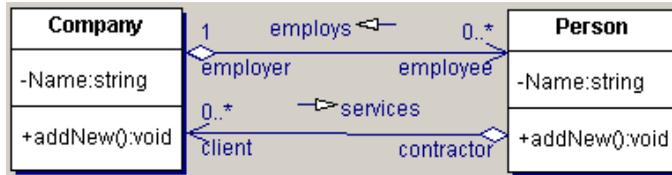
1. Right-click on the link and select **Properties** to open the Inspector.

- From the Link tab of the Inspector, set the *directed* property to **Automatic**, **Directed**, or **Undirected**.

You can show the semantic direction of a link by specifying a label and label direction in the link Inspector.

Semantic direction may be independent of the link navigation direction, as shown in [Figure 29](#). In this example, 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 29 Semantic direction of the link label vs. link direction



Editing members and properties

This section includes instructions for adding and editing members and properties of classes and interfaces.

Adding and editing members

To add an attribute or an operation to a class or interface:

- Right-click on the class or interface
- Choose **New | Attribute**, **New | Operation**, or **New | Constructor**.
- Edit the member with the in-place editor, or use its Inspector.

The new member is inserted at the end of its compartment in the class element on the diagram.

If a class already has attributes or operations, you can right-click on one of them to create an additional member using the right-click menu. The new member is inserted before the selected member.

Tip Class and interface nodes in the Model tab of the Explorer have right-click menus for adding new members.

Adding and editing properties

To add a property to a class, right-click on the class and choose **New | Property**.

The location of the newly created property depends on whether the Recognize JavaBeans option is on. If so, the new property is added to the properties compartment and to the list of properties in the Beans tab of the Inspector. The class displays as a bean symbol. If the option is off, the property is added to the attributes section, and its accessor methods are added to the operations section.

When the Recognize Java Beans option is off, you must take special care when editing or deleting properties. If you edit a new property with the in-place editor, the relevant types for the accessor methods will not be synchronized. When a property is deleted, the accessor methods remain and you must delete them individually. The safest method for editing properties is to use the Choose Pattern dialog.

To edit properties using the Choose Pattern dialog:

1. Right-click on the property to be edited and select **Choose Pattern**. This opens the Choose Pattern dialog.
2. Choose the **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 or operations

You can use drag and drop to reorder members within a class. Together simultaneously updates the source code with the new order.

Dropping one member on the name of another member 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 display an expansion/contraction control in the attributes and operations compartments of class and interface elements, and in the classes, interfaces and diagrams compartments of the package elements. Selecting the package, class or interface activates the control, which you can use to toggle the display of members in the compartment. This is particularly useful when you have large container elements with content that does not need to be visible at all times.

To set the compartment controls:

1. From the main menu, choose **Tools | Options | <level>**.
2. Select **Diagram** in the resulting Options dialog.
3. Check *Show controls for compartments*.

Sequence and collaboration diagrams

Sequence and collaboration diagrams model the dynamic aspects of a system. Both depict interactions described by a set of objects, their relationships, and the messages exchanged between them.

Collaboration diagrams emphasize the structural organization of objects, as illustrated by [Figure 30](#). In contrast, sequence diagrams emphasize the time ordering of messages, as illustrated in [Figure 31](#).

Collaboration diagrams are 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 with no loss of information.

Although they are semantically equivalent, sequence and collaboration diagrams do not necessarily reveal the same information. For example, collaboration diagrams explicitly show how objects are linked, while in sequence diagrams the links are implied. Message return values display in sequence diagrams but not in collaboration diagrams.

Figure 30 Collaboration diagram

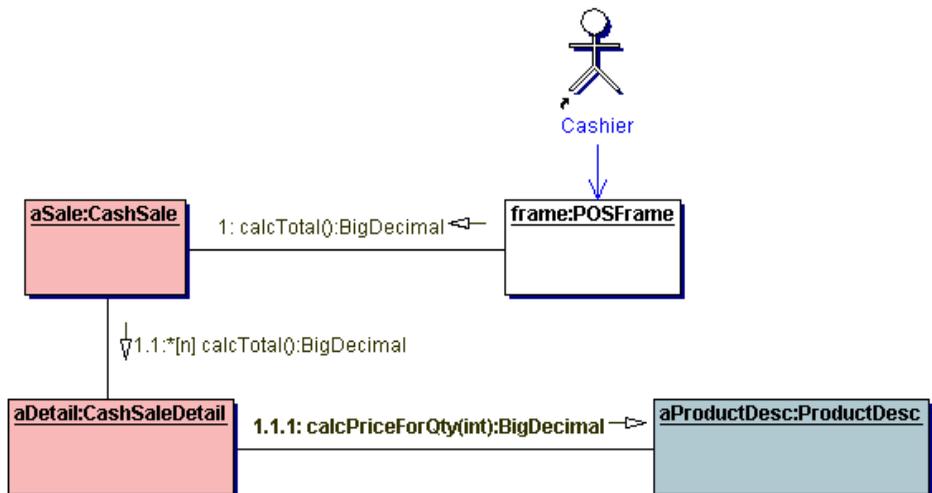


Figure 31 Sequence diagram

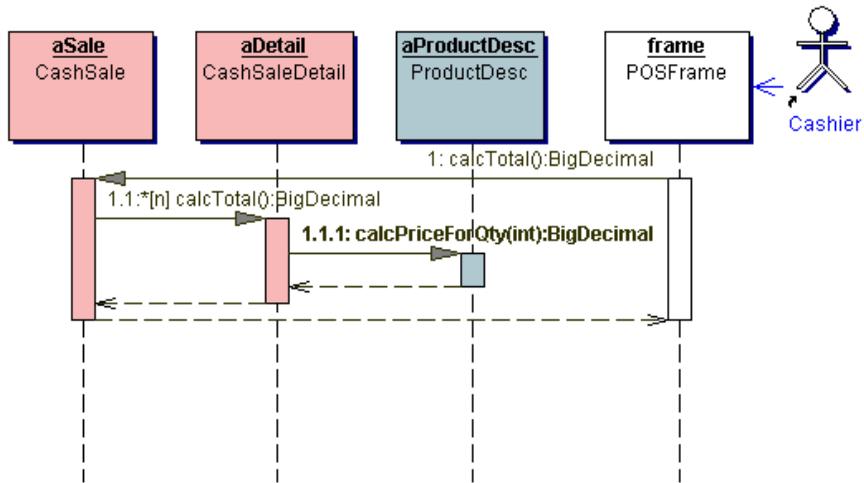


Diagram elements

Table 17 lists the elements of collaboration diagrams and Table 18 lists the elements of sequence diagrams. Relationship links are in *italicized text*.

Table 17 Elements of collaboration diagrams

Icon	Element
	Object
	Actor
	<i>Message</i>
	<i>Self Message</i>
	<i>Associates</i>
	<i>Aggregates</i>
	Note
	<i>Note link</i>

Table 18 Elements of sequence diagrams

Icon	Element
	Object
	Actor
	<i>Message</i>

Table 18 Elements of sequence diagrams (continued)

Icon	Element
	Message with delivery time
	Self Message
	Statement Block
	Note
	Note link

Converting between sequence and collaboration diagrams

You can 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.

To convert between sequence and collaboration diagrams:

1. Right click on the diagram background.
2. If the diagram is a sequence diagram, choose **Show as Collaboration**. If viewing a collaboration diagram, choose **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 sequence and collaboration diagram nodes in the Explorer.

Creating sequence and collaboration diagrams

You can create sequence and collaboration diagrams and populate them using buttons on the diagram toolbar. You can also generate sequence diagrams from operations on a class diagram.

Note To generate a collaboration diagram from an operation, first generate the sequence diagram and then convert the diagram into a collaboration diagram.

Generating sequence diagrams from operations

To generate a sequence diagram from an operation:

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. Right click the operation and choose **Generate Sequence Diagram**. This opens the Generate Sequence Diagram, which lists packages and classes involved in the operation.

4. In the Package/class list, check the packages and classes you want to display in the generated diagram. All packages and classes are selected by default. However, some Java classes might not be relevant, such as `java.lang.Integer`. To increase the meaningfulness of the generated diagram, remove anything that is not helpful in explaining the sequence of operations.
5. For those elements you decide to show in the diagram, check whether or not to show implementation details in the generated diagram.
6. Click **Ok** to generate the diagram.

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 Designer pane.

Setting options to control generation of sequence diagrams

The Options dialog enables you to set options for the generated diagrams.

To access sequence diagram options:

1. From the main menu, choose **Tools | Options | <level>**. This opens the Options dialog.
2. Expand **View Management | Sequence diagram**.

It is possible to generate sequence diagrams for several methods. You can display a sequence diagram for each method in a separate tab or show the 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*.

If *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.

If *Show multiple diagrams* is checked, all diagrams open automatically after they are generated.

It is also possible to control nesting level. *Depth of call nesting* enables you to change nesting values according to the desired degrees of complexity.

Specifying classes for objects

You can specify a class for an object on a sequence or collaboration diagram.

To specify the class for an object:

1. Open the object's Inspector (Right click | Properties) and click on the **Properties** tab.
2. In the *instantiates* field, enter a value or click the browse button to pick the class from a dialog that lists all the resources available to the project.

The object displays its fully qualified name. The Inspector displays a Class tab in which you can access the properties of the object's class.

Tip You can also specify the classifier by right clicking the object and choosing Choose Class.

To create a shortcut to the object's class on the diagram, right click the object and choose Import Class. To remove its association with a class, right click the object and choose Unlink Class.

Working with messages

This section describes techniques for working with messages in sequence and collaboration diagrams. Although the two diagram types are equivalent, the techniques for dealing with messages differ.

Messages in collaboration diagrams

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. Messages display in time-ordered sequence from top to bottom of the messages list. You can edit message properties in their Inspectors.

In addition to message links, you can add links to collaboration diagrams that show association and aggregation relationships. These links do not display if you view the diagram as a sequence diagram.

Messages in sequence diagrams

Messages in sequence diagrams have many more options than messages in collaboration diagrams. The options are on the Link tab of the message Inspector.

Creating a message-to-self

To create a message from an object back to itself:

1. Click the **Self Message** button on the diagram toolbar
2. For a sequence diagram, click the object's lifeline at the point where you want the message to appear.

Note To create a message to self in a collaboration diagram, click the Self Message button then click the object twice (for the start and the end of the message).

Creating a message link that corresponds to an operation call

If the Editor pane is visible, it displays the source code of this operation.

To create a message link that corresponds to an operation call:

1. Create a message link between two objects. The recipient object must instantiate a class.
2. Open the Link tab of the message link Inspector.

3. Enter an operation in the *operation* field, or click the browse button to choose the operation.
4. Select the operation and click **Ok**. If the operation is not a member of the recipient's class, you must confirm whether to create a new operation or rename an existing one.

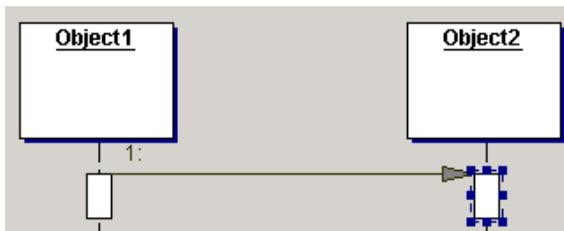
The message link is named according to the name of the operation. The Inspector for the link displays two tabs. The MessageLink tab displays message link properties and the Operation tab displays the operation's properties.

Tip You can create a new operation from a message by right clicking the message and choosing **New | Operation** or **New | Constructor**.

Changing activation times

Together automatically renders the activation of messages that show the period of time that the message is active. When you draw a message link to the destination object, the activation bar is created automatically, as shown on [Figure 32](#).

Figure 32 Activation of an object on a sequence diagram.



You can extend or reduce the period of time of a message by vertically dragging the top or bottom line of the activation bar as required. A longer activation bar means a longer time period when the message is active.

Reordering message links

To reorder messages, drag message links up and down the object lifeline. Reordering automatically updates the message link numbers.

You can 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 several message links by pressing **CTRL** key, then those selected are moved keeping their increments.

Branching messages

You can branch together messages from the same activation bar using the Branching command of the message right-click menu. Right click the message and choose **Branching | With previous**.

To remove branching, right click the branched message and choose **Branching | None**.

Annotating message links

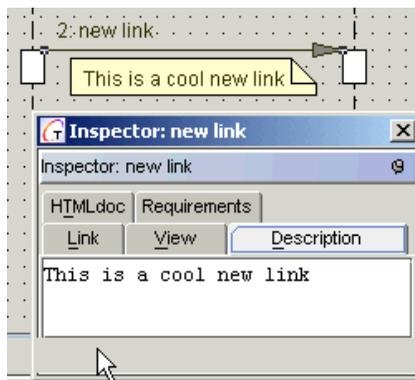
You can comment links using the Note button on the diagram toolbar. As the diagram changes, however, such notes do not move according to the messages they are attached to. To avoid this problem, annotate the links.

To enable link annotation:

1. On the main menu, choose **Tools | Options | <level>**.
2. Expand *View Management - Sequence diagram*.
3. Check *Show message description*.

To annotate a message link, use the Description tab of the message Inspector, as shown in [Figure 33](#).

Figure 33 Message description

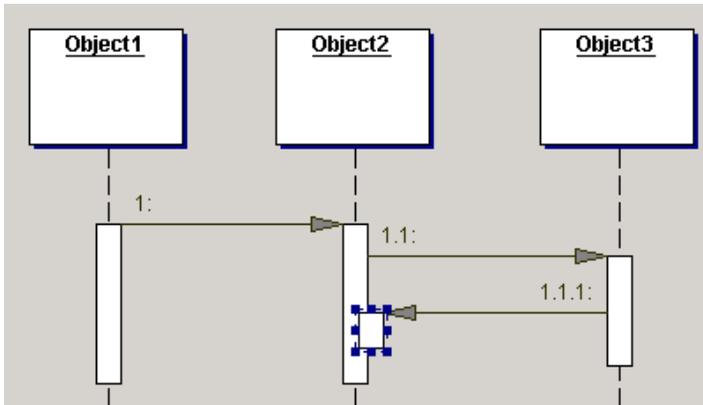


Nesting messages

You can nest messages by originating message links from an activation bar. The nested message inherits the numbering of the parent message. For example, if the parent message has the number 1, its first nested message is 1.1.

It is also possible to create message links back to the parent activation bars, as shown on [Figure 34](#).

Figure 34 Nested messages



Creating statement blocks

When you place a statement block on an activation bar, a dialog opens for specifying which control structure the block represents. If the control structure requires a condition, you can enter the condition with the in-place editor.

Working with lifelines

Adjusting the default lifeline of objects

Objects display with a default lifeline when you place them on a sequence 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 moves downward to show that it exists at a point later 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 up or down. You can arrange the position of other intervening messages this way also.

Changing the order of objects in a sequence diagram

You can reorder objects in a sequence diagram and maintain any message links already created between them. Select any object and drag it across the *x*-axis of objects to the desired position. You cannot move objects vertically along the *y*-axis of time 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 in the Link tab of the Inspector.

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 Designer 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.

Generating implementation source code

You can generate a sequence diagram from source code. You can also generate source code from messages or activation bars of sequence diagrams. Each message in the call sequence must correspond to an actual operation.

To generate source code for a message or activation bar:

1. Right click the message or activation bar.
2. Choose **Generate Implementation**.

When the process is completed, labels of messages for which implementation code has been generated display in **bold** style on the diagram.

By default, Together does not generate code in any operations that do not have empty bodies. You can change this default behavior through the sequence diagram options.

Statechart diagrams

Statechart diagrams enable you to model the behaviors of objects as they goes through different states in response to events. Statechart diagrams show the possible states of an object and the transitions between states.

[Figure 35](#) is an example of a statechart diagram.

Figure 35 Statechart diagram

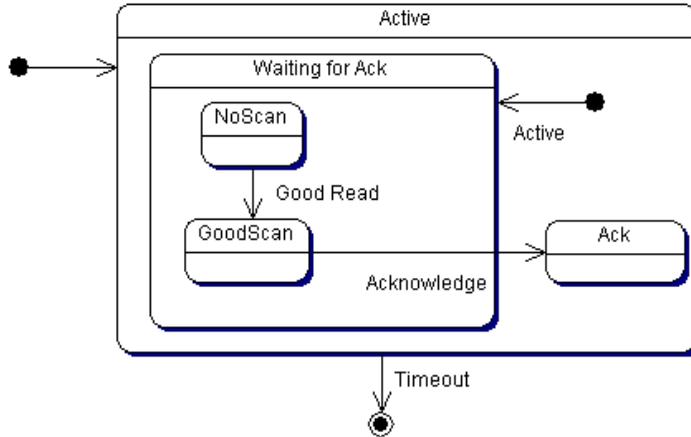


Diagram elements

Table 19 lists the elements of statechart diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 19 Elements of statechart diagrams

Icon	Element
	State
	Start
	End
	History
	<i>Horizontal Fork/Join</i>
	<i>Vertical Fork/Join</i>
	<i>Transition</i>
	Object
	Note
	<i>Note link</i>

Drawing a self-transition

A self-transition flow leaves the state, dispatching any exit action(s), then re-enters the state, dispatching any entry action(s).

To create a self-transition:

1. Drag a transition link from the desired state and drop it on the diagram background.
2. In the resulting dialog, select the state where you began the link and click **Ok**.

Tip An alternative way to draw a self-transition is to draw a transition between two states, and then drag the opposite end of the link back to the desired state.

Creating internal transitions

An internal transition is a shorthand for handling events without leaving a state and dispatching its exit/entry actions.

To create an internal transition:

1. Select the desired state on the diagram
2. From the state's right-click menu choose **New Internal Transition**.

You can 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. You create entry and exit actions in Together statechart diagrams as stereotyped internal transitions.

To create an entry or exit action:

1. Create an internal transition in the desired state.
2. Edit the name of the transition in place, using the following syntax:
`stereotype/actionName[argument]`

For example: `exit/setState(idle)`

Optionally, you can create the internal transition and set the event name, event arguments, and action expression properties in the Inspector for the transition.

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 between the contained substates. The easiest way to create a nested substate is to place a state element on the diagram background, then drag and drop it on top of another state.

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 in statechart diagrams with either a vertical or horizontal orientation. The statechart diagram toolbar provides separate fork/join buttons each one. 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 a horizontal or vertical fork/join on the diagram and resize it as needed.
3. If depicting multiple sources, draw transitions from each of the source states to the fork/join, then draw a transition to the target state.
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.

Modeling complex states

The techniques listed here pertain to models of complex composite states and substates.

- **Resize 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 a substate diagram.
- **Use hyperlinks to indicate nested substates.** You can nest multiple levels of substates inside one state. For especially complex substate modeling, however, it may be more convenient to create different diagrams, model each of the substate levels individually, and hyperlink the diagrams sequentially.
- **Apply existing content to a new diagram.** You can create a new diagram with the same content as the existing one by using the Clone command on the right-click menu of a diagram node in the Explorer. The new diagram has a unique name and is created in the same package as the original.
- **Reuse elements of existing diagrams.** You can reuse existing elements in other statechart diagrams by using the Add Shortcut command on the diagram's right-click menu. This opens the Add Shortcut dialog, in which you navigate to the existing statechart diagram and select elements, states, histories, and/or fork/joins.

Activity diagrams

An activity diagram is 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, and 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.

Figure 36 is an example of an activity diagram.

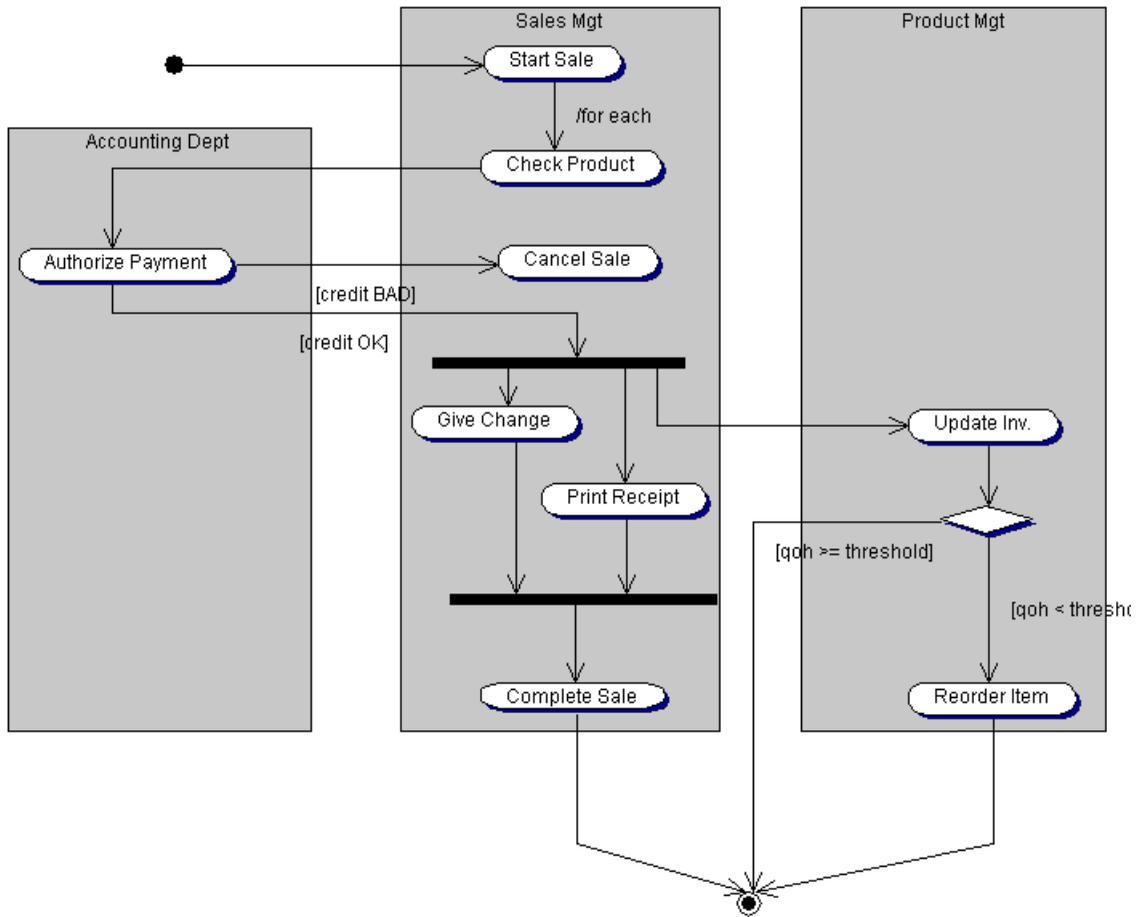
Diagram elements

Table 20 lists the elements of activity diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 20 Elements of activity diagrams

Icon	Element
	Activity
	Decision
	Signal Receipt
	Signal Sending
	State
	History
	Object
	Start
	End
	Horizontal Fork/Join
	Vertical Fork/Join
	Swimlane
	<i>Transition</i>
	<i>Object Flow</i>
	Note
	<i>Note link</i>

Figure 36 Activity diagram



Working with activity diagrams

Following are tips and techniques that you can use when creating activity diagrams:

- **Create diagrams with identical content.** Non-trivial systems can 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.
- **Reuse elements of existing diagrams.** Invoke the Add Shortcut command on the diagram's right-click menu to 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 long labels may affect the diagram layout.

Component diagrams

Component diagrams show the components and interfaces of systems and the relationships between them. [Figure 37](#) is an example of a component diagram.

A component is a container of other logical elements, and represents things that participate in the execution of a system. Components also use the services of other components via one of its interfaces. Components are typically used to visualize logical packages of source code (work product components), binary code (deployment components) or executable files (executions components).

Diagram elements

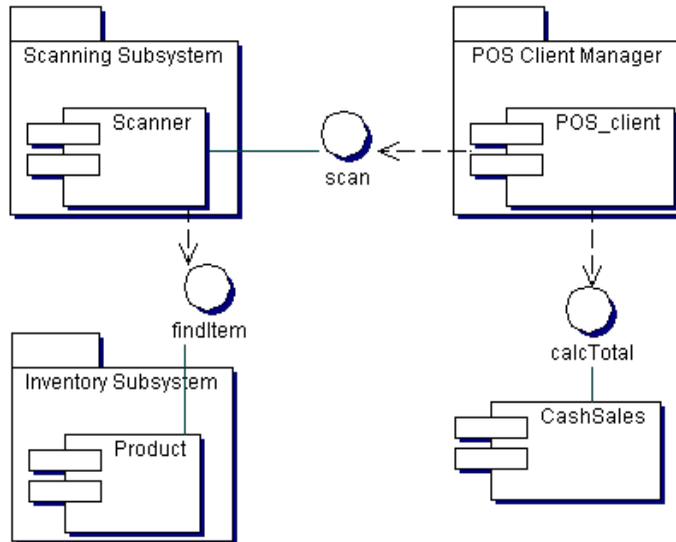
Component diagrams depict two kinds of relationships: dependency and realization. Component diagrams also can contain packages or subsystems that group logical elements of the model.

[Table 21](#) lists the elements of component diagrams that are on the diagram toolbars. Relationship links are in *italicized text*.

Table 21 Elements of component diagrams

Icon	Element
	Subsystem
	Component
	Interface
	<i>Supports</i>
	<i>Dependency</i>
	Note
	<i>Note link</i>

Figure 37 Component diagram



Working with component diagrams

Following are tips and techniques that you can use when working with component diagrams:

- **Apply 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.
- **Reuse elements of existing diagrams.** Invoke the Add Shortcut command on the diagram's right-click menu to reuse existing elements in other component diagrams.

Note Elements imported this way are independent copies of the existing ones.

- **Represent realization.** You can represent realization in two ways: use the support stereotype and the dashed line (canonical form), and use the solid line (lollipop notation).
- **Hide 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 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 38 is an example of a deployment diagram.

Figure 38 Deployment diagram

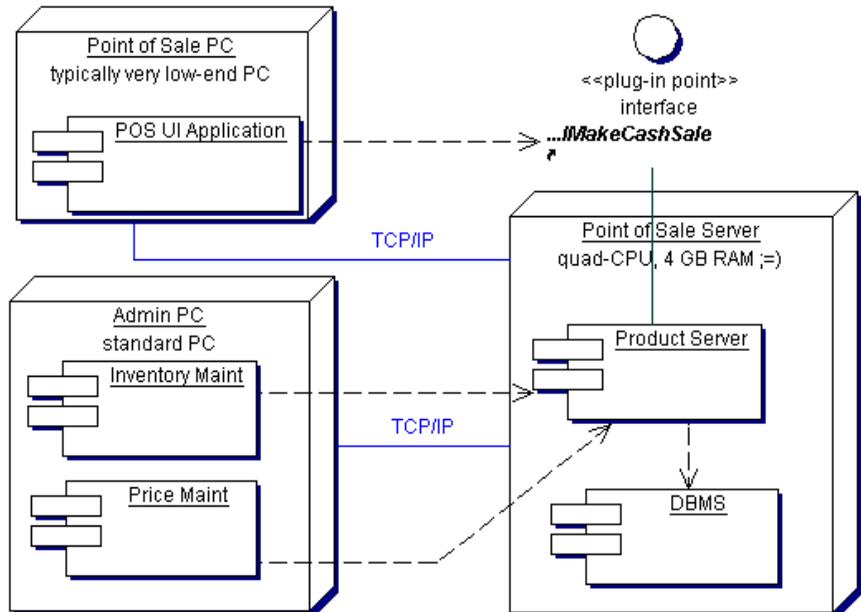


Diagram elements

Table 22 lists the elements of deployment diagram that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 22 Elements of deployment diagrams

Icon	Element name
	Node
	Component
	Interface
	<i>Supports</i>

Table 22 Elements of deployment diagrams (continued) (continued)

Icon	Element name
	<i>Associates</i>
	<i>Aggregates</i>
	Object
	Dependency
	Note
	<i>Note link</i>

Working with deployment diagrams

Following are tips and techniques that you can use when working with deployment diagrams:

- **Apply existing content to a new diagram.** Using the Clone command on the right-click menu of the diagram node in the Explorer pane, you can 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.** Invoke the Add Shortcut command on the diagram's right-click menu to reuse existing elements in other deployment diagrams.

Note Elements imported this way are independent copies of the existing ones. Organize components by specifying the relationships between them.

- **Indicate 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 only temporarily. To indicate this, use the dependency relationship with a *becomes* stereotype.
- **Represent how a component resides on a node.** A component may reside on nodes. You can represent this in two ways: use the *support* stereotype and the dashed arrows, or graphically nest the component element within the node element.

UML Extension Diagrams

In addition to the UML diagrams discussed in [Chapter 11](#), Together provides extension diagrams that are not currently supported by the UML standard. You can create these diagrams by choosing types from the Together tab of the New Diagram dialog. For more information, see [“Creating diagrams in projects” on page 133](#).

This chapter includes the following topics:

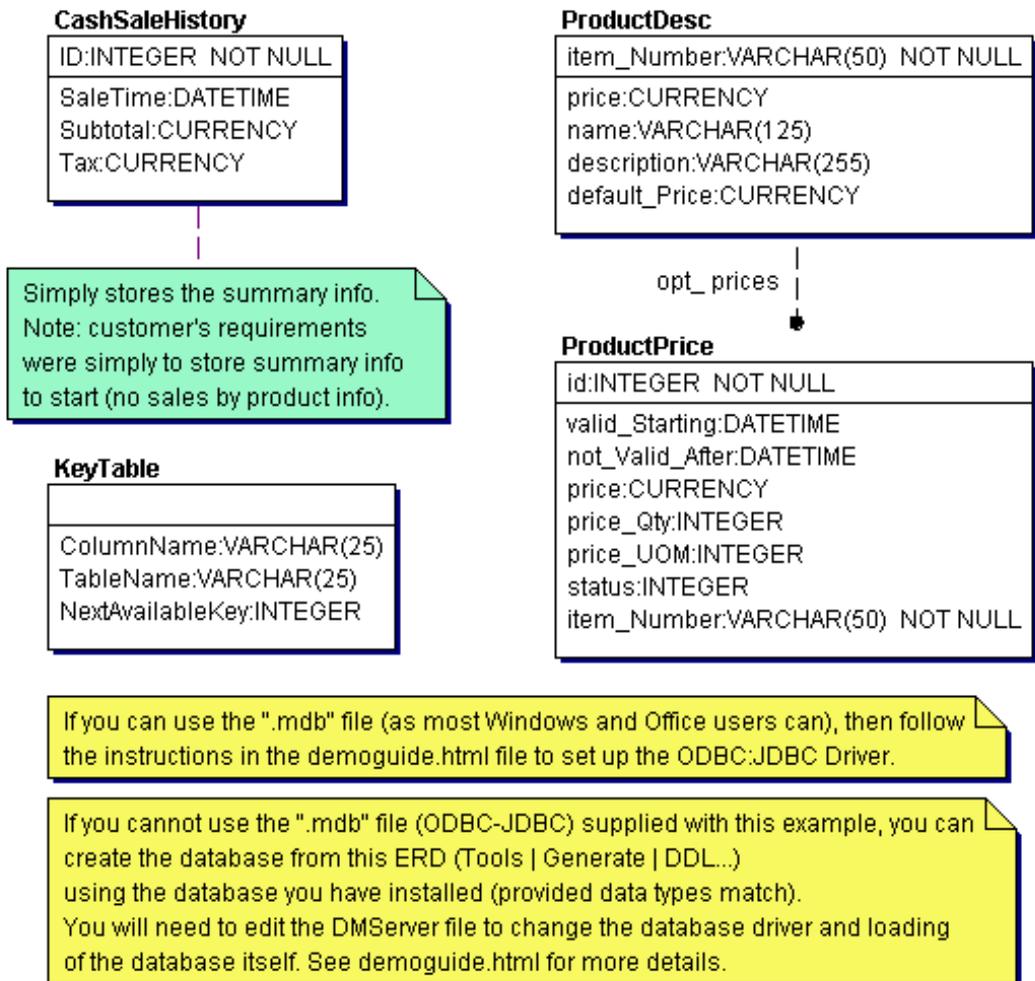
- [“Entity relationship diagrams” on page 195](#)
- [“Business process diagrams” on page 200](#)
- [“Robustness analysis diagrams” on page 201](#)

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 39](#), 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 39 Entity relationship diagram

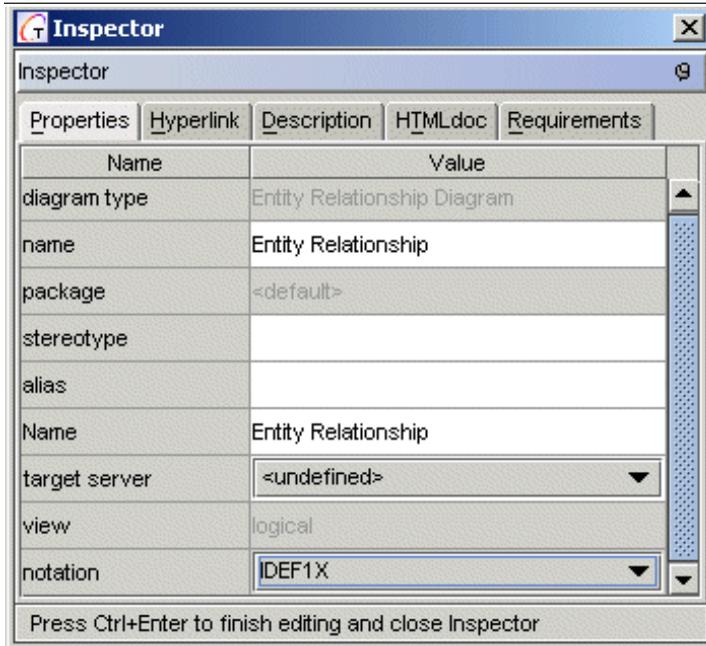


Notation

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 40 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.

Tip You can also toggle between views using the diagram right-click menu. Choose View node on the right-click menu, and check the required checkbox.

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.

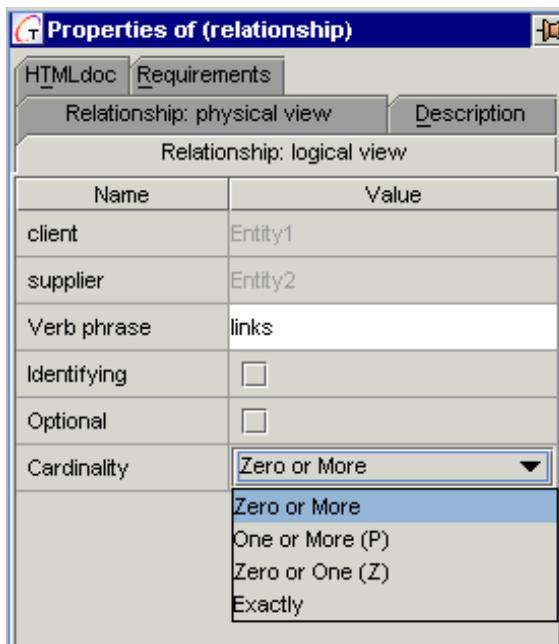
The *Size* parameter used for the string variables specifies the maximum number of characters (Digits field is disabled). If attribute type is numeric, *Size* specifies the total length of the number. The sense of *Digits* field depends on the selected database server. In general it defines the number 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 41](#).

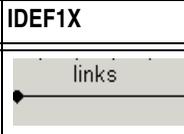
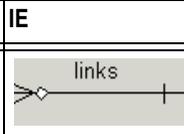
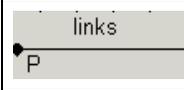
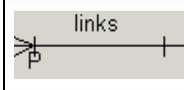
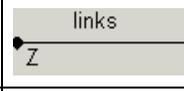
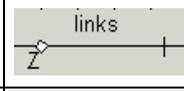
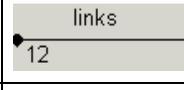
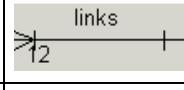
Figure 41 Properties inspector for an ERD



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

Business process diagrams enable you to apply 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 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.

Business process diagrams present the static aspects of a business object model. Sequence and activity diagrams can present the corresponding dynamic aspects.

Diagram elements

Table 25 lists the elements on the business process diagram toolbar. Relationship links are in *italicized* text.

Table 25 Elements of business process diagrams

Icon	Element
	Actor

Table 25 Elements of business process diagrams (continued)

	Use Case
	Business Worker
	Business Entity
	<i>Communicates</i>
	<i>Extends</i>
	<i>Includes</i>
	<i>Generalization</i>
	<i>Aggregates</i>
	<i>Subscribes</i>
	<i>Associates</i>
	System boundary
	Note
	<i>Note link</i>

Notation

The current UML specification does not determine graphical variations 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.

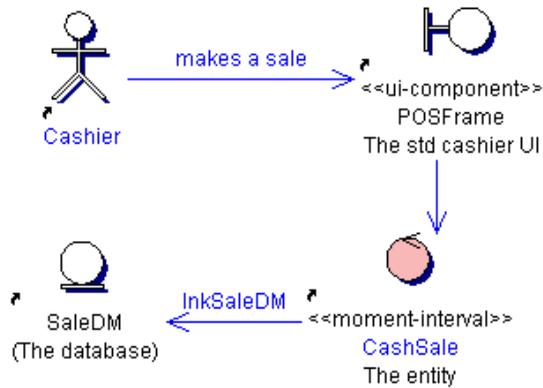
Robustness analysis diagrams

Robustness analysis involves evaluating the text of a use case and planning how to implement it using the objects you have discovered up to this point. A robustness model provides a bridge between the “analysis level” view described in the text of the use case and the “detailed design” view shown on a sequence diagram.

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. Robustness analysis suggests classifying objects into one of three stereotypes: boundary, control, and entity.

[Figure 42](#) is an example of a robustness analysis diagram.

Figure 42 Robustness analysis diagram



It is beyond the scope of this document 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 elements on the robustness analysis diagram toolbar. Relationship links are in *italicized text*.

Table 26 Elements of robustness analysis diagrams

Icon	Element
	Boundary
	Entity
	Controller
	Worker boundary
	Worker controller
	Actor
	<i>Association</i>
	<i>Robustness Association</i>
	Note
	<i>Note link</i>

Key elements and properties

Robustness diagram elements such as entity, boundary, and worker boundary represent stereotyped views of classes. Each such element on the robustness diagram should correspond to a class. Controller and worker controller correspond to methods. Rosenberg's suggestion of using "invokes" in place of "includes" and "precedes" in place of "extends" is realized via robustness associations.

Key elements and properties of robustness diagrams include:

- **Boundary.** Objects in the new system with which the actors will be interacting. These may be windows, screens, dialogs, and menus.
- **Entity.** Database tables and files that store data, fetch data and perform computations that do not change frequently. These objects should be simple and generic enough to provide the possibility of future reuse.
- **Controller.** Objects that 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.** Logical, directed associations.

Associations follow these rules:

- Actors can talk only to boundary objects.
- Boundary objects can talk only to controllers and actors.
- Entity objects can talk only to controllers.
- Controllers can talk to both boundary objects and controllers but not the actors.

Real-Time Modeling

The Together Real-Time feature provides support for defining, modeling, and simulating object-oriented real-time systems and applications. The feature 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 Kuusela, and Jurgen Ziegler.

The topics discussed in this chapter include:

- [“Introduction to Together Real-Time” on page 205.](#)
- [“Special real-time diagrams” on page 207.](#)
- [“Real-time audits” on page 222.](#)
- [“Simulation” on page 223.](#)

Introduction to Together Real-Time

The Together Real-Time feature is available in Together. The feature 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 feature

Real-time modeling in Together provides the following features:

- **System context diagram:** Captures real-time requirements
- **System architecture diagram:** Models subsystems
- **Event sheet diagram:** Defines real-time system events
- **Interaction diagram:** Shows object groups, threads, and messages

- **Concurrent state element:** Statechart diagrams
- **Expanded set of real-time asynchronous messages:** 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 Together Real-Time is a 3-step process:

1. Activate the feature:
 - 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 feature 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 feature. 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.

To create a real-time diagram:

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**.

To filter the views of some of the elements of real-time diagrams:

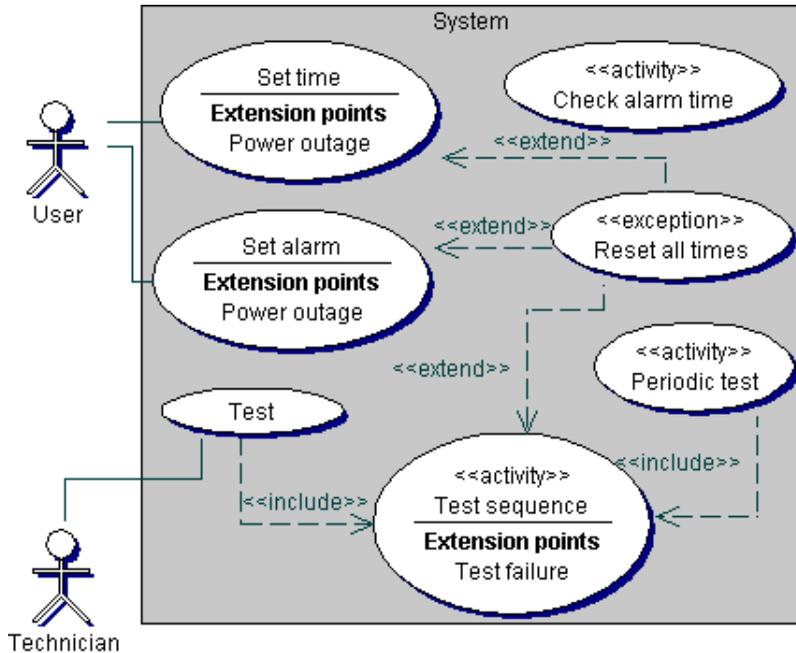
1. From the main menu, choose **Tools | Options | <level>**.
2. In the resulting dialog, navigate to *View Management - Show - Real-time*.
3. Check or clear the views for each of the listed element types as desired.
4. Click **Ok** to close the dialog and save the diagram view options.

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 43](#) is an example of a real-time use case diagram. We will use it to discuss the different stereotypes and links.

Figure 43 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 43](#), 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 43](#), 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 43](#) 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 43](#), 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.

Table 27 Elements of a system context diagram (continued)

Icon	Element Name	Description
	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 44](#) 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.

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 feature.
2. **use stereotype icon:** Displays the element using its icon instead of the standard class rectangle.
3. **system requirements:** Described below.

System requirements

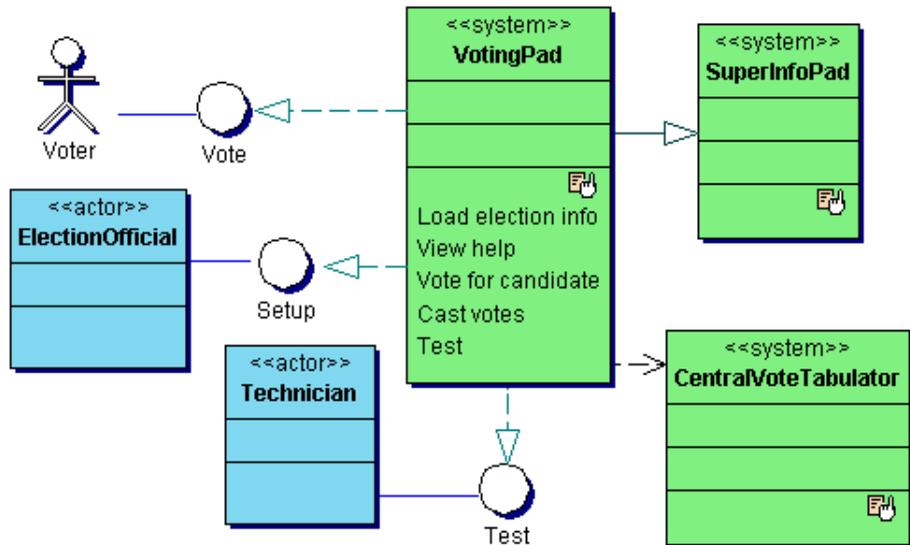
Together Real-Time introduces the idea of specific requirements elements attached to a system. System requirements appear in the bottom compartment of a system.

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 44](#) 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.

Figure 44 System context diagram



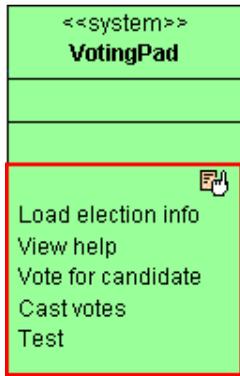
Creating a new system requirement

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 45 shows the requirements compartment, which is outlined in red.

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.

Figure 45 Requirements compartment of a system



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 46](#) shows an example of a system architecture diagram for a retina scanner, which is a networked biometric access system.

[Figure 46](#) shows the retina scanner divided into three main subsystems:

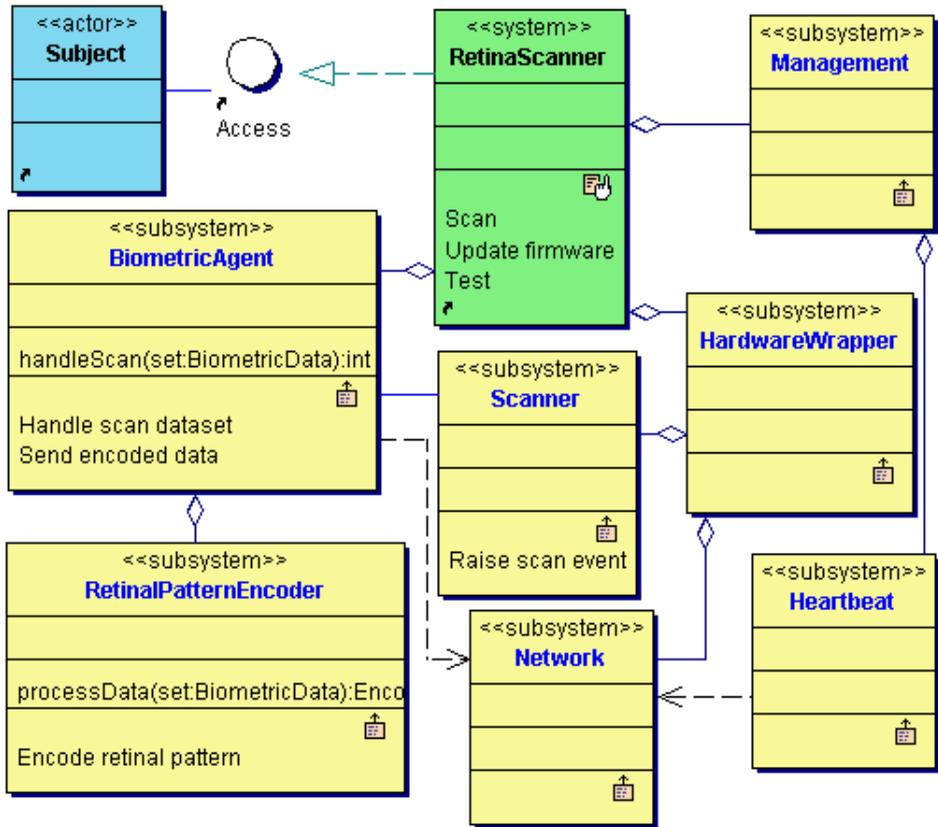
- BiometricAgent
- Management
- Hardware Wrapper

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.

Figure 46 System architecture diagram



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 46](#). 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 46](#) for an example.

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.

Note Subsystem names must be unique in the context of the entire project.

Subsystem Properties

In addition to the standard properties and those common to real-time elements, discussed in [“System context diagrams” on page 210](#), 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 the real-time feature. 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 the subsystem 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

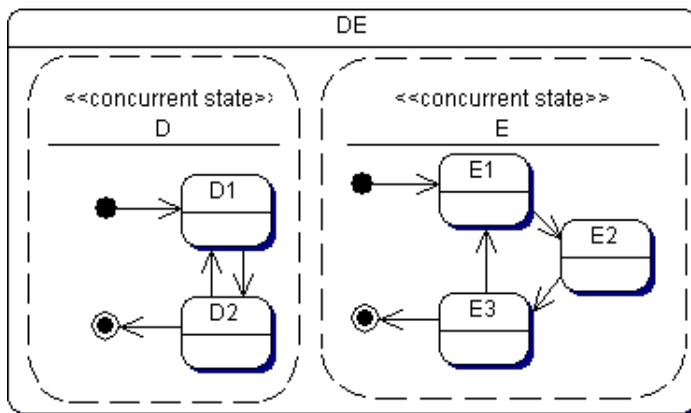
Together Real-Time adds one element to the standard statechart diagram, *concurrent state*.

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 47 shows a statechart diagram with concurrent states.

Figure 47 Statechart diagram



If a system such as DE in Figure 47 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.

Event Sheet diagrams have two event related elements: *Events* and *Generalizations/Implementations*, shown in Table 30.

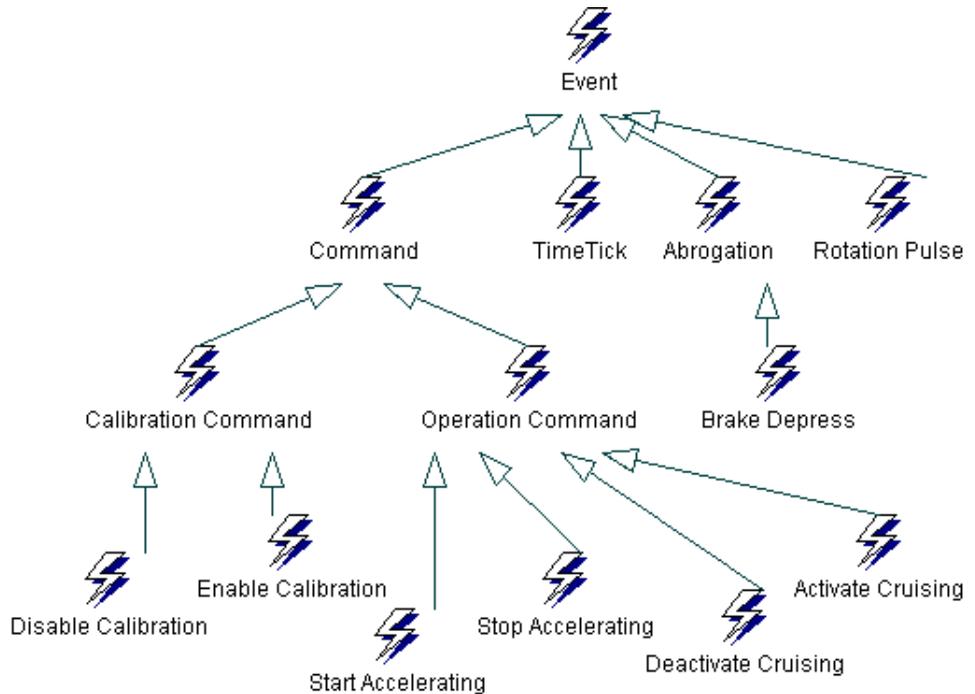
Table 30 Event Sheet diagram elements

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.

Figure 48 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 48 Event sheet diagram



Interaction diagrams

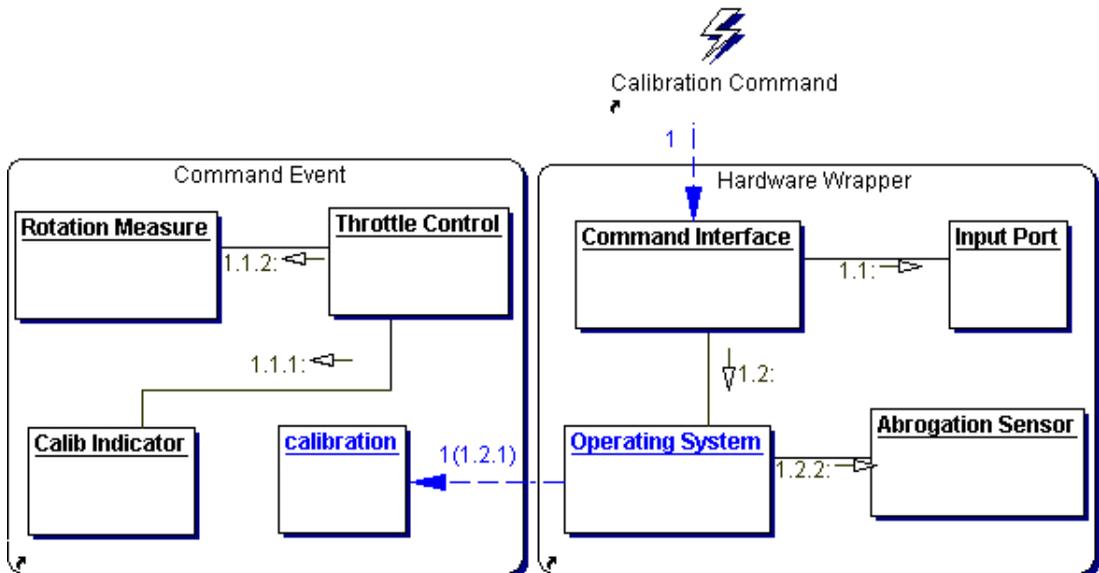
Real-time systems place special emphasis on the interaction of objects in different threads. Together's real-time interaction diagrams model object groups, or object interaction threads, and the events and messages that initiate program flow within a thread.

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.

The example diagram of [Figure 49](#) 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 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.

Figure 49 Interaction 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

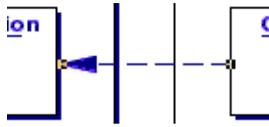
Together's real-time interaction diagrams support several kinds of asynchronous messages.

simple asynchronous

Simple asynchronous calls return immediately. (See [Figure 50](#).) 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.

Figure 50 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.)

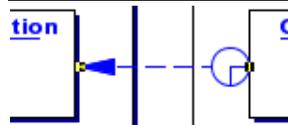
sender waiting reply

With a *sender waiting reply* message, the sender begins immediately to wait for a response. (See [Figure 51](#).) The sender waits indefinitely unless a timeout is supplied.

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.

Figure 51 Sender waiting reply message



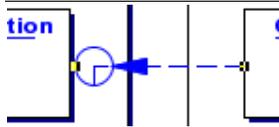
Properties of sender waiting reply messages include:

- **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 52](#).) The properties of receiver polling reply messages are the same as for sender waiting reply.

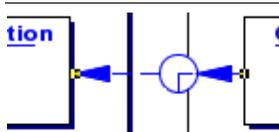
Figure 52 Receiver polling reply message



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. (See [Figure 53](#).) If the reply comes after the time stamp expires, the sender ignores the reply. 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).

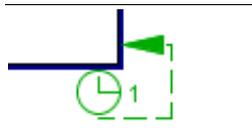
Figure 53 Time stamp message



process delay link

A *process delay link* delays thread execution for a set period of time. (See [Figure 54](#).) This could be through some mechanism like blocking on a time notification or putting the thread to sleep. Properties for a process delay link include the *delay* time.

Figure 54 Process delay link



Real-time audits

There are two 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:

1. From the main menu, choose **Tools | Audit**.
2. In the resulting audit dialog, check one or both audits from the *Real Time* group.
3. Click **Start**.

Figure 55 shows the audit selection for Java projects with SODS selected.

Figure 55 Real-time audit selection

Title	Abbreviation	
+ Coding Style		<input checked="" type="checkbox"/>
+ Critical Errors		<input checked="" type="checkbox"/>
+ Declaration Style		<input checked="" type="checkbox"/>
+ Documentation		<input type="checkbox"/>
+ EJB Specific		<input type="checkbox"/>
+ GUI Builder Specific		<input type="checkbox"/>
+ Naming Style		<input checked="" type="checkbox"/>
+ Performance		<input type="checkbox"/>
+ Possible Errors		<input checked="" type="checkbox"/>
- Real-Time		<input checked="" type="checkbox"/>
Requirements Traceability	RT	<input type="checkbox"/>
Stereotypes Of a Diagram Subtype	SODS	<input checked="" type="checkbox"/>
+ Superfluous Content		<input type="checkbox"/>

Simulation

Together real-time 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.

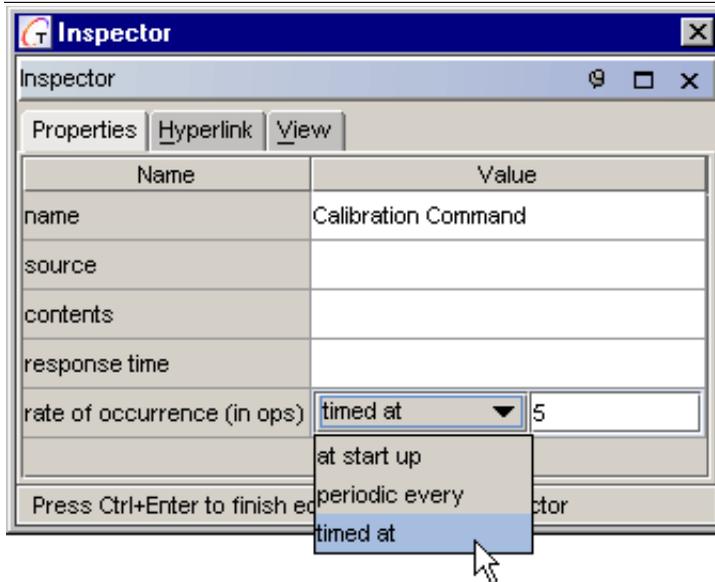
Preparing a simulation

Simulation requires an interaction diagram such as the diagram in [Figure 49](#). 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 56](#) shows setting event properties for the real-time sample that ships with Together.

Figure 56 Setting event properties



The ops simulation unit

All time-based properties use the *op* 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. Inter-process messages

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 preemptiveness 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 the 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 section³., “[Select a location for the exported file in the resulting dialog box.](#)”

Simulation controls

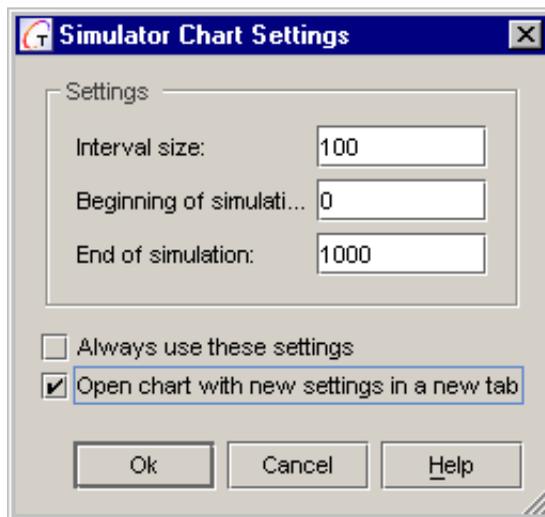
The real-time simulator has four controls located in the upper left hand corner of the simulation pane:

- **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:** Opens 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 57](#).

Figure 57 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 Real-Time provides the ability to export the interaction diagram simulator information to an XML file for use with other tools.

To export the simulator information:

1. Make sure that the interaction diagram to be simulated is open in the designer pane.
2. From the main menu, choose **Tools | Real-Time | Export to XML**.
3. 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.

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](#)” on page 234).
- 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](#)” on page 238).
- Exchange information between the XML and data models using the DTD Interchange module (“[XML Serialization](#)” on page 239).

It is assumed that you are familiar with the basics of XML. XML training is beyond the scope of this document. 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 the XML Schema, refer to the following documents at <http://www.w3.org/TR/>:

- *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)
- *XML Schema Part 2: Datatypes* (02 May 2001, Paul V. Biron, Ashok Malhotra)

XML structure diagrams

Together provides an XML structure diagram that you can use to create an XML structure definition from scratch.

Alternatively, you can import an existing DTD or XSD file as an XML structure diagram. 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.

Important If you want to use an XML structure diagram created in previous versions of Together, you have to convert the diagram to the new format. Failing to do so results in incorrect diagram behavior. To convert a diagram, choose **Tools | Convert to New Diagram Format** on the diagram right-click menu.

Content of XML structure diagrams

The content of the XML structure diagram is format-dependent. There are two possible formats: DTD and XSD. As you can see in [Table 32](#), the sets of elements for DTD and XSD formats are overlapping: the majority of design elements are common to both formats. However, certain elements are specific for a particular format, and are not enabled in the other format. For this reason, be careful when exporting a diagram to a DTD or XSD file, to avoid loss of information.

Table 32 XML structure 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: <ul style="list-style-type: none">• ? zero or one• * zero or more• + one or more• 1 required	
	Attribute: Adds an attribute to an element. This design element can be created using the toolbar button, or from the right-click menu for an element.	
	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 using the toolbar button, or from the right-click menu for an element.	

Table 32 XML structure diagram components (continued)

	Data type: Creates a visual design component for an arbitrary data type by means of inheritance (<i>type</i> field on 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
	Entity: Creates a visual design component for an entity.	DTD
	Notation: Creates identification for external binary entities format.	DTD
	Note	
	Note Link	

Creating an XML structure diagram

The XML structure diagram is part of the XML Support feature, which you can activate or deactivate as necessary. Before creating an XML structure diagram, make sure that the *XML Support* box is checked in the list of Activatable modules (Tools | Activate/Deactivate Features).

To create a new XML structure diagram:

1. On the main menu, choose **File | New** to open the Object Gallery.

Tip Alternatively, select a package in the Explorer, and choose **New | Diagram** on its right-click menu.

2. In the **General** category, select the **Diagram** icon, and click **Next**.
3. In the New Diagram dialog, click on the Together tab.
4. Select the **XML Structure** icon, enter the package name, diagram name, and description if necessary, and click OK.

Note You can also edit the diagram properties in the Inspector. To do so, choose the **Properties** command on the diagram right-click menu.

Creating design elements on the XML structure diagram

The basic technique for creating diagram elements is the same as for other diagrams. There are several ways to create design elements:

- Use the toolbar buttons.
- Choose one of the commands from the **New** submenu on the diagram or element right-click menu.

- Use the keyboard shortcuts listed in the **New** submenu on the right-click menu.

To create an element, attribute, or group:

1. Click the required button on the diagram toolbar.
2. Click on the diagram background to create the design element.
3. In the in-place editor, type 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 the 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. Choose **Properties** on the right-click menu of a design element and do all the necessary editing in the Inspector.

Format of the XML structure diagram

A 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** on the diagram right-click menu to open the Inspector.
2. Open the **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 the diagram. For example, if DTD format is selected, entities and notations disappear from the resulting diagram.

The reverse situation is slightly more complicated. Complex types and data types are deleted. If there are elements that inherit certain data or complex types, their attributes (if any) are added to those elements in the resulting diagram.

Example:

1. Create an 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 Element1 and change its base type (field base) to ComplexType1 in the Properties tab of the Inspector.
5. Change the format from XSD Schema to DTD, and observe that Attribute1 is added to Element1 in the resulting diagram.

Working with DTD-specific components

Entities, notation, and processing instructions are specific to DTD format only. Entities and notations are created using toolbar buttons. The Inspector for an entity provides the *external* checkbox. When *external* is checked, additional controls for external entities are displayed. In particular, there is a reference to a notation.

To assign a notation for an external entity:

1. Open the Inspector for an external entity.
2. In the *notation* field, click the file chooser button to open the Select Notation dialog.
3. Select 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:

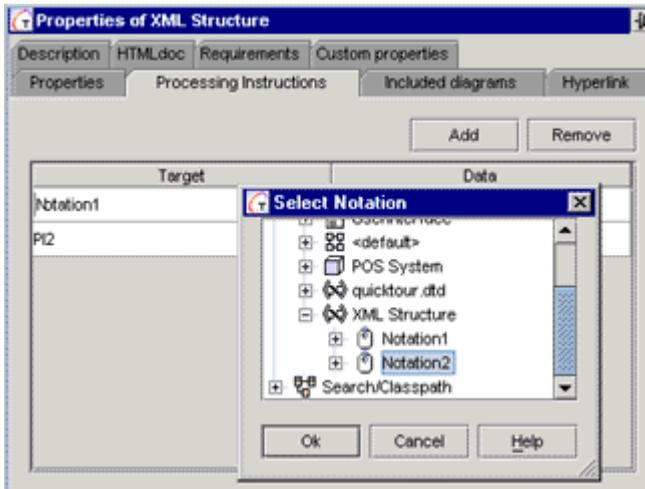
1. Right-click on the diagram background and choose the **Properties** command from the right-click menu.
2. In the Inspector, select the **Processing Instructions** tab.
3. Click the **Add** button.

To specify notation for a processing instruction:

1. On the Processing Instructions tab of the Inspector, select the processing instruction.

2. Click the 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 58 Processing instructions and notations



XML modeling step by step

Let us consider the creation of a sample XML model of personnel management data in XSD format.

Creating the XML structure diagram

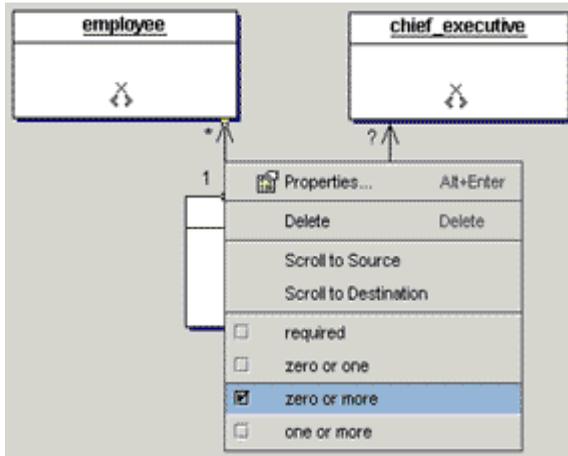
1. Activate the XML Support feature. Choose **Tools | Activate/Deactivate Features**, and make sure that the *XML Support* box is checked.
2. Create a new XML diagram. On the main menu, choose **File | New** to open the Object Gallery. In the **General** category, select **Diagram** and click next. On the **Together** tab, select **XML Structure**, name the diagram `personnel`, and click **Finish**.
3. Change the diagram format to XSD. Choose **Properties** on the diagram right-click menu to open the Inspector. On the **Properties** tab, set the *format* field to *XSD Schema*.

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* on the **HTMLdoc** tab.

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. Right-click on the *chief_executive* link and check the box *zero or one*, to specify that there can be only one chief executive.

Figure 59 Creating diagram elements and links



3. Add an attribute to each element. Choose **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 the attribute name to *identifier*. This attribute is a unique identifier of an employee. In order to verify or change the properties of the attribute, open the Inspector for the attribute, go to the Properties tab, and set the *occurs* field to *required*, which means that this property is mandatory.

Besides a unique identifier, each employee and the chief executive are characterized by name, position in the organization's hierarchy, URL, email address, and postal address. Create the following elements:

- *url* to a file that contains additional data, such as 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* 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 full name and date of birth. The attribute *Date of birth* is mandatory (the field *occurs* in Properties is set to *required*), and the field *type* is set to *CDATA*.

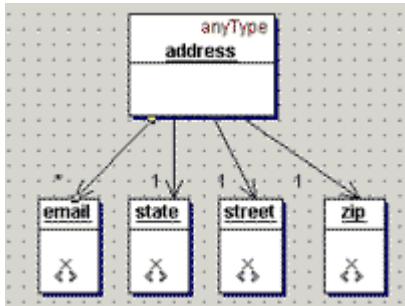
Note To insert the data type of an attribute, click the file chooser button in the Inspector and select the appropriate type in the *Search/Classpath | XML data types* directory.

Creating and using complex types and data types

Complex types enable you to provide special customized data types containing internal elements. Consider an address type that includes zip code, state, street, and several email addresses.

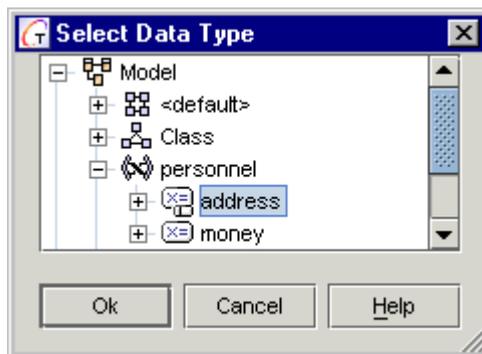
1. Create a Complex Type visual component. Rename it to *address*. Next, create elements *email*, *state*, *street*, and *zip*, and draw links from the *address* complex type to these elements. Set appropriate cardinality values for the links. State, street, and zip code are *required*, while the link to *email* has *zero or more* cardinality, which means that a person may have an unlimited number of email addresses.

Figure 60 Creating the *Address* complex type



2. Now create the element *address*, open its Inspector, select the *base* field on the Properties tab and click the file chooser button to open the Select Data Type dialog. Expand the *Model* node and select the *address* type under your *personnel* diagram.

Figure 61 Choosing the data type from the Select Data Type dialog

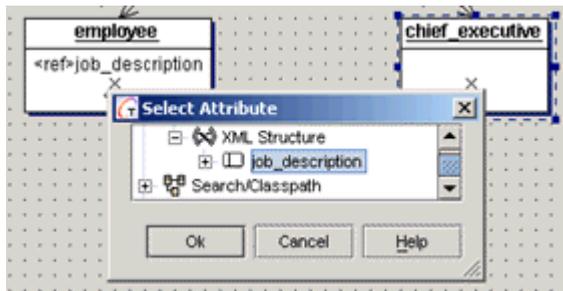


3. Create a new data type, and rename it to *money*. On the Properties tab of the Inspector, click the file chooser button to select the data type from the Model, Search/Classpath or Favorites.
4. Create the *salary* element. On the Properties tab of the Inspector, click the file chooser button for the *base* field, and select the *money* type under the *personnel* node of the Model tree. You can use the *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

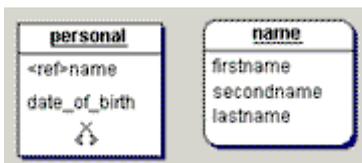
1. Click the Attribute button on the diagram toolbar and add the *job_description* attribute to the diagram. All elements of the diagram can refer to this attribute.
2. Add this attribute to the elements *employee* and *chief_executive*. Choose **New | Attribute Reference** on the right-click menu of the appropriate element.
3. In the Select Attribute dialog, choose the reference from the model and search/classpath tree. In the same way, you can create the global attribute *comments*, and add it to any element where needed.

Figure 62 Applying a global attribute to the diagram elements



4. Create the global group of attributes and call it *name*. Add the attributes *firstname*, *secondname*, and *lastname* to this group. These attributes have the `String` data type. *Firstname* and *lastname* are mandatory (the field *occurs* is set to *required*), while the attribute *secondname* 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 63 Applying Attribute group reference

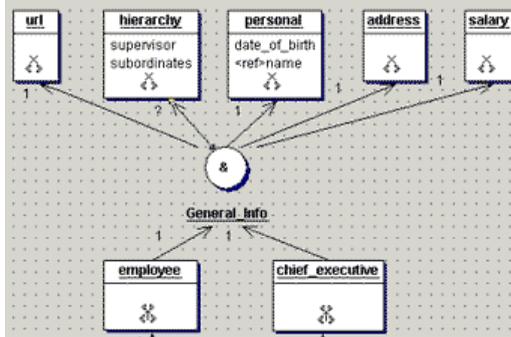


Creating groups

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 resulting links are entangled and difficult to read. A more elegant way to link multiple elements to multiple elements is by grouping.

1. Create a new group *General_Info*. By default, a Sequence group is created. Link the *employee* and *chief_executive* elements to this group. Right-click on the *employee* link, and check the *required* box in the right-click menu.
2. Link the group to each of the above elements. Right-click on the link to the *personal* element and check the *required* box, which indicates that personal information is unique and mandatory. Finally, right-click on the link to the *hierarchy* element and check the *zero or one* box on its right-click menu, thus indicating that an employee can exist without any subordinates or supervisors.

Figure 64 Linking diagram elements to a group of common information



Import and export operations

When the XML structure diagram is ready, it can be stored in the local file system. You have 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, choose the target format: **DTD** or **XSD**.
3. Click the **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.

Tip If the components of the XML structure diagram to be exported contain text in the Description tab of the Inspector, these descriptions are translated into comments in the resulting DTD file. However, notes on the diagram are ignored.

To change the target location:

Open the Inspector for the XML structure diagram to be exported.

1. Select the **Properties** tab.
2. 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* list. The files of the selected type are displayed in the tree-view.
3. In the tree-view, select the source file and click **Import**.

Important When exporting or importing DTD or schema files, keep in mind that some components are only enabled for the specific format. In particular, complex types and data types are allowed only in XSD format, while entities, notations and processing instructions are allowed only in DTD. 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 the activatable feature *DTD Interchange*.

To activate the DTD Interchange feature:

1. On the main menu, choose **Tools | Activate/Deactivate features**.
2. On the Together Features tab, check the *DTD Interchange* box.

Result: The XML Serialization submenu is added to the Tools menu. This submenu 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 Designer pane gets the focus. The set of enabled commands depends on the current diagram. For example, in a class diagram the Export ER Entities to DTD command is disabled.

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**.
2. In the Export dialog, specify the target location.
3. If needed, check the boxes to immediately create an XML structure diagram and open the generated DTD file in the Together 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 need to select the source DTD file.

4

PROGRAMMING WITH TOGETHER

- Chapter 15, “Editing”
- Chapter 16, “Using search facilities”
- Chapter 17, “Compiling and Running”
- Chapter 18, “Debugging”
- Chapter 19, “Configuring Together to support JDK 1.4”
- Chapter 20, “Using the UI Builder”
- Chapter 21, “Using Version Control”

Editing

Together comes with a built-in full-featured text editor that allows you to work with any of the supported languages. The editor is flexibly configurable.

This chapter includes the following topics:

- “Configuring the Editor” on page 243
- “Using the Editor” on page 245
- “Bookmarks” on page 247
- “Breakpoints” on page 251
- “Advanced Editor features” on page 252
- “Context Help” on page 258
- “Integrated XML Editor” on page 259
- “External Editor” on page 269

Configuring the Editor

You can configure the Editor using the Options dialog at any of the multiple configuration levels.

To configure the Editor:

1. On the main menu, choose **Tools | Options | [level]**, where level is the configuration level at which the settings are applied (Default, Project, or Diagram).

Note Alternatively, choose Text Editor Options on the right-click menu of the Editor pane.

2. Navigate to the **Text Editor** node, which includes the following subnodes representing the main categories of configuration options for the Editor:

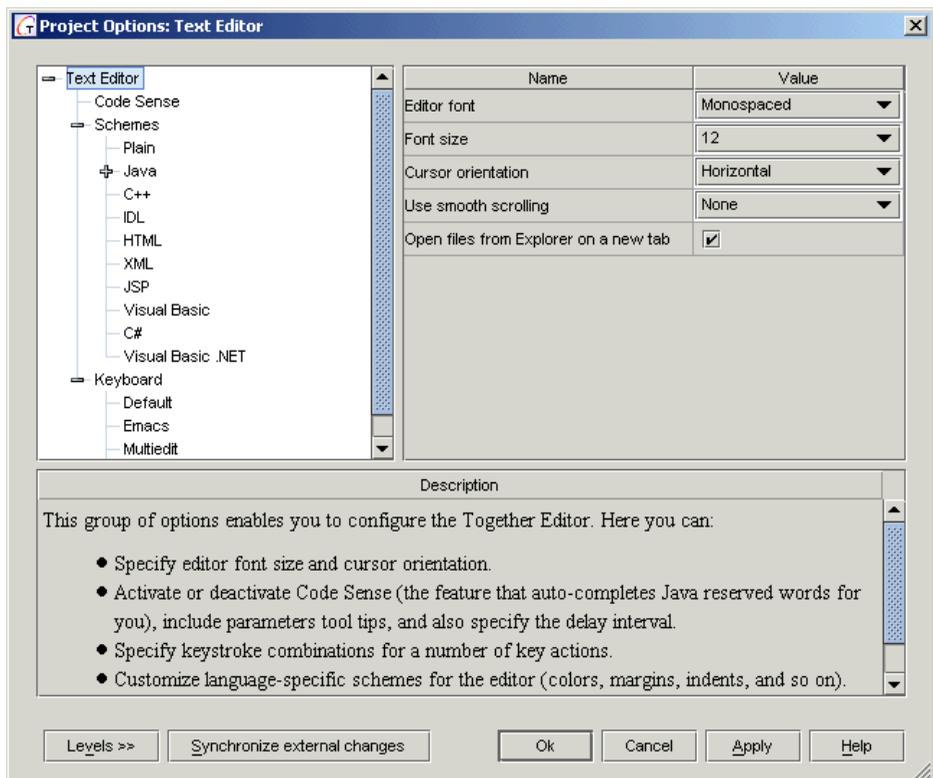
Code Sense. The feature that auto-completes reserved words in Java source code.

Schemes. Use these options to tune the Editor for working with different contents (plain text, Java, C++, Visual Basic, IDL, HTML, XML, JSP, C#, Visual Basic .Net). For each language, you can customize colors, indentation, text formatting and error highlighting. The Editor automatically picks up the particular scheme depending on the type of file loaded and focused in the Editor.

Keyboard. 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.

3. Set the options as required, and press **OK** to apply changes and close the dialog.

Figure 65 Editor Configuration options



Using the Editor

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 a new tab is a configurable option, assumed by default (*Options | Text Editor - Open files from Explorer on a new tab*). If this option is unchecked, 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 Designer. The source code opens in the Editor 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 right-click menu for the tab, choose **Preserve Tab**, while the active tab is selected. The current tab remains and a new tab named <untitled> opens. 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. This enables you to keep several source files from the same diagram open in the Editor pane at once.

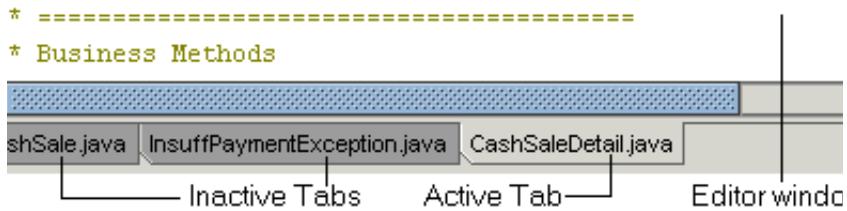
You can switch between tabs by clicking on the tab header at the bottom of the Editor pane.

To close a file:

1. In the Editor pane, select the tab of the file to be closed.
2. Right-click on the tab and choose **Close** on the right-click menu.

Tip The Close All command closes all open files.

Figure 66 Tabs in the Editor window



Editor commands are available on the Edit menu, on the right-click menu, and on the toolbar of the Editor pane. To speed up access to certain commands, use keyboard shortcuts.

Edit menu commands

Table 33 Edit menu commands

Command	Description	Shortcut
Undo	Rolls back several recent changes.	Ctrl-Z
Redo	Reinstates the last Undo operations.	Ctrl-Y
Cut	Removes selected text and puts it on the clipboard.	Ctrl-X
Copy	Copies selected text to the clipboard.	Ctrl-C
Paste	Inserts text from the clipboard at the cursor position.	Ctrl-V
Copy Image	Copies the current diagram to the clipboard in the selected format.	(none)
Delete	Deletes the selected text. (Restore using Undo.)	Del
Select All	Selects all the text in the currently focused file.	Ctrl+A
Insert Text File	Displays the standard Open File dialog to select 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 (Cut, Copy, Paste, Select All, Run), while others require some brief explanation.

Table 34 Commands on the Editor right-click menu

Command	Description
Select in Diagram	Highlights the diagram element that corresponds to the current line of the code.

Table 34 Commands on the Editor right-click menu (continued)

Command	Description
Browse Symbol	Displays the source code of project and library classes. For more information, see “Browse Symbol” on page 257 .
Collapse Lines	Collapses groups of lines of the specified type (comments, imports, methods, fields) and displays the first line in the group marked with a plus sign (+).
Expand Lines	Expands the nodes of the specified type (comments, imports, methods, fields).
Tools	Contains a submenu with a list of tools you can apply to the active tab. You can configure additional tools in the Tools options of the Options dialog.
Tools Format Source	Performs syntax formatting of the source code in the active tab. You can set formatting options in the Source Code options of the Options dialog.
Tools External Editor	Opens the content of the active tab in the pre-configured external editor. The external editor is configured in Tools - External Editor in the Options dialog.
Toggle Breakpoint	Sets or removes a breakpoint on the current line. For more information, see “Breakpoints” on page 251 .
Text Editor Options	Opens the Text Editor category of the Options dialog on the project level.
Version Control	Displays a submenu of version control commands. Available only when version control is enabled for the project.
Refactoring	Displays a submenu of various refactoring commands. The current context determines which refactoring commands are available.
Bookmarks	Displays a submenu of commands for setting, removing, or editing bookmarks. For more information, see “Bookmarks” on page 247 .
Override/Implement Method	Opens a dialog for selecting methods of the parent class to be implemented or overridden. For more information, see “Advanced Editor features” on page 252 .

Bookmarks

Together provides two types of bookmarks: global bookmarks that apply to the entire project, and local bookmarks that are used only within the currently opened file.

Global Bookmarks

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.

Once set, the bookmarks are stored in the project profile and will be reused when the project opens next time.

Bookmarks have lower priority than breakpoints. This means that if you set a breakpoint and a bookmark on the same line, the breakpoint overrides the bookmark and executable line icon.

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 **Bookmarks | Toggle Bookmark**, or use the keyboard shortcut.

The default bookmark icon is displayed in the margin. A default bookmark description is saved using up to the first 50 characters of the line. You can edit this description (for instructions, see [“Editing global bookmark descriptions” on page 249](#)).

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 **Bookmarks | Toggle Bookmark**.

Tip The following are additional notes for bookmarks:

- You can also remove bookmarks using the Edit Bookmarks dialog (for instructions, see the section [“Viewing, editing, and classifying global bookmarks” on page 248](#)).
- Undo/Redo does not apply to bookmarks.
- The Ctrl+M shortcut toggles 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 The Ctrl+D shortcut opens the Edit Bookmarks dialog.

Editing global bookmark descriptions

By default, the *Description* field for a bookmark contains the characters of the bookmarked line (up to a maximum of 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, right-click on the bookmark to reposition.
2. On the right-click menu for this bookmark, choose **Move Up** or **Move Down**.

Tip There are alternative ways to move bookmarks:

- On the toolbar of the Edit Bookmarks dialog, click the Move Up or Move Down buttons.
- 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 for marking constructors, another for the start of business methods, another for JavaBean getter/setter methods, and so on.

To change the associated icon of a bookmark:

1. Select the required bookmark in the Edit Bookmarks dialog.
2. On the right-click menu, choose **Change Icon** and choose an icon from the submenu.

The icon you select for each bookmark is displayed in the left 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 the Edit Bookmarks dialog.

2. On the right-click menu of the bookmark table, choose **Sort by Icon**.

Deleting bookmarks

You can delete one or several bookmarks from the list.

To delete bookmarks:

1. Open the Edit Bookmarks dialog.
2. Select the bookmarks to be deleted from the list.
3. On the right-click menu, choose **Delete**.

Tip There are alternative ways to delete bookmarks:

- Click the Delete button on the toolbar of the Edit Bookmarks dialog.
- Use the Delete key on the keyboard.

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.
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. (The corresponding diagram does *not* open.)

If the target file is already open in the Editor, the insertion cursor moves to the start of the bookmarked line.

Tip If you have a small number of bookmarks, you can navigate within a single file using keyboard shortcuts. Ctrl+NumPad + moves forward, and Ctrl+NumPad - moves backwards. Shortcuts do not work for the entire project.

Local bookmarks

Local bookmarks are fast and handy to operate. They are numeric instead of titled, meaning that they are numbered from 1 to 10. For this reason, there can be only ten local bookmarks per file.

Setting and removing local bookmarks

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 should be assigned a new number. Local bookmarks are not added to the list of bookmarks and are not displayed in the Edit Bookmarks dialog.

Tip Press the number 0 on the keyboard to display the number 10 on the bookmark.

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 local bookmarks

Navigating with local bookmarks is simple. Press **Ctrl+number**, and the cursor moves to the start of the bookmarked line that corresponds to that number.

Note Local bookmarks work with the main keyboard only, *not* with numeric keypad buttons.

Breakpoints

Breakpoints specify where to stop code execution during debugging to permit inspection of variables, expressions, class members, and so on. This feature of the integrated debugger is accessible from the Editor. The basic functions are outlined here. For in-depth information, see [Chapter 18, “Debugging”](#).

Setting and removing breakpoints

There are several alternative ways to set breakpoints in the Editor.

To set a breakpoint:

1. Place the insertion point cursor at the beginning of the line of code where the execution should stop.
2. Do one of the following:
 - Choose **Toggle Breakpoint** from the Editor right-click menu.
 - Click on the extreme left margin next to the line.
 - Press F5.

When a breakpoint is set, a red rectangle is displayed in the left margin, and the entire line is highlighted in red.

To remove a breakpoint, repeat the procedure for setting a breakpoint, using any of the techniques listed in step 2.

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 are displayed as grey rectangles.

To edit the properties of a breakpoint, place the insertion point in its line and choose the **Breakpoint Properties** command on the right-click menu.

Advanced Editor features

The Editor toolbar provides a number of buttons that significantly speed up the coding process. These buttons are:

- **Code Sense, Advanced Code Sense.** These features automatically complete your code. For details, see [“Code completion” on page 253](#).
- **Parameters Tool Tip.** This feature is also a part of code completion, and displays the list of possible parameters for the method under the cursor. For details, see [“Code completion” on page 253](#).
- **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 the Surround With button on the Editor toolbar.
 - Choose a construction from the list.
- **Toggle Comments.** Enables you to comment and uncomment the selected section of code.
 - Select the required lines of code.
 - Click the Toggle Comments button on the Editor toolbar.
- **Override/Implement Methods.** Enables you to choose methods of the parent class to be implemented or overridden.
 - Click the Override/Implement Method button.
 - In the Override/Implement Method dialog, choose the method to be overridden or implemented.
- **Expand Snippet.** Expands one of the pre-defined snippets into the code block that it represents. For details, see [“Snippets” on page 255](#).
- **Browse Symbol.** Enables you to view the source code of the library classes. For details, see [“Browse Symbol” on page 257](#).
- **Previous/Next Declaration.** Navigates through the list of declarations within the current class.

Code completion

The code completion feature is currently available in products with Java language support. It helps you auto-complete references to standard Java classes in Java code, significantly speeds up your coding, and helps you avoid syntax errors.

This feature includes Code Sense, parameter tool tips, and Advanced Code Sense. All functions are activated by toolbar buttons and keyboard shortcuts.

Using Code Sense and parameter tool tips

Code Sense and parameter tool tips work in the following modes:

- Code Sense is invoked by the keyboard shortcut (**Ctrl+Space** by default), or using the Code Sense button  on the Editor toolbar.
- Parameter tool tips are displayed by the keyboard shortcut (**Ctrl+F8** by default), or using the Parameters Tool Tip button  on the Editor toolbar.
- Both features are activated upon delay.

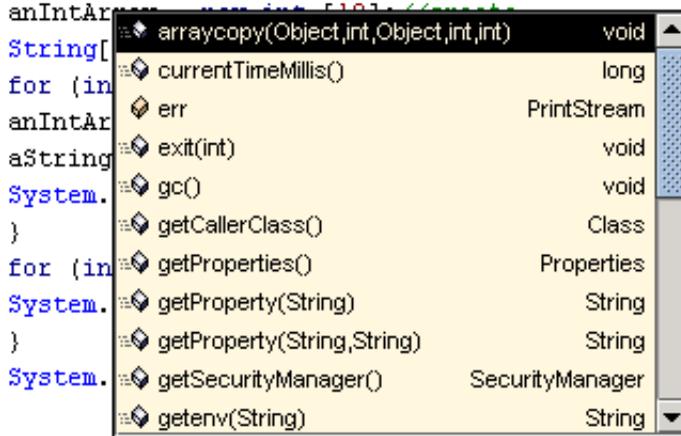
To configure activation upon delay:

1. Open the Options dialog (**Tools | Options | <level>**) and navigate to *Text Editor - Code Sense*.
2. Check the *Activate* option if you want Code Sense to be activated automatically upon delay.
3. In the *Delay* option, type the amount of time (in milliseconds) to wait before activating.

Example 1

1. Type `System.`
2. With the insertion point cursor placed after the period character, press **Ctrl+Space**. The list of available methods for this class is displayed.

Figure 67 Code sense



3. Select the desired method using the 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 using the mouse or arrow keys and press **Enter**. The name of the selected method is inserted into the line.

Example 3

1. Type `System.getProperty(`
2. Press **Ctrl+F8** to show the list of possible parameters above the opening bracket (or wait for it to appear after the pre-defined delay).

Using Advanced Code Sense

This function is activated by the **Shift+Space** shortcut, or using the Advanced Code Sense button  on the Editor toolbar. Relevant to the current caret position, Advanced Code Sense completes the code appropriately. Advanced Code Sense can perform type casting, complete expressions with the `new` keyword, and suggest commonly used names.

Type casting

Advanced Code Sense enables to cast an expression value to the required type.

Example:

```
String s = (<caret is here>
```

After pressing the keyboard shortcut or clicking the toolbar button, the code fragment is completed to:

```
String s = (String)
```

Instantiating new objects

Invoking Advanced Code Sense after the `new` keyword instantiates an object of the expected type.

Example:

```
StringBuffer buffer = new <caret is here>
```

After pressing the keyboard shortcut or clicking the toolbar button, the code fragment is completed to:

```
StringBuffer buffer = new StringBuffer(<caret is here>);
```

If the instantiated object has parameters, the list of possible parameter sets is displayed next to the opening parenthesis.

Suggesting commonly used names

Advanced Code Sense suggests commonly used names for 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 source code contains references to a class that belongs to a package or library in the project but has not been properly imported, this class name is highlighted in red, and a tool tip prompts you to perform the required import.

To invoke the Import Assistant:

1. Navigate to the highlighted class name to see the tool tip.
2. Press **Alt+Enter**.

Result: The import statement is added to the source code and the red highlighting is removed.

Snippets

Using Snippets significantly speeds up the process of application development. A Snippet is a name or abbreviation that represents a defined block of code. You can define as many Snippets as you want. Then, while coding in the Editor, type the

name of a 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, such as `if`, `for`, and `while`.

Defining Snippets

To define a snippet:

1. Open the Options dialog (**Tools | Options | <level>**) and navigate to *Text Editor - Schemes*. Select the node for the language you want to work with.
2. Select the *Snippets* option and click the **Edit** button to display the Snippets Editor dialog.
3. 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 a name in the *Abbreviation* field.
4. In the lower editing area, type the actual code that you want inserted in your source files when you use this Snippet later.
5. Check the *Fast Expand* box to enable expanding the Snippet by pressing Tab or Space after the keyword.

Tip The fast expand key is defined in the Text Editor options (*Text Editor - Schemes - Keyboard - <scheme>: Fast Expand Snippets*).

6. In the *Where to expand* field, select the area where the Snippet can be applied.
7. Click **OK** when finished, and close the Options dialog.

Deleting Snippets

To permanently delete a snippet:

1. Perform steps 1 and 2 listed above.
2. Select the Snippet you no longer need.
3. Click the **Delete** button. The Snippet is permanently deleted.

Using Snippets

To make use of a pre-defined Snippet:

1. Type in the name keyword.
2. Expand the Snippet in one of the following possible ways:
 - Press **Ctrl+J**, or a fast-expand key (**tab** by default). Note that the fast expand keyboard shortcut is configurable in the Options dialog, in *Text Editor - Keyboard - <scheme>: Fast Expand Snippet*.
 - Click the Expand Snippet button  on the Editor toolbar.

3. Replace the Snippet tags with the required expressions. Use the Next Snippet Tag button , or **Ctrl+Shift+J** to move to the next placeholders. Note that the Next Snippet Tag keyboard shortcut is configurable in the Options dialog, in *Text Editor - Keyboard - <scheme>: Next Snippet Tag*.

Tip You can press **Ctrl+J**, or click the Expand Snippet button  at the insertion point, and select a Snippet from the drop-down list.

Browse Symbol

The Browse Symbol feature enables you to view the source code 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 the 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. Working with library classes requires additional sourcepath customization.

To add the required libraries to the Source/Classpath of your project:

1. On the main menu, choose **Project | Project Properties**.
2. Open the Source/Classpath tab.
3. Click the **Add Path, Library, or Archive** button, and choose the required resource that contains source classes.

Tip When adding library classes, keep in mind the following:

- Simply adding an `src.jar` file does not help, because the directory structure of the `src.jar` file is different from the structure of package statements. For example, the upper level directory of `src.jar` file is `src`, which contains the `java` directory, and so on, while the package statements start at a lower level (such as `java`). For this reason, you need to unpack the source classes, or re-pack them in order to provide the same directory structure as used in the package statements.

Example: If the source classes are unpacked to the folder `C:\JDK_sources`, then the resource added to the source/classpath should be `C:\JDK_sources\src`.

- The library classes open in read-only tabbed pages.
- Forward and Back buttons,  and  in the Editor toolbar, switch between the file tabs and enable you to easily return to the original point of the browse sequence.

Context Help

The Editor enables you to create and use context help. The Context Help feature provides access to the JDK specification (for example, JDK 1.3 downloaded from <http://java.sun.com/j2se/1.3/docs.html>), and other documentation at your discretion (such as J2EE specifications, or project documentation generated using Together documentation generation facilities).

Important Context Help does not display HTML files with applets. If you generate project documentation using Generate HTML, do not switch to the frame mode if you are intending to use the documentation as context help.

Creating a help database

To enjoy the advantages of this feature, you need to create the database of libraries yourself. Help libraries are not supplied with Together. It is your responsibility to create a context help system according to the specific requirements. Each library is by default available for all projects. You can exclude some libraries from the global search, and make them available to 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 the **Libraries** tab and click **Add** to create a library.
3. In the Library dialog, specify the library name in the *Name* field, and choose the required encoding from the *Encoding* combobox. This enables the correct display of files that contain text in various languages.
4. Click the **Add** button to add files or folders to the *Folders* field. This opens the Select Folder or File dialog.
5. In the Select Folder or File dialog, select the required resource and click **OK**.

Note You can repeat this step to add as many files or folders to the library as needed.

6. Check the *Skip* box for libraries that you want to make unavailable on the global level.
7. In the Context Help Libraries dialog, choose the **Assign Libraries to Project** tab.
8. In the *Assign* column, check the libraries to assign to the current project.
9. 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, and 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. Create the project documentation (**Project | Documentation | Generate HTML**).
3. Choose **Help | Context Help Libraries**.
4. Create the library that contains JDK documentation:
 - Click the **Add** button to open the Library dialog.
 - In the Library dialog, click **Add** to add a 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 the **Add** button to open the Library dialog.
 - In the Library dialog, click **Add** to add a 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, press F1 with the cursor on a certain keyword to search the context help database.

You can find a detailed description of controls for the Context Help dialog in the online help system (click Help in the dialog).

Integrated XML Editor

The XML Editor enables 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 (for details, see [“Advanced Editor features” on page 252](#)):

- 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 on the Editor toolbar and right-click menu. You can find examples of mature XML files under `TGH\samples\xml`.

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 on 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. Select a node in the tree-view to automatically navigate 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 243](#).

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 XML Support feature.

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. Choose **Add Child** on the right-click menu and 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. Choose **Append** on the right-click menu and 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 the element where you want to add an attribute.
2. Choose **Add Attribute** on the right-click menu.

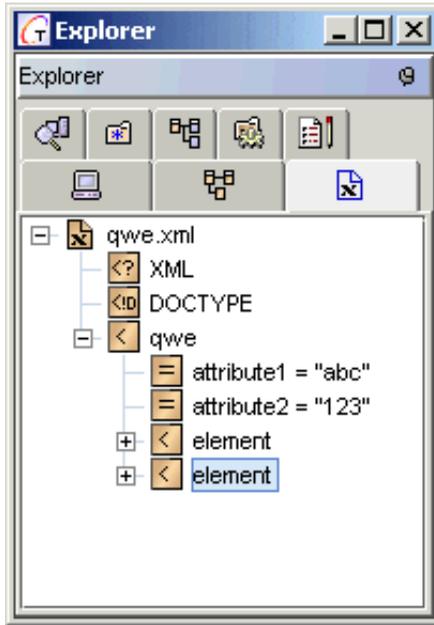
Result: The new attribute is added under the selected element, and the appropriate section is added to the source code.

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.

Figure 68 XML Structure tab in the Explorer



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.

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:

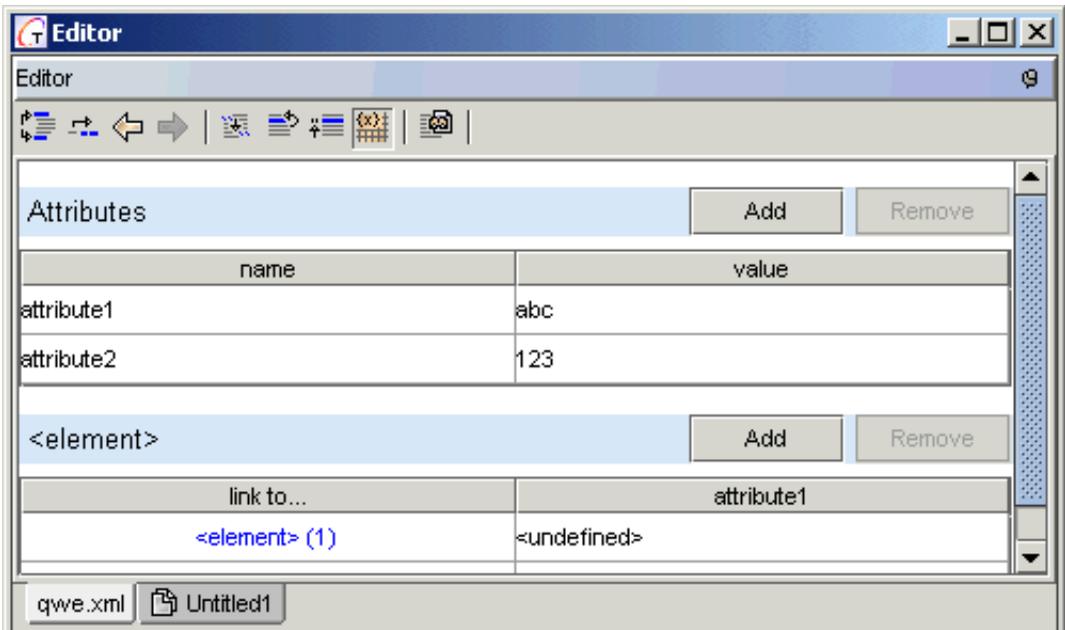
1. Right-click on the element to be moved.
2. Choose **Convert To** on the right-click menu and choose **Promote** or **Demote** from the submenu.

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 69 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 Model tab of the Explorer 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 right-click menu.
- Navigate to the desired file in the Explorer and choose **View** on the right-click menu for the node.

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. For details, see [“TagLib diagrams” on page 628](#).

Tips and Tricks

Bracket Highlighting

When you place the cursor on a line located *between* two parentheses, brackets, or curly braces, blue highlighting in the left margin of the Editor outlines the entire construction contained in the brackets. You can turn this feature on or off in the Options dialog (navigate to *Text Editor - Schemes - <Language>: Outline current source block*).

When you place the cursor *on* a bracket, the matching bracket is highlighted for easy navigation. Also, the bracket contents can be outlined in the left margin or highlighted with a background color. These features are configurable in the Options dialog (navigate to *Text Editor - Schemes - <Language>* and refer to the options *Highlight matching brackets* and *Highlight content of brackets*).

Swapping case

When you place the cursor somewhere inside a word, you can use a keyboard shortcut to automatically change the case of the whole word, or just the

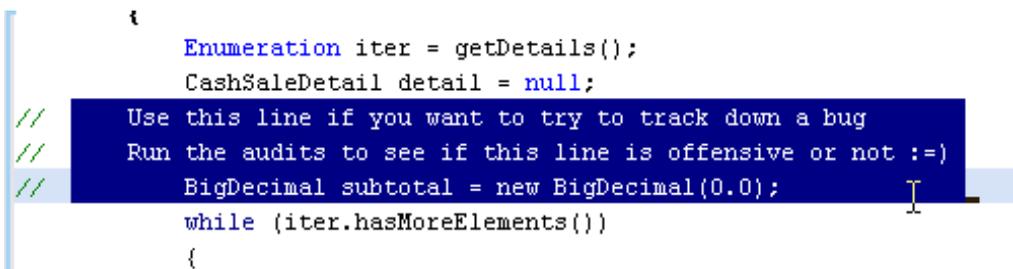
capitalization of the first letter. This feature is helpful when changing variables into constants, or classes into objects, and vice versa.

Ctrl+F3 changes the case of the word, and **Ctrl+Shift+F3** changes the capitalization of the first letter. The keyboard shortcuts are configurable in the Options dialog (navigate to *Text Editor - Keyboard - <scheme>* and refer to the options *Swap case* and *Swap initial capitalization*).

Selecting rectangular blocks

The Editor enables you to select rectangular blocks of text, as shown in [Figure 70](#). The keyboard shortcut **Ctrl+L** toggles rectangular block capabilities on and off.

Figure 70 Rectangular block selected in source code



```
Enumeration iter = getDetails();
CashSaleDetail detail = null;
// Use this line if you want to try to track down a bug
// Run the audits to see if this line is offensive or not :=)
// BigDecimal subtotal = new BigDecimal(0.0);
while (iter.hasMoreElements())
{
```

To work with rectangular blocks:

1. Press **Ctrl+L** to switch to rectangular block mode.
2. Select the desired block of text with the mouse, or hold down the Shift key and use the arrow keys.
3. Handle the selection as usual.
4. To switch back to line block mode, press **Ctrl+L** again.

Tip There is an alternative way to select a rectangular block, using the Alt key and the mouse pointer. Hold down the **Alt** key, and move the mouse pointer to select the required area. Use this method when you need to quickly select one rectangular block, instead of switching between rectangular block and line block modes.

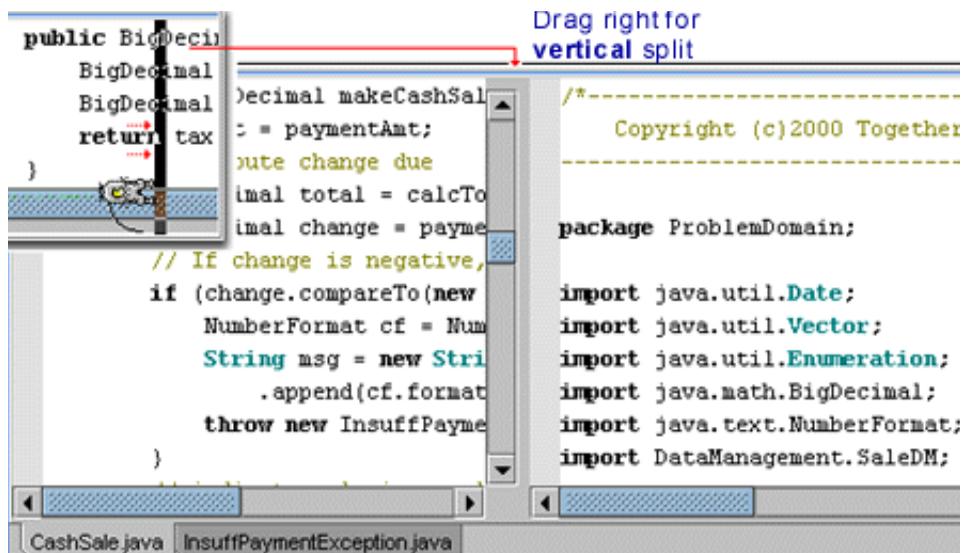
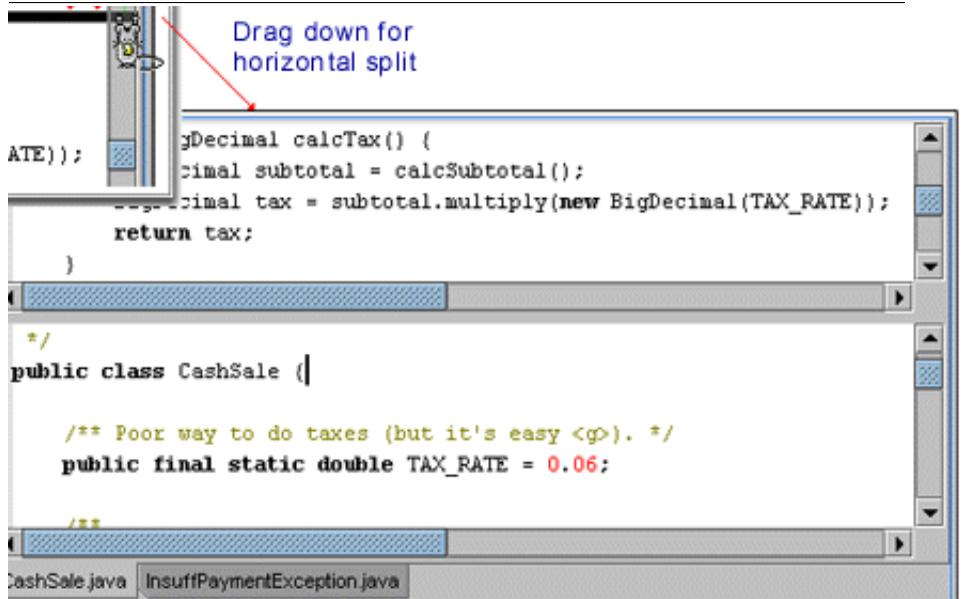
Splitting the Editor pane

For the ease of editing large files, split the Editor pane into two or four segments, each one with its own scroll bars. This way you can view different sections of a file simultaneously, as shown in [Figure 71](#).

To split the Editor pane horizontally, grab the upper right corner of the pane and drag the splitting line down. To split it vertically, grab the lower left corner of the pane and drag the splitting line to the right.

Lower left and upper right corners are marked with double arrows. Typing in any split pane is reproduced in all the other panes.

Figure 71 Horizontal and vertical split Editor pane



Showing and hiding the Editor pane

You can easily hide or show this pane using the View menu or the Editor Pane button  on the main toolbar. It is also possible to undock the Editor pane by clicking the thumbnail button.

The Editor pane may be hidden by default, depending on the *role* that you chose during installation or in the Options dialog. To set your role, open the Options dialog (**Tools | Options | <level>**) and navigate to *General: Role after restart*.

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 Designer pane. As you click on source-generating elements in the current diagram, the Editor content 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 diagrams 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 is hidden automatically.
- If multiple diagrams, some source-generating and some not, are open concurrently, the Editor pane is shown when you select a source-generating diagram in the Designer pane and hidden when you select any other diagram.

This default behavior prevails unless you override it 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 open on one of the Editor tabs, Together opens the existing tab on an attempt to open that file again.

Using the Editor with no open project

When you launch Together without a project, the Editor pane 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. 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.

Working with Java files with non- java extensions

The Editor makes it possible to work with java files that have arbitrary extensions. By default, the extension of the Java files is specified as “java”. If you wish to create a variety of the allowed new extensions, you have to specify them in the `resources.config` file. Further, you can create or open such classes in the Editor, and work with them in the Editor and Designer same way as with the ordinary java files.

Defining new extensions

To define new extensions:

1. In the Together installation, open the file `TGH\config\resource.config`.
2. Navigate to the line

```
resource.file.java_source.extension = "java"
```

3. Replace it with the line:

```
resource.file.java_source.extension.1 = "java"
```

4. Add as many new extensions as required, by inserting the lines in the following format:

```
resource.file.java_source.extension.n = "new_extension"
```

For example:

```
resource.file.java_source.extension.2 = "My_ext_2"
```

```
resource.file.java_source.extension.3 = "My_ext_3"
```

5. Save changes to the `resource.config` file

Result: upon restart of Together, you can open and edit files with extensions, specified in the resource configuration file.

Working with classes with non-java extensions

To create a class with an extension specified in the `resource.config` file, it is just enough to save a class with the new extension, using **Save As** command. For example:

1. Click on any tab in the Editor pane, and choose **Preserve tab** command, to create the <untitled> tab.
2. Type in the source code of the new class.
3. On the main menu, choose **File | Save As** command. Save dialog opens.
4. In *File name* field of the Save dialog, type in the file name with the pre-defined extension.
5. Click **OK**.

Result: the class is added to the diagram.

Important In version 6.0, the set of Editor functions for the classes with non-java extensions is limited. The Editor toolbar contains the following buttons only: Snippets, Surround with, Last Change, Go to line, Go to Super Method, and Context Help.

External Editor

In addition to the built-in editor, you can use another popular editor as an external tool. Win Vi is the default external editor and comes bundled with Together (\$TGH\$/bundled/winvi). Other editors should be installed separately.

Configuring the external editor

To assign a different external editor:

1. Open the Options dialog (**Tools | Options | <level>**) and navigate to *Tools - External Editor*.
2. In the *Language* field, choose the target language from the drop-down list.
3. In the *External Editor* field, choose the editor from the drop-down list. Make sure that the editor you choose is installed, and the command is configured correctly in the corresponding option.
4. Expand the *External Editor* node and select the *Show in menu* subnode. Check the boxes for the types of menus where you want to access the external editor command.

Using the external editor

To invoke the currently assigned external editor:

1. Right-click on one of the following:
 - A file in the Explorer.
 - A source-generating visual element in the Designer pane.
 - The Editor pane.
2. Choose **Tools | External Editor** on the right-click menu.

Using search facilities

Together enables searching for elements and strings throughout the diagrams and projects. Commands are available on the Search menu. This chapter includes the following topics:

- “Overview of search facilities” on page 271
- “Find and replace” on page 272
- “Find and replace in files” on page 272
- “Search on diagrams” on page 274
- “Query-based search” on page 275
- “Go to line” on page 277
- “Search for usages” on page 278

Overview of search facilities

Search results are displayed on separate tabs of the Message pane as a treeview, or as a plain list. This option is configurable in the Options dialog (navigate to *General: Display text search results as*). You can navigate through the search results in the Message pane, which are immediately highlighted in the Editor pane. Toolbar icons at the left of search result tabs enable expanding and collapsing the treeview nodes and saving the search results. Re-invoking a dialog starts a new search.

Note The availability of commands on the Search menu depends on the *role* that you selected during installation or in the Options dialog (*General: Role on restart*). For example, for the Business Modeler role, only Search on Diagrams and Query-Based Search are available.

Find and replace

Find and replace are the most basic search facilities. You can find the specified string in the current file and replace it with another string. Both functions support case sensitivity and searching for whole words or substrings.

To find a string:

1. On the main menu, choose **Search | Find** to open the Find dialog.
2. In the *Text to find* field, enter the search string.
3. Check the boxes *Case sensitive* and *Find whole words only* if required.
4. Click the **Search** button.

Tip To open the Find dialog, alternatively use the **Ctrl+F** keyboard shortcut, or click the Find button  on the main toolbar.

To replace a string:

1. On the main menu, choose **Search | Replace** to open the *Replace* dialog.
2. In the *Text to find* field, enter the search string.
3. In the *Replace with* field, enter the replacement string.
4. Check the boxes *Case sensitive* and *Find whole words only* if required.
5. Click the **Replace** button.

Tip To open the Replace dialog, alternatively use the **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

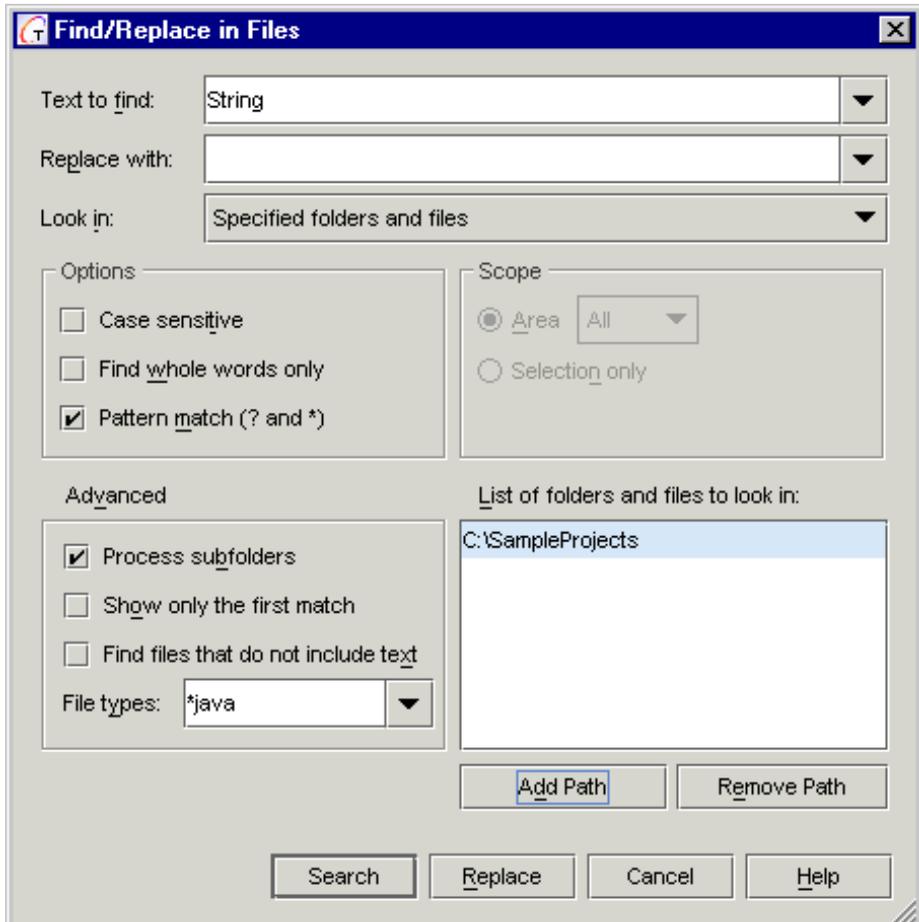
This feature enables you to find and replace occurrences of the search string in a specified range.

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 replacement string.

4. From the *Look in* list, 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 boxes if necessary:
 - *Case sensitive*: Searches for text that matches uppercase and lowercase characters.
 - *Find whole words only*: Matches the whole text and ignores substrings.
 - *Pattern match*: Uses wildcards. Pattern match may include an asterisk (*) to represent any number of characters, and a question mark (?) to represent a single character.
6. In the *Scope* section, choose *Area* 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* list.
7. Check *Process subfolders* to search nested directories. Note that this checkbox is only enabled for searching in specified folders and files.
8. Check *Show only the first match* to confine your search to the first occurrence of the search string.
9. Check *Find files that do not include text* to perform an inverted search. This checkbox is enabled for all search ranges, except for the current file.
10. If searching in specified folders and files, use the **Add Path** button to specify the folders and files to be searched. Click **Remove Path** to delete a selected path.

Figure 72 Find/Replace in files dialog



Search on diagrams

This function enables you to search the current diagram or all opened diagrams for the specified string in a certain scope.

To search for a string on diagrams:

1. On the main menu, choose **Search | Search on Diagrams**. Alternatively, use the **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 boxes if necessary:
 - *Case sensitive*: Searches for text that matches uppercase and lowercase characters.

- *Find whole words only*: Matches the whole text and ignores substrings.
 - *Pattern match*: Uses wildcards. Pattern match may include an asterisk (*) to represent any number of characters, and a question mark (?) to represent a single character.
4. In the *Scope* section, choose the current diagram or all opened diagrams.
 5. Click **Search**.

Query-based search

This advanced feature enables you to find project elements (packages, classes, variables, and others) based on filters or queries. The filters are created using logical expressions. The Query-Based Search dialog provides basic facilities for building queries.

Note Because 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 non-source generating diagrams (for example, use cases or actors), and to class diagrams created in projects with the Design language selected as default.

A 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 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 the element property, and <operation> is =, !=, <,  
or >.
```

If the conditions are applied to strings, it is possible to use a pattern match (wildcards). Pattern match may include an asterisk (*) to represent any number of characters, and a question mark (?) to represent 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 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 is performed first. Such an expression returns the value `Class1`.

If you need to change the priorities of operations, create expressions with brackets.

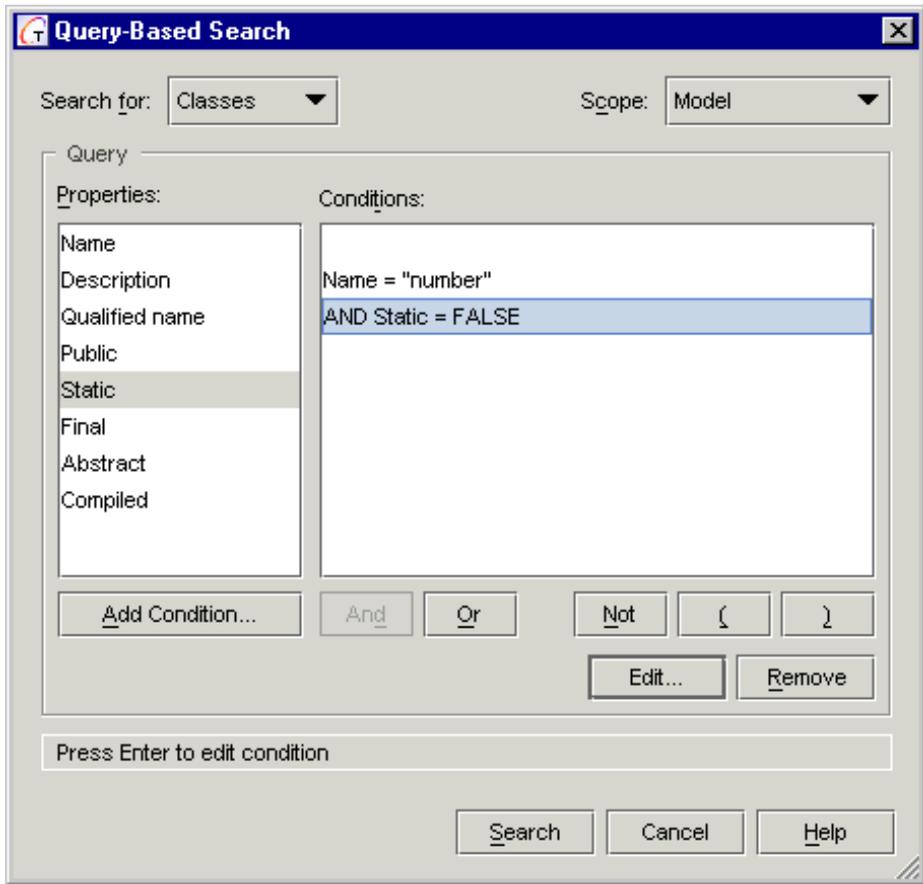
To filter out an element:

1. On the main menu, choose **Search | Query-Based Search** to open the Query-Based Search dialog.

Tip Alternatively, use the keyboard shortcut **Ctrl+Shift+Q**.

2. In the *Search for* list, choose which element to find. The possible options are: packages, classifiers, classes, interfaces, operations, and attributes.
3. In the *Scope* list, choose where to look for the element. The possible options are: model, classpath, whole project, current package or result set.
4. Create a query that will help you find 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 create a logical expression.
5. Click **Search**.

Figure 73 Query-Based Search dialog



You can find a detailed description of the dialog controls in the online help system (click the Help button in the dialog).

Go to line

This is the most basic command for navigating directly to a specified line in the Editor.

To navigate to a line:

1. On the main menu, choose **Search | Go to Line**, to open the Go to Line dialog.

Tip Alternatively, click the Go to Line button  on the Editor toolbar, or use the keyboard shortcut **Ctrl+G**.

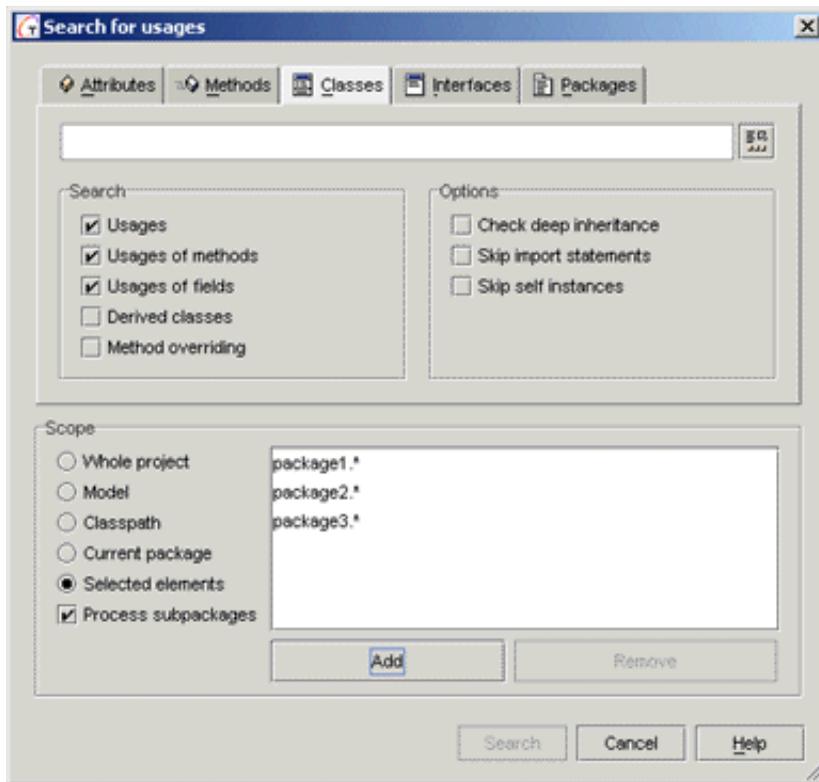
2. Type the target line number.

3. Click OK.

Search for usages

The Search for Usages command enables you to track how an element is used in a project.

Figure 74 Search for Usages dialog



The Search for Usages dialog contains tabs that enable you to organize searches for packages, classes, operations, and attributes. Each tab has its own set of controls, described in detail on the appropriate page of the online help system.

To find the usages of an element:

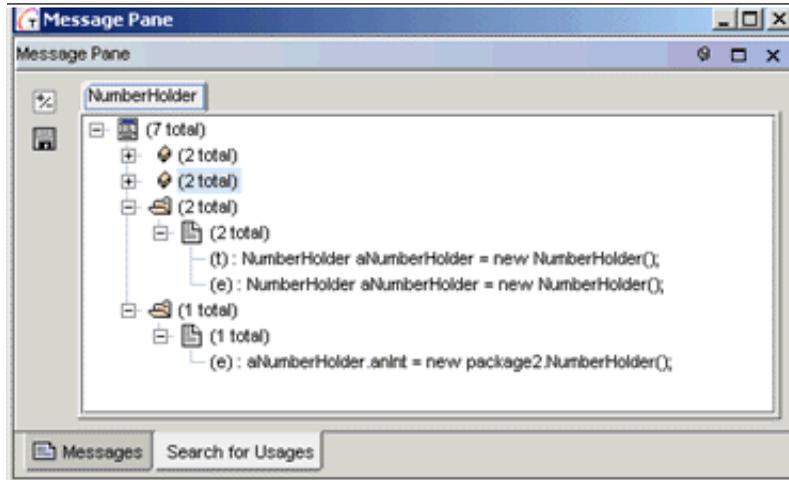
1. On the main menu, choose **Search | Search for Usages**.

Tip Alternatively, choose **Search for Usages** on the right-click menu of the Designer or Editor pane, or use the keyboard shortcut **Ctrl+Shift+U**.

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 search for.
4. In the *Options* section, check specific usages to skip, if necessary.
5. In the *Scope* section, choose the range of the search.
6. Click **Search**.

The results are displayed on the *Search for Usages* tab in the Message pane, 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).

Figure 75 Search for Usages results in the Message Pane



Compiling and Running

This chapter includes the following topics:

- “Overview of compiling facilities” on page 281
- “Executing the compile and make tools” on page 282
- “Configuring and using the standard compiler” on page 282
- “Running programs in Together” on page 285

Overview of compiling facilities

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 the Options dialog 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 the Options dialog to point to a different compiler.

To configure compile and make tools, open the Options dialog (**Tools | Options | <level>**) and refer to the *Builder* options.

Executing the compile and make tools

Execute the default Java compile/make tool from any of the following places:

- **Selection menu.**
 1. Click the diagram background or 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 tab.** Click Make  and Rebuild  buttons on the Builder tab of the Message pane.

Tip Alternatively, use the 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. If 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.

Tip You can navigate through the error messages in the Builder pane using keyboard shortcuts. With the Builder pane having focus, press **P** to move to the previous error message, and **N** to move to the next error message.

Configuring and using the standard compiler

On all platforms, Together installs Java 2(tm) SDK version 1.3, which contains the built-in compiler. This compiler is always used by default regardless of the operating system or Java machine that is used. The main class of this compiler resides in `JDK_HOME/lib/tools.jar`.

Compile and Make commands use this compiler by default, as mentioned above. 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*).

If you want to use another compiler tool, follow the instructions provided in the section [“Configuring and using an alternative compiler”](#) on page 284.

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 during compiling, 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. The default value of the *-bootclasspath* option depends on the Java Runtime library of the project (for details, see the section “[Adding Resources](#)” on page 103). You can change the cross-compilation options as required using the Options dialog.

To modify cross-compilation options:

1. Choose **Tools | Options | <level>** on the main menu.
2. In the Options dialog, navigate to *Builder - Built-in Javac - Compiler options - Cross-compilation: Boot class path (or Extension directories)*.
3. Click the path editor button to the right to open the Edit Path dialog.
4. In the opened dialog window, use **Add path**, **Add Zip/JAR** and **Remove** buttons to create the required path, and click **OK**.
5. In the Options dialog, click **OK** to apply the changes and close the 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. In the *VM version* field, choose the platform, and specify boot classpath, and extension directories as described above.
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 the Run command on the main menu. Running a program requires specifying the 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 the run configuration or just specify the class with `main()` method, VM options, and program arguments.
3. Click **OK** to run the program.

Tip If a class contains the `main()` method, Run command is available on the right-click menu of such class in the Model tab of the Explorer, in the Editor and in the Diagram Designer. For example, if `Class1` contains the method `main()`, the right-click menus of this class include command **Run Class1**.

Run/debug configurations

Large-scale projects may have numerous main classes and a significant number of parameters. Together helps simplify the process by means of the Run/Debug Configuration feature, which enables you 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 Alternatively, use the keyboard shortcut **Shift-F10**.

2. Click **Add** to open the Arguments and Parameters dialog.
3. Click on the Class, Applet, or Servlet tab as required.
4. Enter the run configuration name.
5. Specify arguments and parameters:
 - For applications: the class with the main method, program arguments, and VM options.
 - For applets: the applet class and parameters, VM options and frame size.
 - For servlets and JSP: the 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 configurations as needed, and assign one of them as the default one. The default configuration is highlighted in bold.

Debugging

The full-featured integrated Java debugger enables you to debug your projects inside Together. This chapter includes the following topics:

- “Overview of Debugging” on page 287
- “Starting a debugger session” on page 288
- “About the Run/Debug tab” on page 288
- “Controlling program execution” on page 289
- “Setting breakpoints” on page 291
- “Evaluating and modifying variables and expressions” on page 293
- “Using watches, threads and frames” on page 295
- “Working with monitors” on page 297
- “Attaching to a remote process” on page 297

Overview of Debugging

The debugger in Together provides the following features:

- **Breakpoints.** You can stop at any line of the source code, including *Logging* and *Pass Count* features. The Debugger provides 5 kinds of breakpoints: line, exception, class, method, and attribute breakpoints.
- **Command execution.** Run, Pause, Continue, and 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.** You can evaluate variables and expressions, and change their values while running in the Debugger.
- **Threads.** You can browse the state, methods, and variables of a thread.
- **Frames.** You can work with the information of the current frame.
- **Skip classes.** You can specify classes that should be skipped. Shows the classes that are already loaded.
- **Remote process.** Attaches to a remote process when you specify address and transport.

There are two ways to access the Debugger commands:

- On the Debugger toolbar, which is accessed using the **Run | Debug** command on the main menu.
- Keyboard shortcuts (See “[Compile and Run/Debug shortcuts](#)” on page 826.)

Starting a debugger session

Tip Get prepared to the Debugger session:

- Before starting a debug session, make sure you have set up all the needed breakpoints. If the breakpoints are not in place, the debugger session does not start. For more information, see “[Using breakpoints](#)” on page 290.
- You can use your favorite JDK instead of the default one, which has been specified during installation. To change JDK, use **Options dialog | Run/Debug | JDK Home**.
- Make sure to add all the necessary sources to the project, to be available for stepping through the code during the Debugger session. To specify the path to java source files (both in directories or jar archives) use **Options dialog | Run/Debug - Sourcepath**.

To start the debugging session:

1. Select **Run/Debug Configuration**, if necessary.
2. Choose **Run | Debug** on the main menu or use the keyboard shortcut **Ctrl+F9**.

This starts your projects in debug mode and automatically shows the Run/Debug tab. The Run/Debug tab significantly speeds up access to the information you could need during the debugging process.

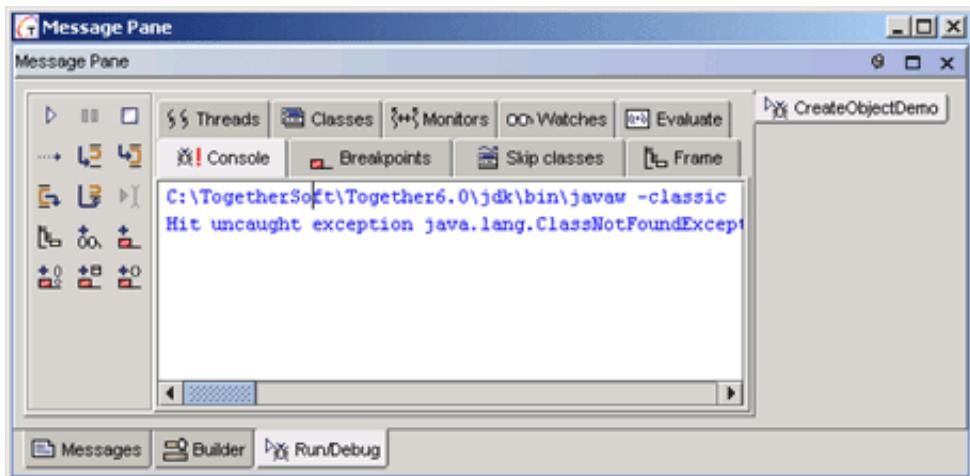
About the Run/Debug tab

The Run/Debug tab includes the following elements:

- **Console.** Displays the Java console.

- **Threads.** Threads viewer. Displays all the running threads in detail.
- **Classes.** Displays a hierarchy of loaded classes.
- **Monitors.** Displays synchronization monitors.
- **Watches.** Watch window for class members/expressions.
- **Breakpoints.** Displays the list of breakpoints.
- **Skip classes.** Contains the list of “disabled for debug” classes.
- **Evaluate.** Enables evaluating expressions and modifying values (including method executions) when the Debugger is in a 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, and so on.

Figure 76 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	Shortcut
Pause Program	⏸	Pauses program and passes control to the Debugger.	

Table 35 Debugger commands (continued)

Command	Icon	Description	Shortcut
Continue Program		Resumes program execution.	Ctrl+Shift+F2
Stop Program		Terminates the program and debugging session. Detaches from a remote process.	Ctrl+F2
Toggle Smart Step		Debugger performs a “smart step.” Set up smart step behavior in the Options dialog (navigate to <i>Run/Debug - Debugger - Smart Step behavior</i>).	
Step Over		Skips debugging of the method currently under the cursor. The method executes and returns results.	F8
Step Into		Forces debugging of the method currently under the cursor. Debugger stops at the first line of this method.	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

Using 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, and so on.

The Together Debugger provides the following types of breakpoints:

- line
- exception
- class
- method
- attribute

At a breakpoint, you can examine data with the aid of watches. For details, see [“Using watches, threads and frames” on page 295](#).

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 on the Run/Debug tab.

To set a breakpoint at the selected line:

- In the source file, do one of the following:
 - Press **F5** or click on the left margin next to the selected line.
 - On the Editor right-click menu choose **Toggle Breakpoint**.
- On the main menu, choose **Run | Add Breakpoint**.

To set a breakpoint on 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 during program execution in the Watches, Threads, and Frames tabs. When an attribute is encountered, the Add Attribute Breakpoint command is displayed on the right-click menu of these tabs.

To set an attribute breakpoint:

1. Click the 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**.

Note Attribute breakpoints can also be added on the other Debugger tabs.

Modifying breakpoint properties

After the breakpoints are set, you can disable or enable them. To do this, check or uncheck the *Enabled* box for the appropriate breakpoint on 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 the Breakpoints tab in the Debugger.
2. Right-click on the breakpoint to be modified.
3. Choose **Breakpoint Properties** on the right-click menu of the selected breakpoint.
4. In the opened Breakpoint Properties dialog, specify stop execution if necessary, the number of passes through the breakpoint before stopping, conditions, and also whether this stop should be logged or not.

Tip Alternatively, open the Breakpoint Properties dialog using the Editor right-click menu, or the appropriate button on the breakpoints toolbar.

In addition to the Breakpoint Properties dialog, it is possible to modify all types of breakpoints on 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 boxes *Stop Execution* and *Log Message* may not be disabled simultaneously. At least one of these boxes must be checked. For a detailed description of the fields, refer to the description of the Breakpoint Properties dialog in the online help (click the Help button in the dialog).

An attribute breakpoint has two properties that are modified in a slightly different way. These properties are: *Stop on Read*, which allows you to break application when an attribute is about to be read, and *Stop on Write*, which allows you to break application when an attribute is about to be written. Both options are enabled by default.

To modify an attribute breakpoint:

1. Choose the 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** or **Disable (Enable) Stop on Write**.

Evaluating and modifying variables and expressions

Together allows you to access variables during program execution.

To enable evaluation:

1. Set breakpoints in the desired locations of your code.
2. Choose **Run | Run in Debugger** on the main menu.

Result: The **Evaluate/Modify** command becomes enabled on the Run menu, and on the Editor right-click menu.

Not only variables, but whole expressions and function calls can be evaluated during the debugging session. You can select whole constructs in the source code and see their values in the Evaluate tab of the Debugger.

For example:

- `status`
- `frameSize.width`
- `frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2)`
- `instanceof`

Important 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 reports that the variable is out of range.

To evaluate an expression:

1. In the Editor pane, select the expression to be evaluated
2. On the Editor right-click menu, choose the **Evaluate** command. The appropriate entry is added to the Evaluate table, showing the location, type, and value of the object in question.

3. In the course of the debugging session, evaluate the expressions, clicking the right-arrow button in the second column of the Evaluate table.

To change values of the expressions:

1. In the *Expression* column, double-click on the entry to be modified.

Tip Alternatively, press F2 to open the inline editor.

2. Type in the new value of the selected expression.
For example: if the expression is `screen.width`, type in `screen.width=100`
3. Click the right arrow in the next column for the changes to take effect.

Figure 77 Results of evaluating expression

Expression	Location	Type	Result
status	problem_domain.CashSale.clearValues(...)	int	0
frame.pack()	user_interface.CashSalesApp.<init>() [...]	void	<void value>
Toolkit.getDefaultToolkit().ge...	user_interface.CashSalesApp.<init>() [...]		<name not in scope: 'Toolkit.getDefaultToolkit'>
frameSize.width	user_interface.CashSalesApp.<init>() [...]	int	326
screenSize.width=1024	user_interface.CashSalesApp.<init>() [...]	int	1024
frame.setLocation((screenS...	problem_domain.CashSale.clearValues(...)		<name not in scope: 'frame.setLocation'>
instanceof	problem_domain.CashSale.clearValues(...)		<Encountered: "instanceof" Position: 1. Was expecting c
tax	problem_domain.CashSale.clearValues(...)	jav...	null

Displaying structured context

While your program runs in the Debugger, you can check the value of each object. Navigate the cursor to the desired object, and after a small delay the description of this object is displayed.

The form of presentation is defined in the Options dialog (navigate to *Run/Debug - Debugger: Show variable as tool tip*). If the box is checked (by default), the evaluation window shows the entire structure and value of an object. Expand the nodes to reveal the constituent properties.

If this option is unchecked, the evaluation window shows only the address or value of the object.

Evaluating arrays

When evaluating an array, you can specify the bounds of the display range.

To define the display range:

1. Right-click the array name in the evaluation window
2. Choose **Change Display Range** on the right-click menu.

3. Enter starting and ending element numbers, or an asterisk to display the entire array.

Using watches, threads and frames

Watching class members, inspecting objects, frames and threads is an important side of the debugging process.

To enable watches, threads and frames:

1. Set breakpoints in the desired locations of your code.
2. Choose **Run | Run in Debugger** on the main menu.

Result: The **Add Watch** and **Show Current Thread/Frame** commands become enabled on the Editor right-click menu. Watches, Threads and Frames tabs open in the Debugger.

Using watches

The Add Watch command on the Editor right-click menu is 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, right-click the expression you want to watch.
2. Choose **Add Watch** on the right-click menu. The Add Watch dialog opens with the selected expression displayed in the *Expression* field.
3. Enter the watch description, if necessary.
4. Click **OK**.

To add a new watch in the Debugger:

1. Choose the **Watches** tab and right-click on the table.
2. Choose **Add Watch** on the right-click menu. The Add Watch dialog opens.
3. In the *Expression* field, specify the expression to be watched.
4. In the *Description* field, enter the description of the watch, if necessary.
5. Click **OK**.

You can add as many watches as needed.

A watch has its own 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 the value of the watch. This command is not available for 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.

Changing the 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. The Change Display Range command on 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 a watch for an array.
2. On the right-click menu of the watch, choose the **Change Display Range** command.
3. In the Change Range dialog, specify the array bounds. Enter an asterisk (*) to display the entire array.

Changing values

Having selected a watch on the Watches tab, you can change the value of the variable. The values are changed in-place, or using the Change Value command on the watch right-click menu. Note that string values must be entered in quotes. Otherwise, any modifications will be ignored.

Changing the display format

For integer variables, it is possible to toggle between decimal and hexadecimal representation. Use the complementary commands Show Decimal Value / Show Hexadecimal Value on the watch right-click menu.

Similar modification features are available on the 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 during program execution. Using the two buttons at the top of the Threads tab, you can display all current threads and frames, display the current thread only, or display all threads.

The Frames tab represents a frame stack of the current thread being resumed. This tab displays information that is a subset of the Threads tab. This spares you from switching to the Threads tab, and enables you to operate with the current frame.

Use the drop-down list at the top of the Frames tab to choose the location of the currently suspended thread.

Working with monitors

The Monitors feature implements synchronization monitors. The Monitors tab of the Debugger displays the monitors owned by the threads, and helps detect deadlocked threads, or threads that are waiting 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 monitors cannot be displayed, depending on the VM in use. For example, the default HotSpot Client VM does not support this feature.

To view monitors, add the `-classic` option to the VM options.

Monitors in deadlock and non-deadlock states

A monitor in a non-deadlock state and the thread that owns this monitor are denoted with a yellow key. The thread object of synchronization and the owner threads are expandable, as on the Threads tab of the Debugger.

A 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 using 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 -  
Xrunjdpw: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 -  
Xrunjdp: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 type 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 the port number, and click **OK**.

You can find an example of remote debugging in the `TGH\bin\win32\DebugExample.bat` and `TGH\bin\win32\DebugExample1.bat`.

Configuring Together to support JDK 1.4

In order for Together to properly recognize Java 1.4 syntax, you need to follow the instructions in this chapter to specify the run-time library and configure syntax support and the compiler tool. This chapter includes the following topics:

- “Specifying `rt.jar`” on page 299
- “Enabling Java 1.4 syntax support” on page 300
- “Configuring the compiler” on page 300
- “Configuring Ant Runner” on page 301

Specifying `rt.jar`

First, you have to specify the `rt.jar` from the Java 1.4 installation. There are two possible cases: a project is not yet created, and a project already exists.

For a new project

If a project does not exist, specify the path to the appropriate `rt.jar` using the Options dialog:

1. On the main menu, choose **Tools | Options | Default Level**
2. In the *General | New Project: Java Runtime Library* option, specify the required java run-time library, for example:

```
C:/j2sdk1.4.0/jre/lib/rt.jar.
```

This will add the Java 1.4 `rt.jar` in the JDK Configuration under the Project Properties for the new project.

For an existing project

If a project already exists, you have to modify the existing project file `*.tpr`:

1. In the project folder, open the `*.tpr` file.
2. Modify the line `Root.#`, where the path to `rt.jar` is specified, and type in the correct path to the `rt.jar` from the Java 1.4 installation. For example

```
Root.0=C:/j2sdk1.4.0/jre/lib/rt.jar
```

Enabling Java 1.4 syntax support

To enable support of the Java 1.4 syntax:

1. On the main menu, choose **Tools | Options | <level> - Source Code -Java**.
2. In the Java page, check the option *Support Java 1.4 syntax*.

Configuring the compiler

The built-in compiler does not compile the SDK 1.4 source code, so you have to configure an external compiling tool.

Configure external tool

Configure one of the existing Together external tools for work with `javac 1.4` (e.g. `Tool1`):

1. On the main menu, choose **Tools | Options | <level> - Tools**.
2. Select the **Tool1** node.
3. Specify the options below:
 - Set *Language* to *Java*
 - In the *Tool1* field, type in the name of this tool (e.g. *Javac 1.4 Compiler*)
 - In the *Command* field, choose the path to `javac` (e.g. `C:\JDK1.4\bin\javac.exe`)
 - In the *Parameters* field, type in the following string:

```
-source 1.4 -classpath  
"$:classpath$$PS$$SOURCEPATHS$$PS$$DESTINATION$" -d "$DESTINATION$"  
$FILELIST$
```

- In the *Filter the tool's output* field, choose *Javac compiler* from the combo box.

Configuring the Builder

Configure the builder to use the specified tool as the default compiler:

1. On the main menu, choose **Tools | Options | <level> - Builder**.
2. Select the **User Compiling Tool** node.
3. Specify the options below:
 - In the *Tool* field, select your predefined compiling tool from combobox (*Javac 1.4 Compiler* in our case)
 - Check the option *Use by default*, for using your tool as the default compiler

The settings for the *Java Runtime Library* and *Support Java 1.4 syntax* will provide you with Code Sense (Code Completion) and proper syntax highlighting. The *User compiling tool* option enables you to compile SDK 1.4 source code from Together.

Configuring Ant Runner

Together has a built-in ANT integration (Ant Runner). Using this facility requires certain spade-work to be done.

Setting Java 1.4 variables for Ant Runner

Ant process requires Java 1.4 executable to be in PATH, and JAVA_HOME variable to point to the JDK 1.4 root directory. Thus, it is necessary to modify the Ant starter, depending on the platform.

For UNIX systems:

1. In the Together installation, open the file `TGH/bundled/tomcat/bin/ant`
2. After the topmost line `#!/bin/sh`, add the following two lines

```
JAVA_HOME=/usr/local/jdk/sun/j2sdk1.4.0
JAVACMD=$JAVA_HOME/bin/java
```

For Windows systems:

3. In the Together installation, open the file `TGH/bundled/tomcat/bin/ant.bat`
4. After the topmost line `@echo off`, add the following two lines

```
set JAVA_HOME= [path to java home in your machine, e.g. c:/j2sdk1.4.0]
set JAVACMD=%JAVA_HOME%/bin/java.exe
```

Activating Ant Runner

To enable the ANT integration, follow these steps:

1. On the main menu, choose **Tools | Activate/Deactivate Features**.
2. Click on the Early Access tab.
3. Check *AntRunner* box.

Configuring compiler tool for Ant Runner

The built-in Ant integration does not work for compiling SDK 1.4 source.

Thus, if you need to use ANT, you have to configure the special external tool.

1. On the main menu, choose **Tools | Options | <level> - Tools**.
2. Select any unused Tool slot, for example the **Tool2** node.
3. Specify the following options:
 - In the *Command* field, specify the path to ANT executable file (e.g. C:\Ant1.4\bin\ant.bat)
 - In the *Parameters* field, type in
`-v -classpath "$:classpath$$PS$$SOURCEPATH$$PS$$DESTINATION$"`

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 303
- “Activating the UI Builder” on page 305
- “Setting UI Builder options” on page 306
- “Creating visually editable UI classes” on page 307
- “Using the UI Designer” on page 310
- “Designing menus” on page 330
- “UI Builder audits and metrics” on page 345
- “Generating user interface documentation” on page 346
- “Customizing the Toolbox component palette” on page 347
- “Java features supported in designable classes” on page 350

Overview of the UI Builder

The UI Builder is connected 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 results using the UI Builder.

The UI Builder is an *activatable feature* in Together, meaning that you can turn it off when not needed to conserve system resources.

Important The UI Builder depends on the RWI2 Inspector. This module is activated by default, when the UI Builder is activated. Never deactivate the RWI2 Inspector - this will make the UI Builder work incorrectly.

When activated, the UI Builder commands appear on the main View menu, a button 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.

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.

Any component selected in the Designer pane, is highlighted in the treeview of the Components Explorer, and vice versa.

Component Explorer

The Component Explorer displays a treeview of the three possible categories of UI components. These are:

- **UI Components**
The visual UI elements, or the UI Components per se (labels, buttons, combo boxes and other controls) are listed in this node.
- **Menu Components**
This node contains the various menu elements (menu bar, popup menu, menu items etc.)
- **Non-Visual Components**
The various UI elements that have no visual representation (for example, be responsible for connection or queries to a database)

Tip Visual Components can be added directly to the allowed locations of the container.

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`.

1.(“What-You-See-Is-What-You-Get”)

Activating the UI Builder

The UI Builder is an *activatable feature*. It is activated by default. This section explains the steps for activation and deactivation, and shows you what to look for in the user interface to know when the UI Builder is activated.

To activate the UI Builder feature:

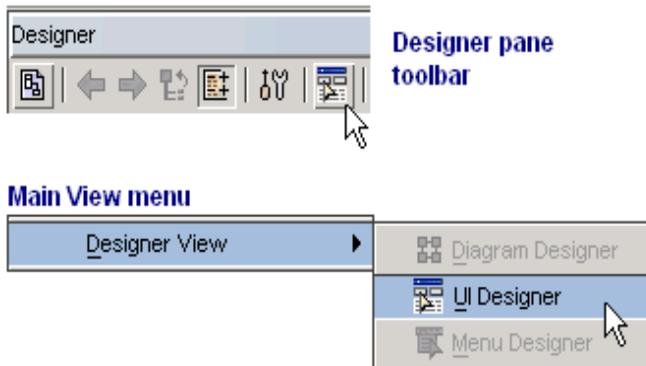
1. On the main menu, choose **Tools | Activate/Deactivate Features**.
2. On the Together **Features** tab, check *UI Builder*.

Note In some pre-release builds, the UI Builder may appear on the **Early Access** tab of the Activate/Deactivate Features dialog.

3. Click **OK** to activate the UI Builder module.

The UI Builder button is now enabled on the Designer pane toolbar and the UI Builder command is added to the main View menu, as shown in [Figure 78](#).

Figure 78 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.

To deactivate the UI Builder feature:

1. On the main menu, choose **Tools | Activate/Deactivate Features**.
2. On the Together **Features** tab, clear the *UI Builder* box.
3. Click **OK** to confirm deactivation of the UI Builder module.

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. Choose 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 choose 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 are treated as *visually editable* and are loaded into the 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, and so on in the source code it creates or generates, and you want to use Together to work on code created with that version.
- Your organization 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 names that UI Builder requires to be present in any class. This allows UI Builder 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 generates 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 generates initialization code for new objects. Choose to have the object initialized at its declaration point or at the beginning of the class 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 preliminary steps:

1. Open a project. For instructions, see [“Opening a project” on page 33](#) and [“Creating new projects” on page 97](#).
2. Focus on a source code file in the Editor that represents a visually editable UI class. Normally you would do this by selecting the relevant class in a class diagram in the Designer pane. The next section provides more information on creating visually editable UI classes.

What is a visually editable class?

Any file submitted to the UI Builder is scrutinized to determine whether it is suitable for visual editing. Such file should have *.java extension, and should contain a designable (or visually editable) class.

A designable class must meet certain requirements on the source code level:

- The class should contain the initialization operation defined in the currently selected IDE profile (**Tools | Options | Default (or Project) - UI Builder: IDE Profile**).
- Its superclass should have either default (parameterless) constructor, or Creator Class, specified in the BeanInfo.
- The class scope must be declared *public*.

Creating UI classes

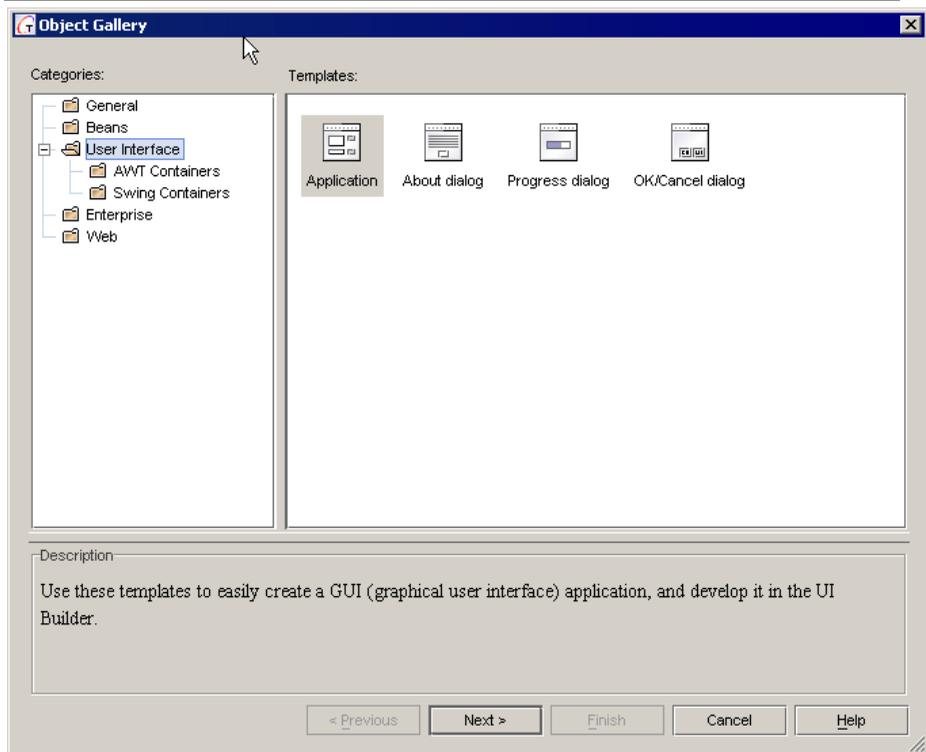
The *User Interface* category of the Object Gallery (**File | New: User Interface**) provides a quick way to create a number of designable UI classes. There you will find both AWT and Swing containers. It also provides a number of templates for more robust things such as applications and dialogs. (See [Figure 79](#).)

To create a designable class:

1. On the main menu, choose **File | New: User Interface**.
2. Click on the desired category.
3. In the *Templates* area, choose the template, and click **Next**.
4. In the dialog window, fill in the necessary information, following the instructions provided by online help. Click **Finish** to complete.

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 79 Object Gallery categories for UI components



Alternatively, use the Choose Pattern dialog to create designable classes:

1. On the diagram right-click menu, choose **New | Class by Pattern**.

2. In the Choose Pattern dialog, expand the *UI Components* node.
3. Choose the desired pattern, and click **Finish**.

Tip It is also possible to turn a non-UI class into a designable class by applying some of the UI patterns. To do that, choose **Choose Pattern** on the right-click menu of the class, and apply the desired pattern.

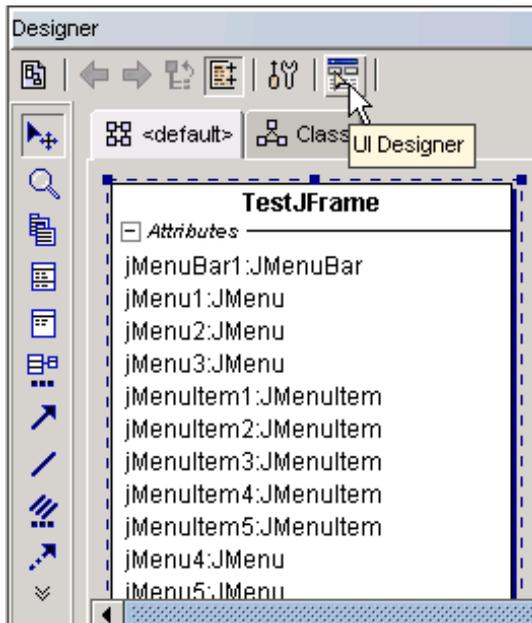
Using the UI Designer

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 65.)

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 Designer pane has three possible views, as shown in [Figure 80](#): Diagram Designer, UI Designer, and Menu Designer.

Figure 80 Diagram designer view



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.

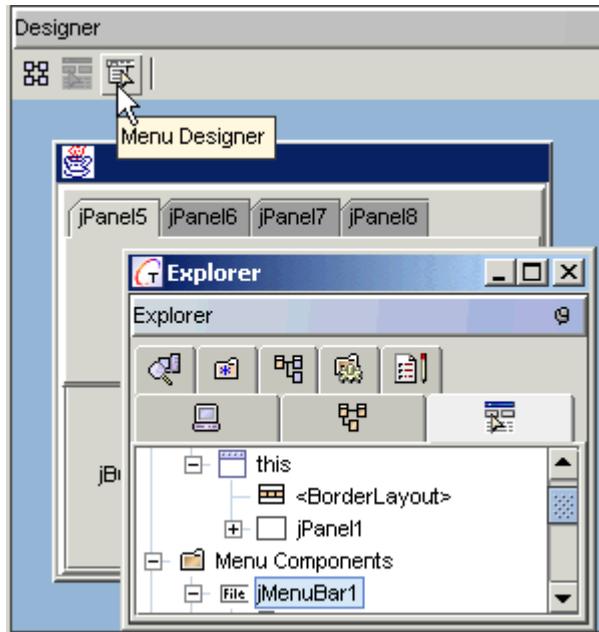
To work on a visually editable class in UI Designer view:

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 button on the Designer pane toolbar. If using the Editor, choose **UI Designer** on the Editor right-click menu.

Note Alternatively, choose **View | Designer View | UI Designer** on the main menu.

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 307).

Figure 81 UI Designer view



Menu Designer view

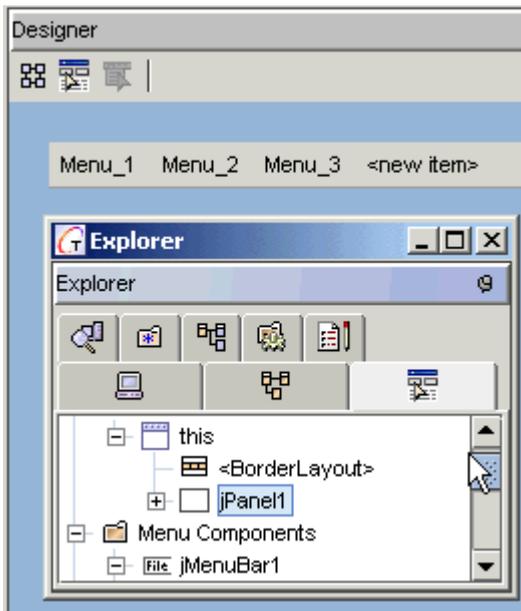
When a container class you are working on contains a menu component, develop the menu visually in Menu Designer view.

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 button is enabled on the Designer pane toolbar and the Menu Designer command is enabled on the main menu (**View | Designer View | Menu Designer**).
2. Click the Menu Designer button on the Designer pane toolbar, or choose **Activate Designer** from the right-click menu for the menu component in the UI Component Explorer.

Tip Double-click a menu component in the UI Component Explorer to switch to Menu Designer view. To switch back to UI Designer view, double-click a visual UI component instead of a menu component.

Figure 82 Menu Designer view



Adding components from the Toolbox

The Toolbox is available in the UI Designer and Menu Designer views. 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 list of available UI components provided in the Toolbox, search Online Help for *Toolbox*.

The procedure of adding visual, non-visual and menu components are slightly different.

To add visual components from the Toolbox:

Provided that a visually editable class is already created, as described earlier in this chapter, you can add a component to it.

1. Open the page of the Toolbox that contains the desired components.
2. Click the component icon in this page of the Toolbox.
3. Move the pointer over the target container on the Designer pane. The pointer changes to the drop shape, and the container changes to show the current layout (the default layout is BorderLayout).
4. Click inside the container frame to place the component.

Result: the component is allocated in the container, and an entry is added to the UI Components node of the Components Explorer.

Tip You can drop a visual component on the background of the UI Designer view. In this case, the component is not visible. You can later place it to a container using Cut-Paste in the Components Explorer (or just drag and drop it to the desired container).

Menu components are added in a different way.

To add menu components from the Toolbox:

Provided that a visually editable class is already created, as described earlier in this chapter, you can add a menu component to it.

1. Open the menu page of the Toolbox that contains the desired menu components.
2. Click the component icon in this page of the Toolbox.
3. Move the pointer over the background on the Designer pane. The pointer changes to the drop shape.
4. Click on the background to place the component.

Result: the menu component is created, and the Menu Designer icon becomes enabled on the UI Builder toolbar. Now you can design the menu, and, when ready, place it in the container. This procedure is described in details in the section [“Setting a menu bar into the container frame” on page 332](#).

Note You can customize the contents of the Toolbox. For more information, see [“Customizing the Toolbox component palette” on page 347](#).

Editing component properties

Every UI component, represented by a Bean, has a set of properties that control how a component behaves at runtime. Properties can be edited at design time and manipulated with code at runtime.

There are two ways to customize a UI component:

- By using *property editors*. Each Bean property has its own property editor, associated with it.
- By using *customizers*. Customizers are used where property editors are not practical or applicable. Unlike a property editor, which is associated with a property, a customizer is associated with a Bean.

The two ways of editing properties are described in the following sections: , [“Editing properties using Customizers” on page 314](#), and , [“Editing properties using Property Editors” on page 316](#).

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 BeanInfo class of the selected component, plus all the properties shown in the *General* level.
- **Hidden:** The Inspector displays only properties flagged *Hidden* in the BeanInfo class of the selected component.
- **Read-only:** The Inspector displays only properties flagged *Read-only* in the BeanInfo class of the selected component.

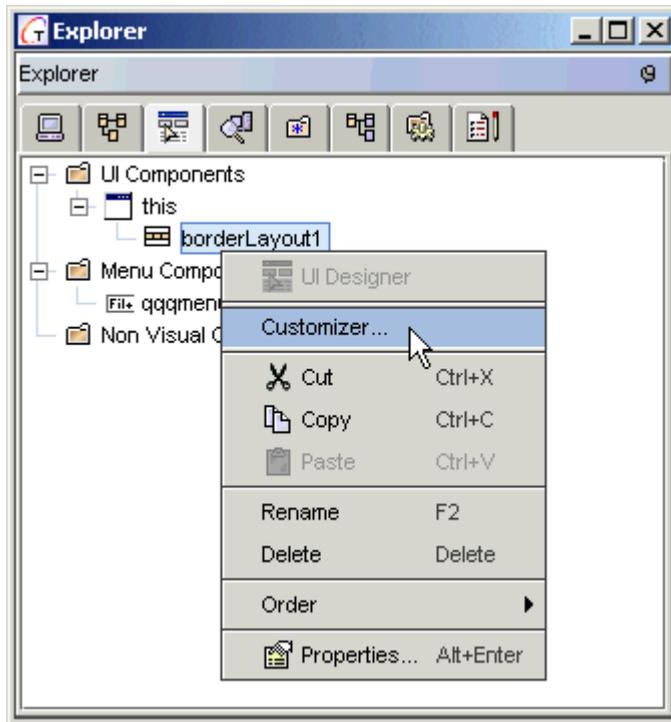
To change the exposure level in the Inspector:

1. Right-click 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 submenu.

Editing properties using Customizers

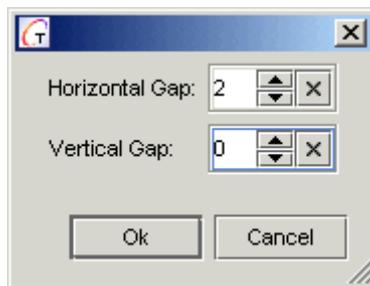
Sometimes properties are not sufficient to represent configurable attributes of UI components. If a UI component has a Customizer class, associated with it via its BeanInfo class, the Customizer command is enabled on the right-click menu of this component in the Components Explorer, as shown on the [Figure 83, “Customizer command in the Components Explorer” on page 315](#).

Figure 83 Customizer command in the Components Explorer



The appearance of the customizer dialogs depends on the specific implementation. A sample Customizer dialog is shown on the [Figure 84, “Customizer dialog for a BorderLayout container”](#) on page 315.

Figure 84 Customizer dialog for a BorderLayout container



As you enter the required values, the Customizer directly repaints the relevant properties of the displayed object. The further behavior of the customizer depends on the button pressed to complete operation.

- **OK:** if you press this button, the changes that you have done to the object, are committed to the source code.

- **Cancel:** changes are not committed to the code, but properties are changed with the nearest update. Default values are not restored with Cancel button.

Editing properties using Property Editors

Properties of UI components are displayed 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.

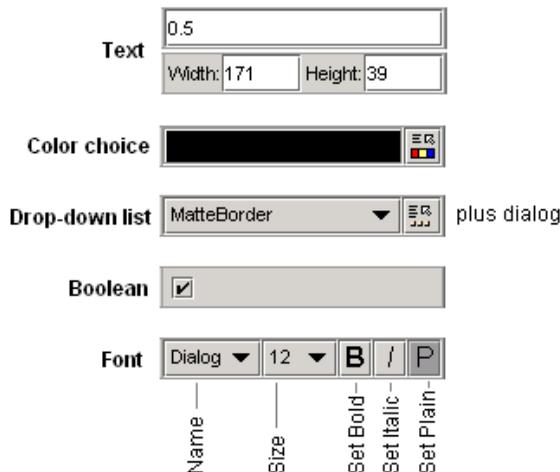
To view the Inspector:

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.

There are several types of property editors ranging from simple text fields to drop-down lists, color choosers, and entire dialogs (Layout Manager, for example).

[Figure 85](#) illustrates the different types.

Figure 85 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, such as Bean Chooser or File Selection. 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 your layout will be consistent on various operating platforms.

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.

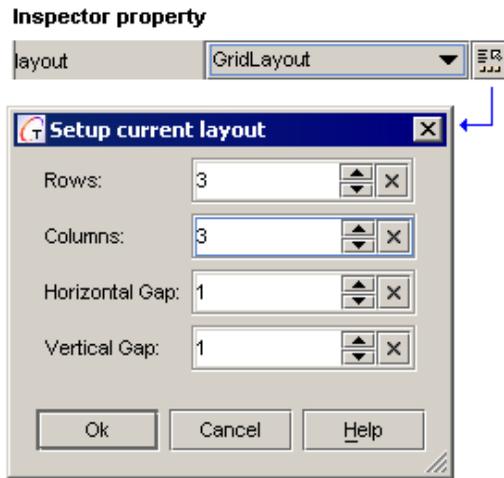
To view and change the layout property:

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. Also set the size of the horizontal and vertical gaps between them.

Figure 86 The layout setup dialog



Layout-specific setup dialog

The visual setup shown in [Figure 86](#) 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 are displayed 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 you 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.

Containers with different layouts

Once you have set the *layout* property for a container, place components into it (for instructions, see [“Adding components from the Toolbox” on page 313](#)). To visually resize containers such as panels, select the component and drag the corners or sides.

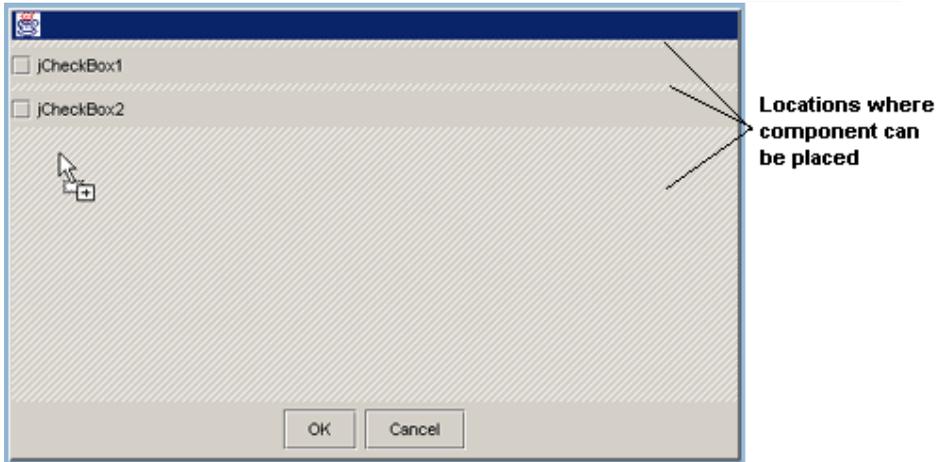
When you select a component in the Toolbox and then hover the pointer over the container, the visual image of the container areas that can accept the component change to a cross-hatch pattern.

Tip The mouse pointer changes its shape depending on the container area:

- Where the container allows placement of a component, the pointer changes to a “drop” shape. 
- On areas that cannot accept a component, the pointer changes to a “prohibit” shape. .

The main point to remember when working with different layouts in a container is that Together only allows you to place a component in a location that the current layout manager supports. Those locations show the white cross-hatching pattern, as shown in [Figure 87](#).

Figure 87 Valid placement locations for any layout manager always display the cross-hatching pattern



Other manipulations with the components also depend on the properties of the container layout manager. When a component is selected by a mouse-click, it is marked with bullets at the component's borders. Where the layout manager enables resizing or relocation of the component, the bullets are black. If any visual manipulations with the components are prohibited, the bullets are grey. In case of multiple selection, the first component selected is marked with the white bullets and plays the role of an anchor, that serves as a reference point for the various adjustment operations.

The following sections ("[NullLayout](#)", "[BorderLayout](#)", "[CardLayout](#)", and "[GridBagLayout](#)") provide specific information on the appropriate layout managers.

NullLayout

NullLayout is the most basic layout that enables rather free visual manipulations with components. You can refine the results of your exercises using the special adjustment commands of the right-click menu. Thus, it is possible to align the components against the selected anchor, spread them within a container, and resize them as required.

Components right-click menu contains the following command nodes:

- **Layout**, that allows to align the components (**Align**), allocate them evenly against an anchor (**Space Evenly**), and spread them within a container (**Distribute in Container**).
- **Match Size**, that allows to produce components with uniform dimensions.
- **Fill**, that allows to produce container-size components.

Important All adjustment operations are performed on a selection of components. The element that is selected first, becomes an anchor. This is denoted by the white square bullets, unlike the black bullets for the ordinary selection.

It is vital to right-click on the anchor, to preserve selection. If you right-click on another selected component, it will become an anchor, and the result will be different from what you expect to obtain.

To align components against an anchor:

1. Select the components to be aligned.
2. Right click on the selected anchor to open the right-click menu.
3. On the right-click menu, expand the **Align** command node, and choose one of the available options (Left, Right, Center, Top, Bottom, Middle).

Result: the selection is aligned against the selected anchor. For example, if you choose Align Left, all the components will be aligned against the left edge of the anchor.

To place the components at the equal distances against the anchor:

1. Select the components to be spaced, keeping in mind the anchor.
2. On the right-click menu of the anchor, choose **Layout | Space Evenly** command.

Tip The nested commands are only enabled if three or more components are selected.

3. Choose the required direction (Horizontal or Vertical).

To spread the components in the container:

1. Select the components to be distributed, keeping in mind the anchor.
2. On the right-click menu of the anchor, choose **Layout | Distribute in Container** command.
3. Choose the required direction (Horizontal or Vertical).

To resize components against the anchor:

1. Select the components to be resized, keeping in mind the anchor.

Tip At least two components should be selected.

2. On the right-click menu of the anchor, choose **Match Size** command.
3. Choose the required dimension (Horizontal, Vertical, or both).

Result: the components are resized so that the selected dimension matches that of the anchor.

To resize components to the container size:

1. Select the components to be resized.
2. On the right-click menu of the anchor, choose **Fill** command.

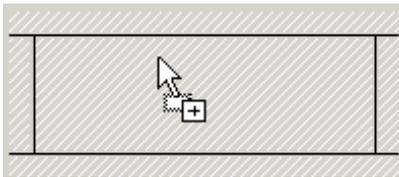
3. Choose the required dimension (Horizontal, Vertical, or both).

Tip NullLayout enables visual resizing of components. As the mouse pointer hovers over one of the black selection bullets, it changes its form to a double arrow. Click on the bullet and drag the component border in the required direction.

BorderLayout

BorderLayout allows only five components to be added to a container in the North, South, East, West, and Center regions. 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. As soon as a component is added to a region, the number of allowed regions is reduced. [Figure 88](#) shows the BorderLayout container with its regions allowed for allocating components.

Figure 88 Empty BorderLayout container: North, South, East, West, Center regions



Example

[Figure 89](#) shows an example of a JFrame using BorderLayout. You can clearly see the North, South, East, West, and Center regions of the container.

Figure 89 JFrame using BorderLayout shows where components can be placed

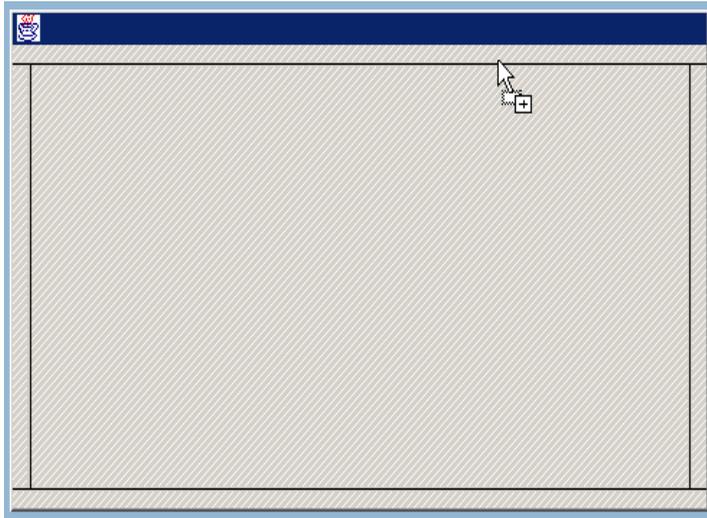


Figure 90 shows the same JFrame as Figure 89, with a button being placed into the JPanel in the South region of the frame. As you can see, only the pane displays the cross-hatching. If the pointer were moved over the JFrame, cross-hatching would show for the other regions of the frame, but the JPanel would not, as in Figure 91.

Figure 90 Placing a second JButton into a JPanel using FlowLayout

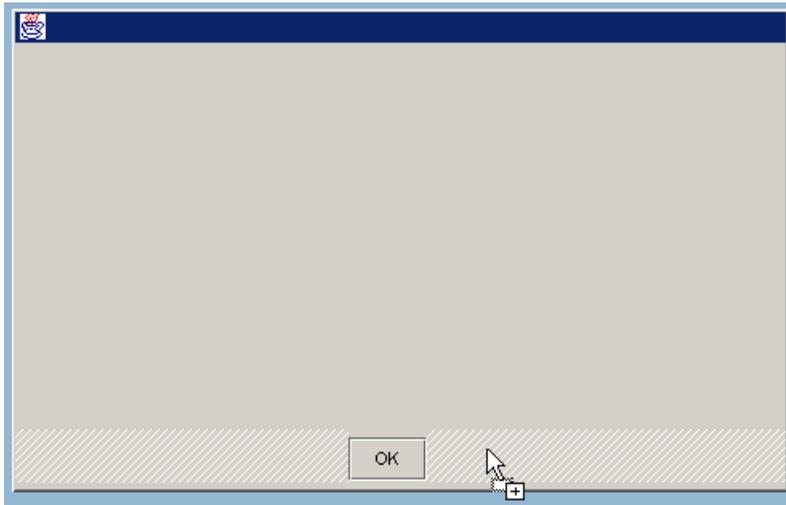
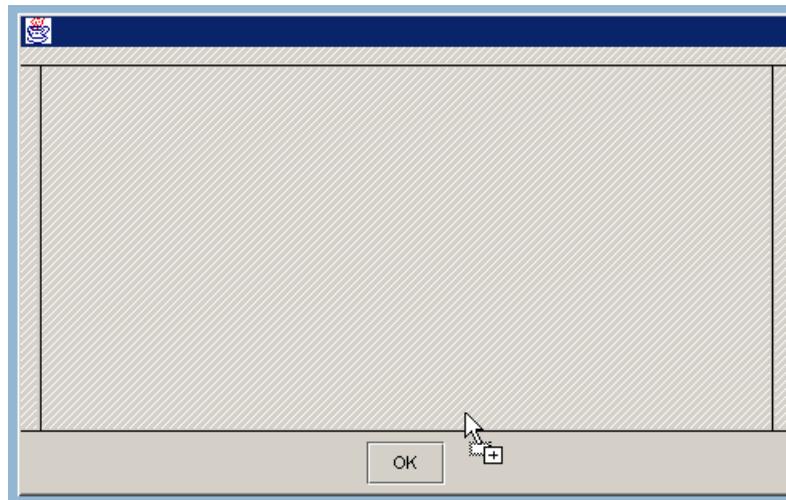


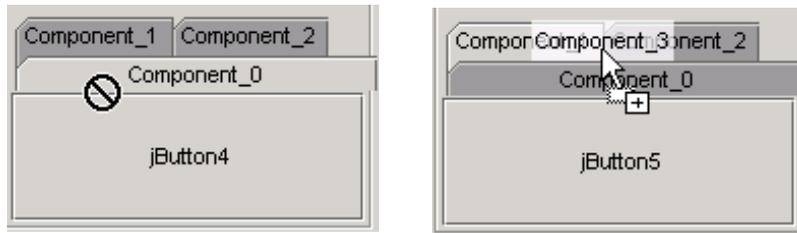
Figure 91 Pointer hovering outside the JPanel



CardLayout

When adding a new component to a CardLayout container, keep in mind that you need to place the mouse pointer between the tabs of the already existing components. Figure 92 below demonstrates adding a new component between the existing cards.

Figure 92 Placing components in a CardLayout container



You can mix components of different types (Swing and AWT) in a single container. When mixing components in a CardLayout container, the AWT components are always displayed on top. You can navigate through the components using the treeview in the UI Builder (Explorer).

GridBagLayout

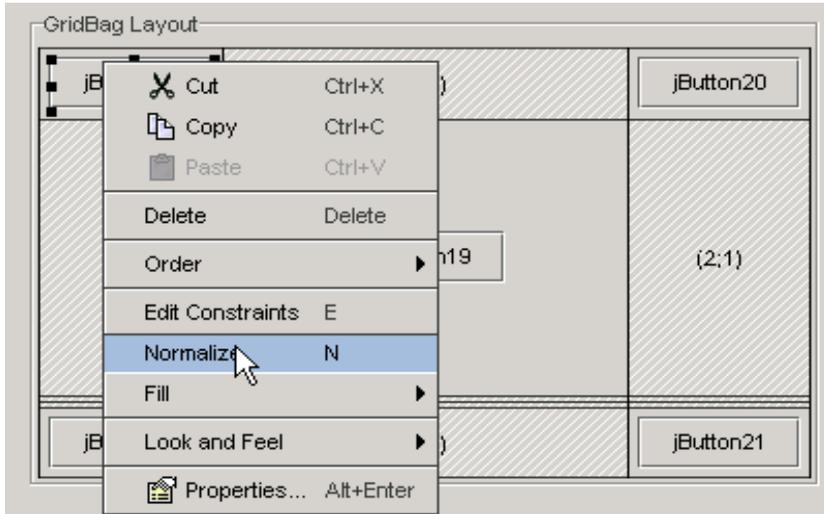
When arranging components in a GridBagLayout container, you may place them in any allowed location, which automatically creates a new cell for each component. However, if a component is deleted, its cell remains in place, and sooner or later you might end up with a number of empty cells. To remove unnecessary empty cells, use the normalization feature. Normalization simplifies constraints of all the components within a container, without actually changing its layout.

To normalize a GridBagLayout container:

1. Right-click a component in a GridBagLayout container.
2. Choose **Normalize** on the right-click menu.

This removes all the unnecessary empty rows and columns, and the spaces between occupied rows and columns, as shown in [Figure 93](#).

Figure 93 Normalizing the GridBagLayout container



When placing components into a `GridBagLayout` container, you have to customize their constraints.

This is done using the Edit Constraints dialog, which is opened from the right-click menu of a component, as shown on [Figure 93](#) above. [Figure 94](#) displays the Edit Constraints dialog window. The controls of this dialog correspond to the variables of the `GridBagLayout` class.

Use the spinners to increase or reduce the values, and the x character to reset the values to the default.

The Position section within the **Display Area** tab allows you to specify the anchor, insets, fill and padding of a component in its display area.

The **Display Area Location** tab enables you to customize the properties of the display area within a container. Using this tab, you can customize the resizing behavior of the component using the *weight* fields, and the gaps between components and the container border. These gaps are regulated by the *width* and *height* fields. If the value *REMAINDER* is selected in these fields, the appropriate areas around the component become prohibited for the allocation of new components. Such areas are marked with bright yellow hatch marks, as shown in [Figure 95](#).

For more details, refer to the relevant topic of the online help system, and to <http://java.sun.com/j2se/1.4/docs/api/java/awt/GridBagLayout.html>.

Figure 94 Edit Constraints dialog

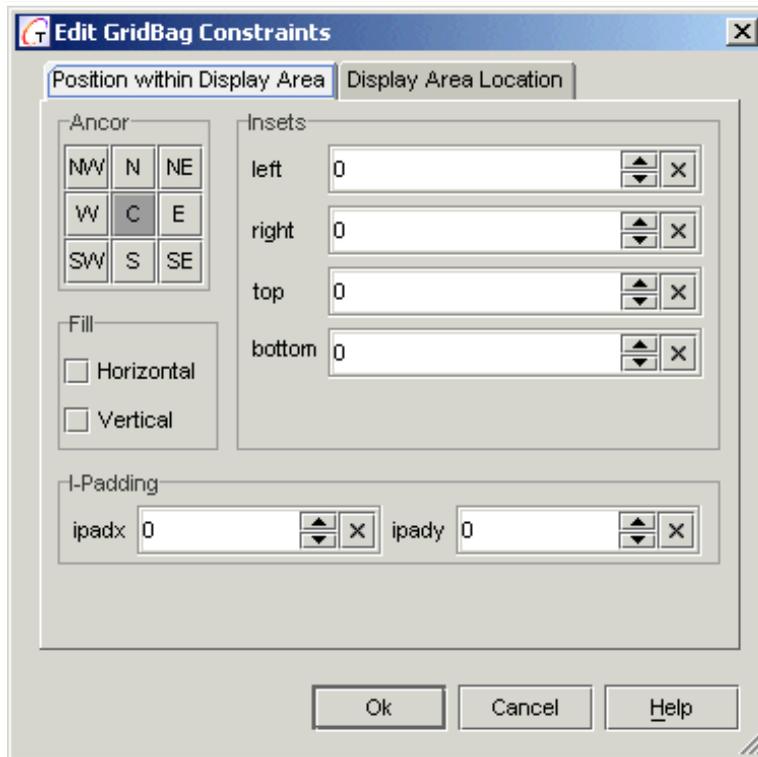
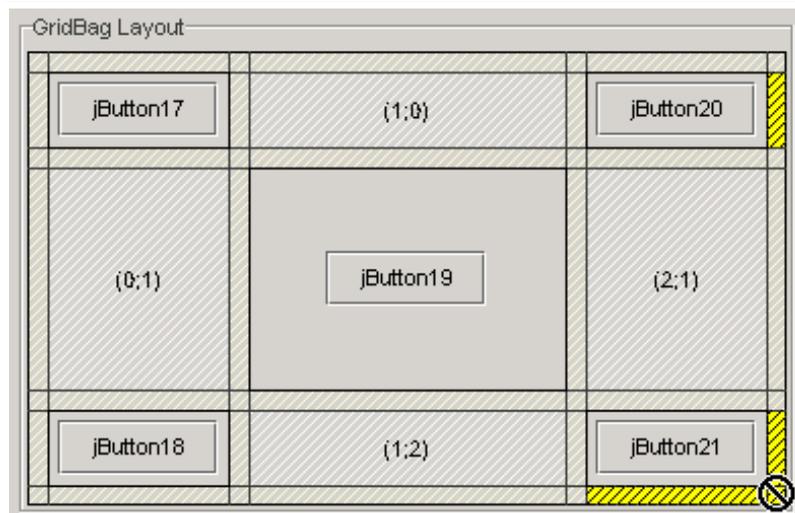


Figure 95 Remainder areas in a GridBagLayout container



Where to learn more about layout managers

There is a great deal of information published on the Web about basic as well as advanced techniques for using layout managers in Java. One useful tutorial is available (at the time of publication) on the Sun 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 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

To change the look and feel of frames, choose the **Look and Feel** command on the right-click menu of the container in the UI Designer. The look and feel options are:

- Metal (the default)
- CDE/Motif
- Windows

Important You can change the look and feel only to the one supported on the given platform. For example, on the Unix platform, the Windows L&F is not supported, and the appropriate menu command is not available.

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 the UI Designer view or the UI Component Explorer.
2. Open 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 field.
4. Press **Enter** to create the event handler block in the source code. The Editor scrolls to the newly inserted event handler line.

Changing alignment, spacing, and distribution

You can change the alignment, spacing, and distribution of components in a frame container with Null layout, using the **Layout** command on the right-click menu of the frame.

Tip To open 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 provides the following commands:

- **Align.** Opens a submenu enabling you to specify horizontal alignment values of Left, Center, and Right, and vertical alignment values of Top, Middle, and Bottom.
- **Space Evenly.** Spaces components evenly either horizontally or vertically.
- **Distribute in Container.** Evenly distributes components over the entire container either horizontally or vertically.

Changing the order of components

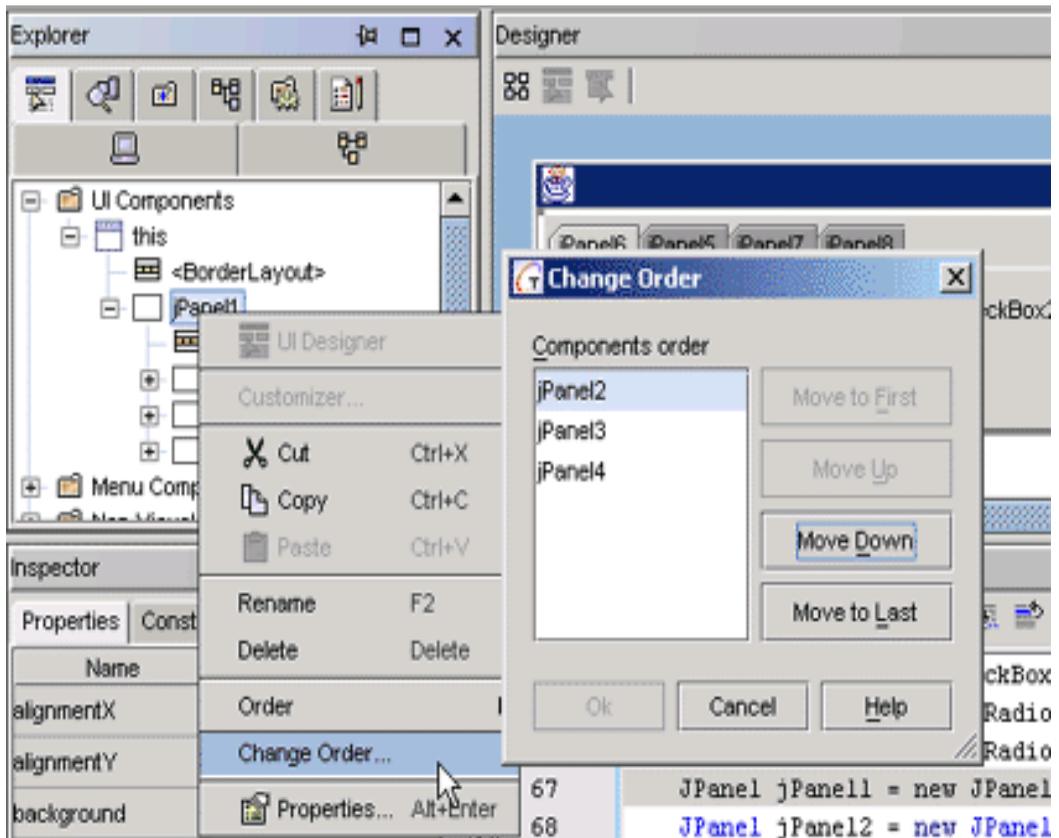
To change the appearance order of components or containers, choose the **Order** command on the right-click menu of a selected component or container. You can make the selected component first or last in its container, or move it backward and forward relative to the adjacent components.

To rearrange all the components within the selected container in one session, use the **Change Order** command on the right-click menu of a container in the **UI Builder** tab of the Explorer (see [Figure 96](#)).

To change the order of components in a container:

1. Select the container whose components should be rearranged.
2. Right-click on this component in the **UI Builder** tab of the Explorer, and choose the **Change Order** command.
3. In the Change Order dialog, choose one of the displayed components.
4. Using the direction buttons, move the component to the desired location.
5. Select the other components and repeat step 4 as required until you end up with the proper arrangement of components.
6. Click **OK** to confirm the changes and close the dialog.

Figure 96 Reordering components in a container using the Change Order dialog



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 is activated, the Designer pane has a special Menu Designer view for visually developing menu components.

The “Designing menus” section covers the following topics:

- Creating a new main menu (see [“Creating a new main menu” on page 330](#)).
- Creating a new popup menu (see [“Creating a new popup menu component” on page 335](#)).
- Defining menus visually (see [“Defining menus visually” on page 336](#)), which includes:
 - Defining menu items including checkbox and radio button items (see [“Defining new menu items” on page 337](#)).
 - Creating nested menus (see [“Creating nested menus” on page 341](#)).
 - Enabling, disabling, hiding, and showing menu items (see [“Disabling and hiding a menu using the right-click menu” on page 342](#)).
 - Defining menu item properties such as icons, mnemonics, and keyboard shortcuts (see [“Adding icons to menus” on page 343](#)), and more.
- Alternative way of defining menu commands (see [“Defining menu items using the UI Component Explorer” on page 345](#))

Important The whole procedure of designing menus involves the following major independent steps:

- creating a container
- creating a menu
- setting menu component in the container frame.

We’ll consider this procedure step by step for a main menu.

Creating a new main menu

To design a main menu:

1. Using the **Object Gallery**, create a container.
2. Create the menu bar component (see [“Creating a new menu bar component” on page 331](#)).
3. Add commands to the menu (see [“Adding commands to the menu” on page 331](#)).

4. Add the created menu to the container (see [“Setting a menu bar into the container frame” on page 332](#)), and set the required properties.

Find the detailed descriptions of specific steps in the following sections.

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. On the Diagram toolbar, click the **UI Designer** button to switch to the UI Designer view (see [“UI Designer view” on page 310](#)).
2. In the UI Designer view in the Designer pane, open the container frame that will contain the menu.
3. In the Toolbox, open the *AWT Menu* or *Swing Menu* page, depending on the type of components you are using.
4. Click **MenuBar** (AWT) or **JMenuBar** (Swing) button on the Toolbox.
5. Move the pointer over the Designer pane and click anywhere on the background of the Designer pane. (You cannot click inside the frame component.)
6. In the UI Component Explorer, expand the *Menu Components* node and choose the new menu component (*MenuBar1* or *JMenuBar1*).
7. Optionally change the name of the menu component in the Explorer (for more information, see [“Renaming menu component identifiers visually” on page 344](#)).

Adding commands to the menu

Now you can populate the menu component with commands. This is how it's done.

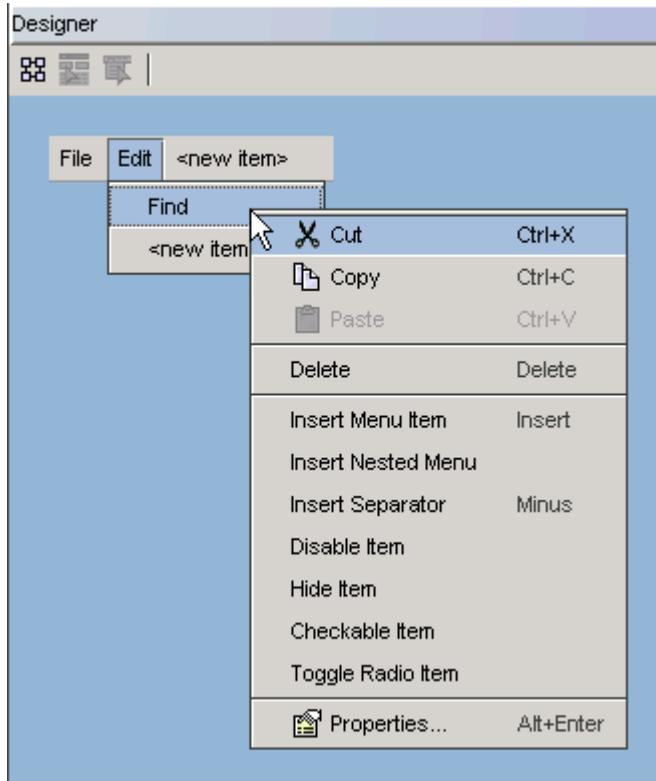
To add commands to the menu:

1. On the Diagram toolbar, click the **Menu Designer** button to switch to the Menu Designer view. You will see the placeholder for the new menu item.
2. On the right-click menu of the new item, choose **Insert Menu** command. This will create a new command on the menu, add a `<new item>` entry on the menu bar, which allows to further extend the menu bar, and another `<new item>` entry under the command, to further extend the command sub-menu. (see Figure 88, [“Adding commands to the menu bar” on page 332](#))
3. Repeat the step 2 as required.

Note Alternatively, you can just double click on a `<new item>`, and type in the command name. This action will both visually rename the command, and automatically add a new entry to the menu.

Using the right-click menu of the menu component, you can create nested menus, insert or remove separators between the groups of commands, disable / enable or show / hide the selected items. It is also possible to create checkable items and radio buttons. These features are described further in the sections under [“Defining menus visually”](#) on page 336.

Figure 97 Adding commands to the menu bar



Finally, the menu is ready. Switch to the UI Designer view - and you see no visible results! The container is still empty... But don't be frustrated - with the next step you will put the menu to its container.

Setting a menu bar into the container frame

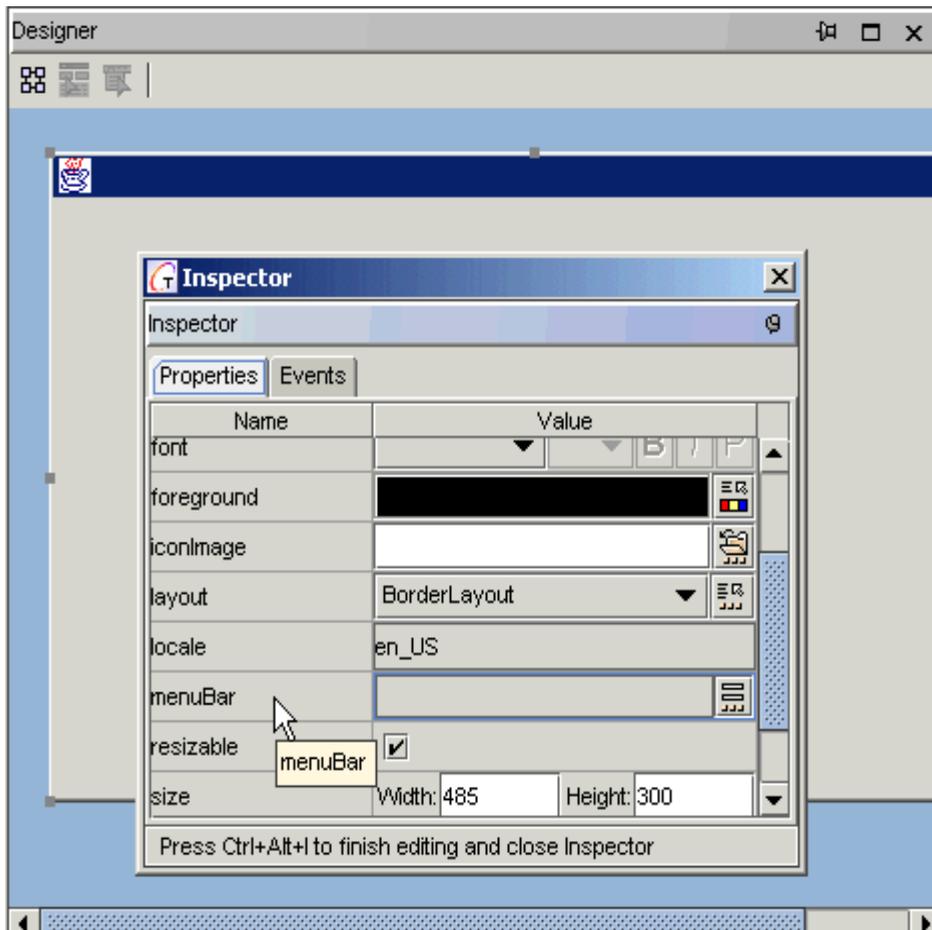
A frame component does not display a menu bar until an appropriate `set` statement is placed into the source code (see [“Understanding the code”](#) on page 334). There is no way to visually place the menu to a container. To do so, you have

to use the Inspector, which enables you to set a menu bar component into a container frame.

To set a menu to a container:

1. Switch back to the UI Designer view
2. On the right-click menu of the container, choose **Properties** command, to open the Inspector.
3. In the Properties tab of the inspector, choose the *menuBar* field. The *menuBar* property specifies which menu bar component should be set in the frame.

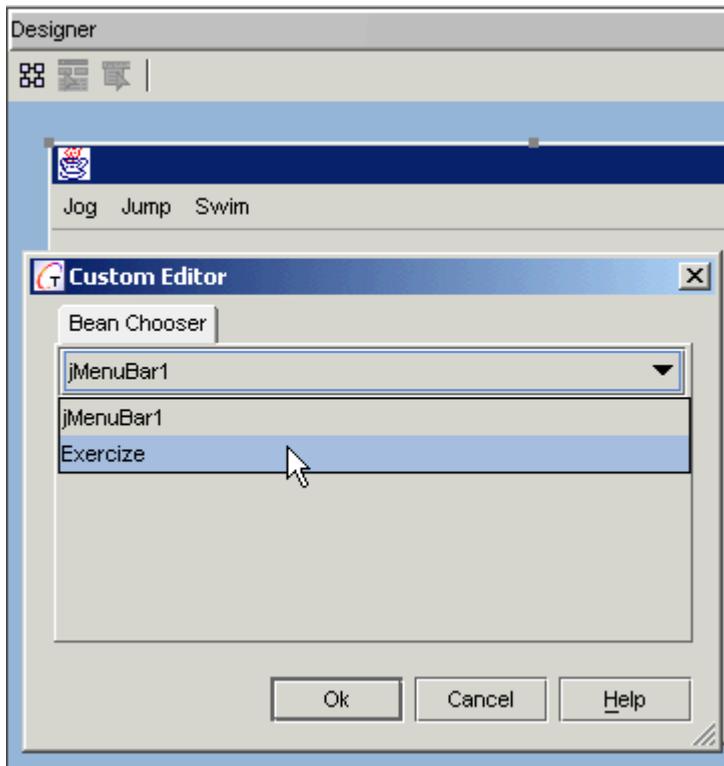
Figure 98 *menuBar* property of the container



4. In the *menuBar* field, click on the Editor button, to open the editor for this property.
5. In the Custom Editor dialog, click on the drop down list, and choose the menu that will be actually added to the container.

Note The drop-down list of the Custom Editor dialog contains 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 menu bar appears in the visual representation of the frame in the UI Designer view, as shown on the [Figure 99, “Using the property editor to set a menu to a container frame,” on page 334](#). See also the section “[Understanding the code” on page 334](#) to learn how the visual operations are reflected in the source code.

Figure 99 Using the property editor to set a menu to a container frame



Tip View an example of how a menu bar is set in the main `JFrame` class created by the *Application* pattern in the Object Gallery (**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 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 choose 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 “attach” a popup menu.

Example:

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 the component that the popup menu was called on, and *jPopupMenu1* is the name of a popup menu component for *commentField*).

Also add a mouse listener using the Inspector (for instructions, see [“Creating event handlers for UI components” on page 327](#)). For example:

```
commentField.addMouseListener(  
    new MouseAdapter() {  
        public void mouseReleased(MouseEvent e) {  
commentFieldMouseReleased(e); }  
    });
```

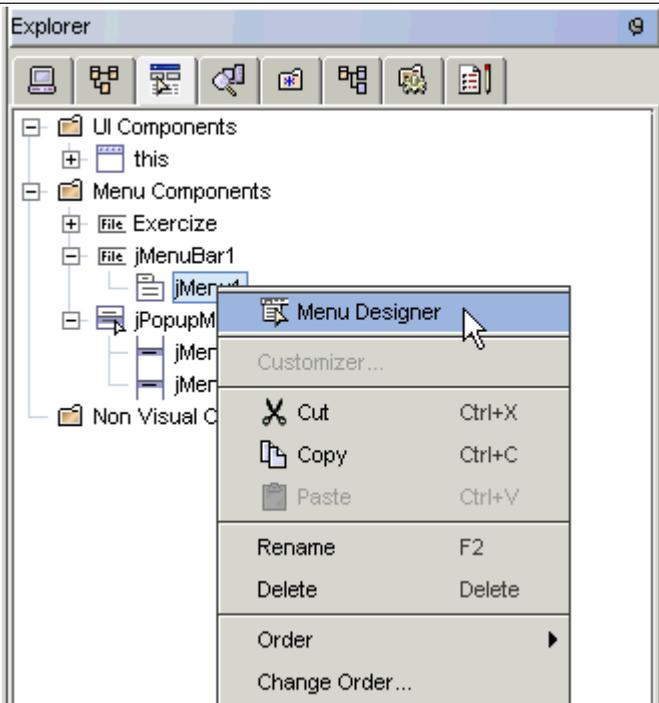
Note The example above refers to the PC only, because on the different operating systems the popup menus are invoked in different ways. For example, in MacOS popup menus are invoked on CTRL+mouse press. Thus, the better way to display a popup menu is to catch both `mouseReleased()` and `mousePressed()` events and test them by means of `isPopupTrigger()` method. The subtle issue here is that both events should invoke the same method:

```
jButton.addMouseListener(  
    new MouseAdapter() {  
        public void mousePressed(MouseEvent e) {  
jButtonshowPopup(e); }  
        public void mouseReleased(MouseEvent e) {  
jButtonshowPopup(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 are listed in the UI Component Explorer, as shown in [Figure 100](#). The *Menu Components* node of the Explorer lists all the menu and menu item components.

Figure 100 The UI Component Explorer and its right-click menu



To activate the Menu Designer view:

1. In the Explorer, choose any menu, popup menu, or menu item component
2. Choose **Menu Designer** command on the right-click menu, 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](#)” on page 339 and “[Defining a radio-button menu item](#)” on page 340.

Menu definition syntax and syntax helpers

You can define a number of properties of new menu items in the in-place edit field for the items, 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 example, 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* are still displayed 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.

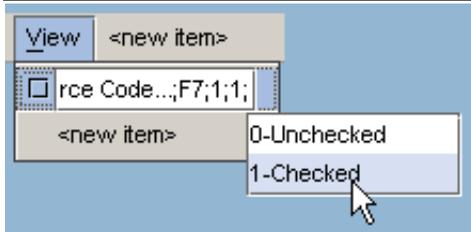
To use the syntax helpers:

1. Type in the menu text followed by a semi-colon. A context-sensitive pop-up helper menu appears after a short pause.
2. Choose 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 101 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.

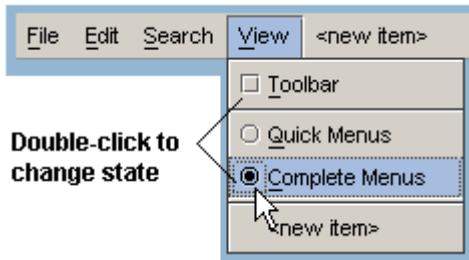
Tip To revert to a regular menu item, right-click on the item in the designer and choose **Non-checkable item**.

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 the *checked* state. For more information, see [“Menu definition syntax and syntax helpers” on page 338](#).

Once a checkbox menu item has been created, change the checked state of the checkbox by double-clicking on the checkbox itself, as shown in [Figure 102](#).

Figure 102 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 the 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 the *checked* state. For more information, see [“Menu definition syntax and syntax helpers” on page 338](#).

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, as shown in [Figure 102](#).

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

You can 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 (-). Press **Enter**.

Creating separators using the right-click menu of the menu item

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.

Creating separators using the Toolbox

Select a separator component in the AWT Menu or Swing Menu page of the toolbox and drop it into the menu at the desired location. See [“Defining menu items using the UI Component Explorer” on page 345](#) 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:

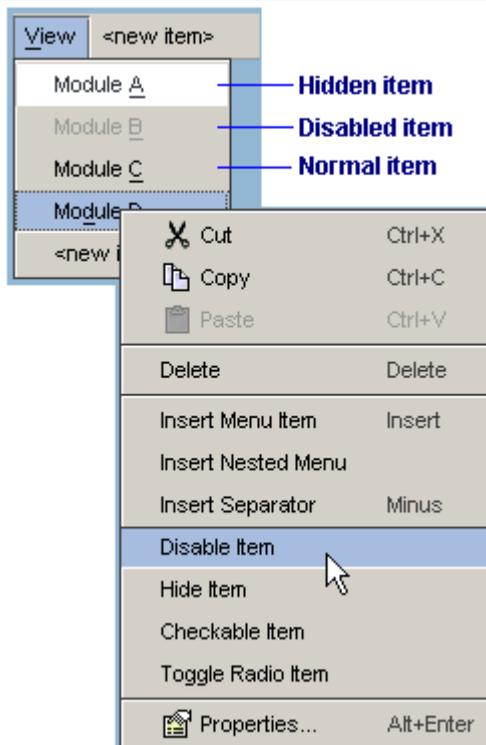
1. Select the item in the Menu Designer view (Designer pane).
2. Press **Alt+Enter** or click the Properties button on the main toolbar to open the Inspector.

Tip You might find it convenient to undock and resize the Inspector.

Disabling and hiding a menu using the right-click menu

As mentioned in [“Menu definition syntax and syntax helpers” on page 338](#), you can use a special syntax in the in-place editing field to enable/disable or show/hide menu items at design time.

Figure 103 Setting design time properties for menu items using the right-click menu



You can modify the settings in the same way, but you might 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, as shown in [Figure 103](#).

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, as shown in [Figure 103](#) above.

Adding icons to menus

Menu items can 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 the item when the item is not in the selected state. Note that the *rolloverEnabled* property must be checked (*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* and *selected* properties must be checked (*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 (*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 properties of the item in the Inspector (to open the Inspector, press `Alt+Enter` or click the Properties button on the 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 of the other iconic properties as required to get the effect you desire.

Tip According to Sun user interface guidelines, the standard size for menu icons is 16x16 pixels.

Defining menu colors and font

It is possible to 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.

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, [“Menu definition syntax and syntax helpers” on page 338](#).

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. To accomplish this task, use the **Search | Replace** command on the main menu when the Editor pane has focus.

Defining menu items using the UI Component Explorer

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 using the Toolbox:

1. Expand the *Menu Components* node of the **UI Builder** tab of the Explorer (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 (such as Swing Menu).
3. In the Toolbox, click 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 the node of the parent item to create the new menu item with a default identifier.

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 30, “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, and so on. Detailed descriptions of each individual audit are provided in the Java Audits 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 308](#).
2. Make sure the UI Builder feature is activated as described in this chapter under [“Activating the UI Builder” on page 305](#).
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 audits you want to run against your project and click **Start**.

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 metrics:

1. Make sure your project includes Java UI classes as defined in this chapter under [“Creating UI classes” on page 308](#).
2. Make sure the UI Builder feature is activated as described in this chapter under [“Activating the UI Builder” on page 305](#).
3. On the main menu, choose **Tools | Metrics** to open the Java Metrics dialog.
4. Locate the *UI* category in the list of metric titles, and expand the category node to see all available metrics.
5. Check the metrics you want to run against your project and click **Start**.

Generating user interface documentation

The Together documentation generation feature provides a special documentation template that you can use to generate documentation for the user interface of a project. Generate documentation in HTML, RTF, or text format using the standard template, modify the standard template, or design your own template.

For more detailed information on documentation generation features, see [Chapter 22, “Generating Documentation for Together Projects”](#).

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 opens.
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_HOME\modules\com \togethersoftware\modules\gendoc\templates\UIBuilderReport.tpl` (where `TG_HOME` 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 simply JavaBeans™. The toolbox comes with a default set of standard AWT and SWING bean components (for a full listing, search Online Help for *Toolbox*). The bean components are divided into named categories (such as *Swing General*, and *AWT Menu*). 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 categories and the components each one contains.
- Reorder existing categories and add new categories.
- 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, but it is not required. Add specific beans from any JAR file to any Toolbox category. The JAR files containing custom bean components can reside on any drive accessible to your system.

Tip Consider adding a *Custom* directory to `TG_HOME/lib` and placing your custom component archives there.

Once you have your component archives 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 307 and [“Creating UI classes”](#) on page 308.

2. Activate the UI Designer view by clicking the UI Designer button on the Designer pane toolbar (for more information, see [“UI Designer view” on page 310](#)).
3. To display the Toolbox, choose **View | Main Panes | Toolbox** on the main menu. Alternatively, click the Toolbox button on the main toolbar.
4. Right-click 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 items are in the desired position.

Creating new component categories

To create a new component category:

1. Click the **New Category** button on the toolbar immediately above the *Categories* list.
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. Click the **Remove** button on the toolbar immediately above the *Categories* list.

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. However, the user-defined categories will stay intact.

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. 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 the Advanced mode of the dialog to add these archives.
5. Click **OK** to close the Add Beans dialog and add the selected JAR file 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. Click the **Remove** button on the toolbar immediately above the *Categories* list.

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 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, choose *Select icons*.
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.

Java features supported in designable classes

Normally, you can design a correct user interface using a basic set of operations. The UI Builder supports Java grammar and lexical structure to the extent they are supported by the Together SCI, and provides partial support for a rather extended subset of features defined by the Java Language Specification, second edition (http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html).

The following tables summarize the supported Java Language Specification (JLS) features:

Table 37 Supported JLS features

JLS feature	Not supported	Partly supported	Supported	Note
Java Syntax			X	
Keywords		X		Table 38 on page 351

Table 37 Supported JLS features

JLS feature	Not supported	Partly supported	Supported	Note
Literals			X	
Separators			X	
Operators		X		Table 39 on page 352
Integer operations		X		Table 40 on page 352
Types, values and variables		X		
Conversions and promotions		X		Identity Conversion, Widening Primitive Conversion, Narrowing Primitive Conversion, Widening Reference Conversion, Narrowing Reference Conversion, String Conversion
Names		X		Array members are not supported
Classes		X		Classes with <code>abstract</code> , <code>final</code> , <code>strictfp</code> modifiers are not supported
Arrays	X			
Exceptions	X			
Execution		X		
Blocks and Statements		X		
Threads and Locks	X			

Table 38 Supported Keywords

Supported Keyword	Note
<code>boolean</code>	May be used as a type of a variable
<code>byte</code>	May be used as a type of a variable
<code>char</code>	May be used as a type of a variable
<code>class</code>	May be used with some limitations
<code>double</code>	May be used as a type of a variable

Table 38 Supported Keywords

Supported Keyword	Note
float	May be used as a type of a variable
int	May be used as a type of a variable
long	May be used as a type of a variable
new	
private	May be used as a modifier
protected	May be used as a modifier
public	May be used as a modifier
return	
short	May be used as a type of a variable
static	Variables defined in a class body with the <code>static</code> modifier (class variables) are processed prior to other variables (instance variables)
super	
this	

Table 39 Supported Operators

Operator	Note
=	
!	For boolean type only
? :	The type of conditional expression is not determined
==	
!=	
&&,	For boolean type only
++, --, +, -, *, /, &, , ^, %	
+=, -=, *=, /=, %=	Not supported for arrays and array elements

Note As of this writing, the left shift <<, signed right shift >>, unsigned right shift >>> and bitwise complement operators ~ are not supported.

Table 40 Supported integer operations

Supported integer operation	Note
Numerical comparison ==, !=	Returns boolean value
Unary +, -	Returns int or long
Multiplicative *, /, %	Returns int or long

Table 40 Supported integer operations (continued)

Supported integer operation	Note
Increment prefix and postfix (++)	Returns int or long
Decrement prefix and postfix (--)	Returns int or long
Integer bitwise operations &, , ^	
Conditional operator ?:	
Cast operator	Converts from an integral value to a value of any specified numeric type
String concatenation +	

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 355](#)
- [“Using Together with a Version Control System” on page 356](#)
- [“Product-specific VCS notes” on page 366](#)
- [“Using Together with ClearCase” on page 371.](#)

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
- **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 3, "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 104](#).)

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.

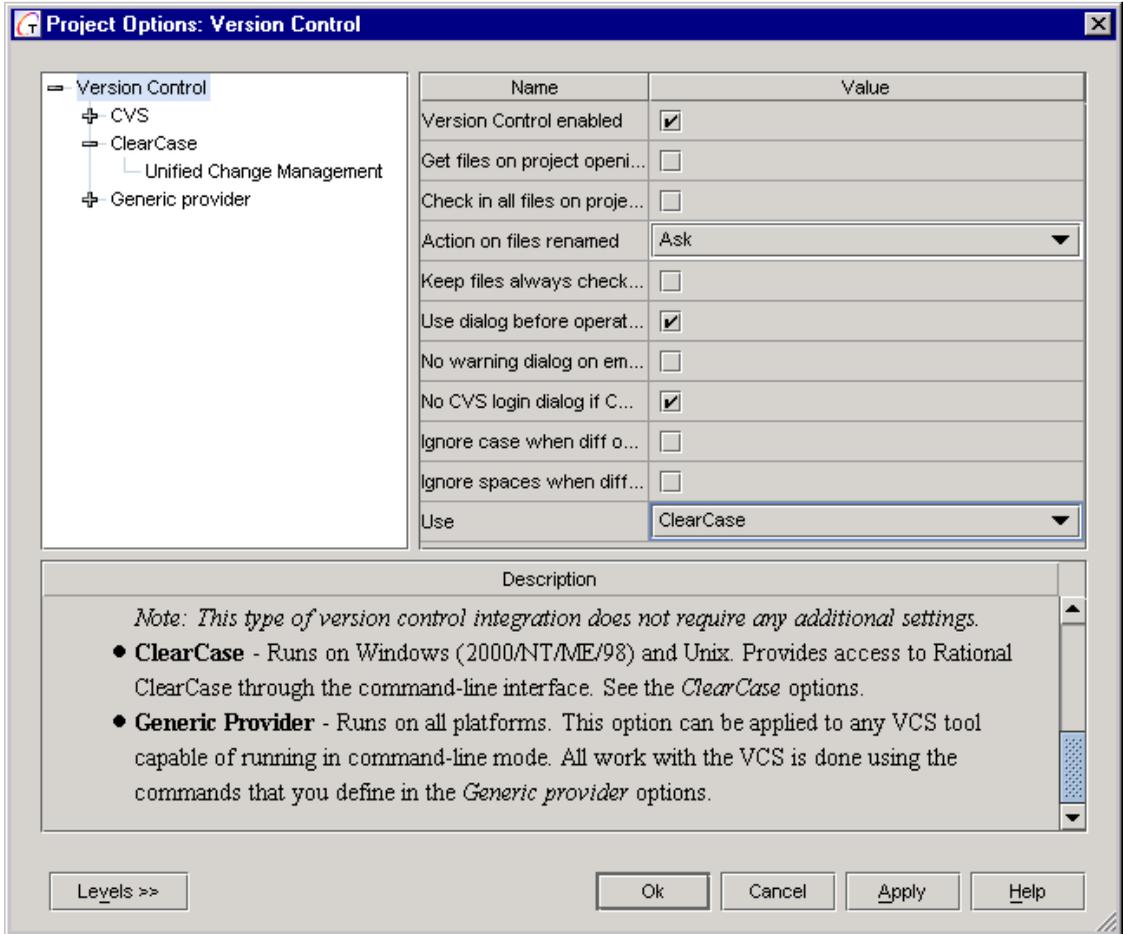
To choose a version control system:

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. 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. Provides access to Rational ClearCase through the command-line interface.

Figure 104 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 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.

Important 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 5, “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 41 Together files under version control

Files	Description	Location
*.tpr	Together project file	Project primary root folder
.df	UML diagram files. Note: Do not version df.package files. Together automatically creates and updates these files.	Dependent upon where created.
*.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, or just click Ctrl+Q, 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 367](#).

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 42 File formats and MIME types

Name	Format	MIME type
Java	Ascii text	text/plain

Table 42 File formats and MIME types (continued)

Name	Format	MIME type
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 43 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
%.dfBusinessProcesses	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
%.tws	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 Continuus 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.

- Continuus 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 Continuus 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.
- Project path should be specified correctly, when a project is put under version control. If a non-existent path is specified for a valid project, a new folder with the specified pathname is created in the StarTeam repository.

Using Together with ClearCase

ClearCase is one of the many options for version control systems (VCS) within Together. 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:

- 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 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 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 357](#) 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 check will be attempted.
- **Show version ids:** Turns on the version id capability in the System pane (see [Figure 105](#)). 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. This option is often found unchecked when ClearQuest is used to create and track new activities.

Basic Version Control

Since Together provides a framework for version control, most of the basic operations (such as add, checkout, and so on) behave the same way in Together. 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.

ClearCase Move

When moving a class or package node that is under version control, you must use the **ClearCase Move** command in order to maintain version control for the files.

To move a class or package node, right-click on the node and choose **ClearCase Move**. If the selected node *is not* under version control, the Choose Destination Folder dialog appears for you to specify the destination of the node. If the selected node *is* under version control, a Rational ClearCase Integration dialog appears asking you to confirm the move before continuing.

Important **Do not** drag and drop versioned classes or packages as this does not trigger a ClearCase Move and ClearCase does not maintain history for dragging and dropping. If, by accident, you do drag and drop, you can use **Undo Move** (CTRL+Z) from the Edit menu. Note that **Undo** applies only to local file system changes and **not** to actions within 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 372](#)) 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.

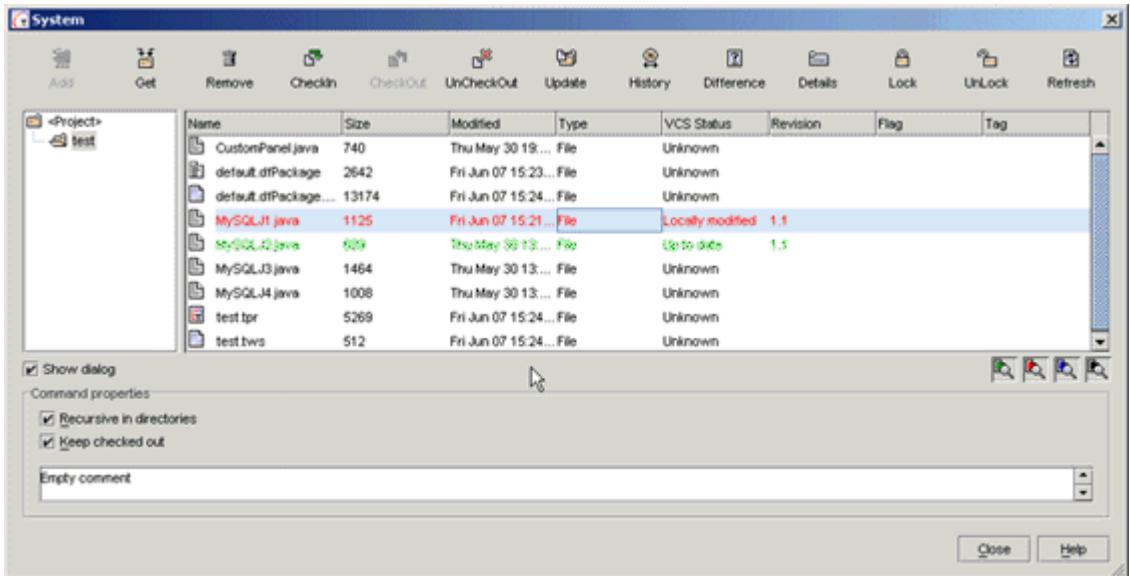
Checkout

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.

System

The system dialog brings up the status of the entire project. (See [Figure 105](#).) 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 105 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. None the less, they are equally important to the interactions between Together 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.

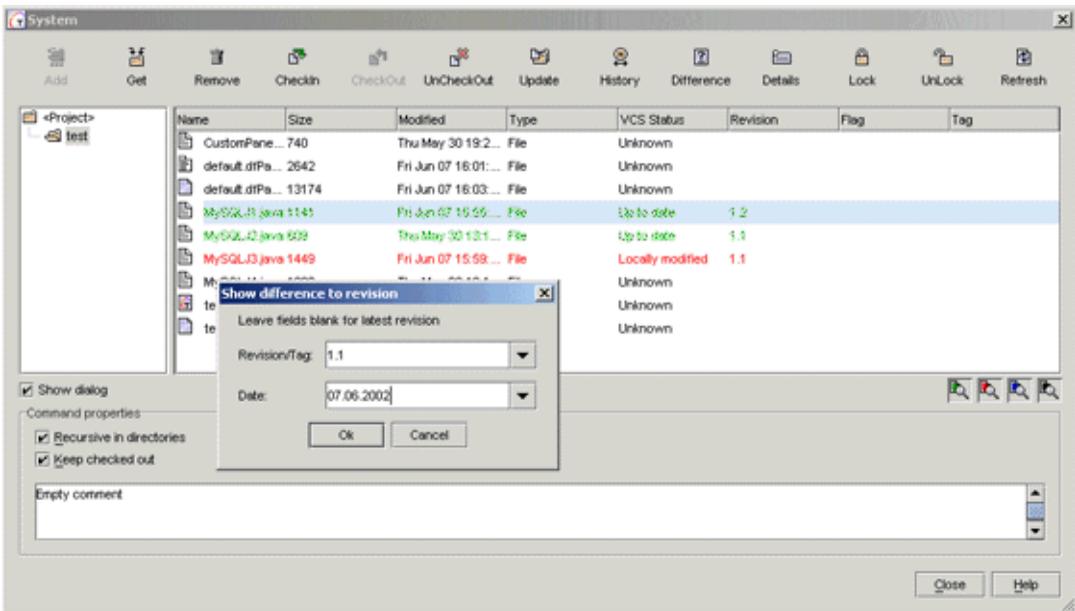
Remove

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 106](#).

Figure 106 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 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 message pertaining to an action that could not be taken.

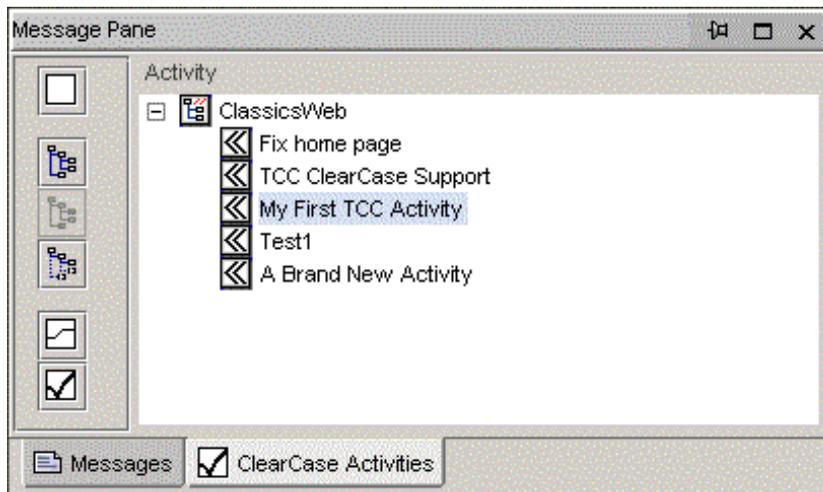
Unified Change Management

Together supports the creation and management of activities according to the principles of Unified Change Management. Under Together, 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 107](#).)

Figure 107 Activity management in Together



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 371](#)), 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 107](#)). 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 Together (see [“Setting up the interface” on page 371](#)). 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 fourth 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 Together 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.

5

GENERATING DOCUMENTATION

- Chapter 22, “Generating Documentation for Together Projects”
- Chapter 23, “Designing Custom Documentation Templates”
- Chapter 24, “Documentation Template Controls”
- Chapter 25, “Creating Multi-Frame HTML Documentation”
- Chapter 26, “Documentation Generator and Template Designer Reference”

Generating Documentation for Together Projects

Together ControlCenter can create external documentation for Together projects. This chapter describes how to use the documentation generator in Together 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 381.](#)
- [“Automating documentation generation” on page 390.](#)

Generating documentation for an open project

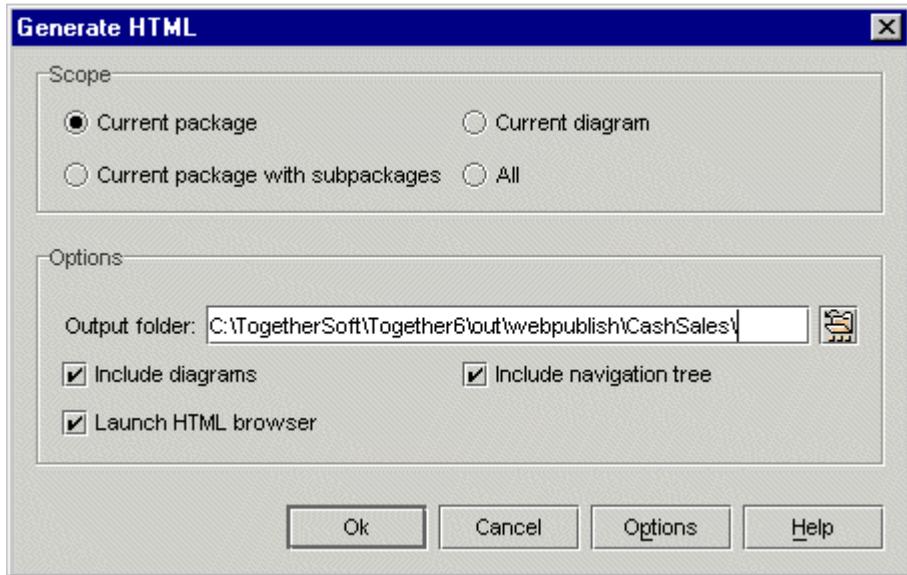
You can start the documentation generator from an open Together project. Simply select one of the following three commands from the main menu in Together:

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 a 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.

Generating HTML documentation

The Generate HTML command generates HTML documentation using a documentation template that Together provides. [Figure 108](#) shows the Generate HTML dialog box.

Figure 108 Generate HTML dialog box



Generate HTML documentation generation parameters

The dialog box for generating HTML documentation allows 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.

The lower section of the dialog box has options to specify the destination and optional actions.

- *Output folder*: Enter the location for the generated output, or select from the file chooser. The default location for HTML documentation is `TGH/out/webpublish/$PROJECT_NAME$`.

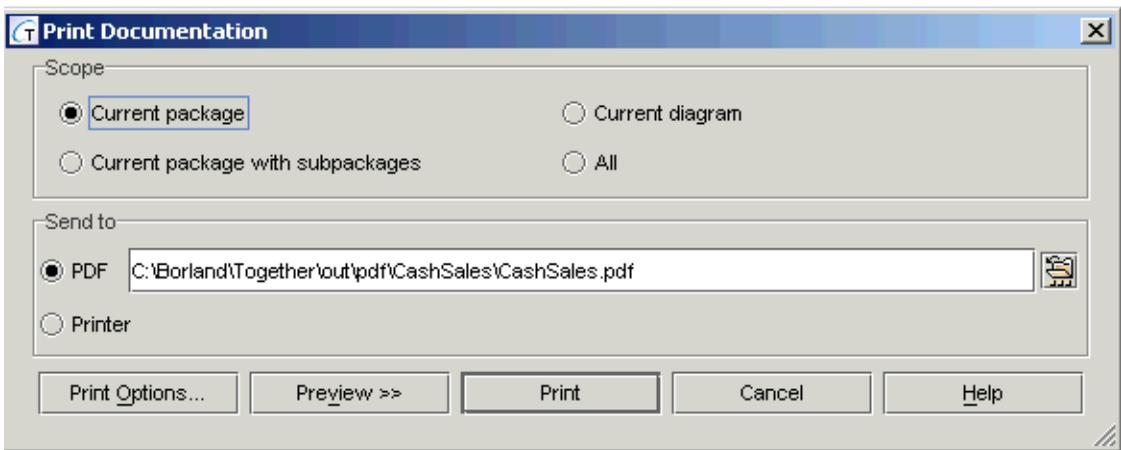
Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

- Checkboxes:
 - *Include diagrams*: Check to include diagram images in the output .
 - *Include navigation tree*: Check to include a navigation tree in the output.
 - *Launch HTML browser*: Check to load the documentation into the default web browser of the operating system. This starts the application if necessary.
- *Options button*: Click the **Options** button to refine additional parameters for generating HTML documentation.

Printing documentation

The Print Documentation command generates PDF documentation or prints directly to a printer using a documentation template that Together provides. [Figure 109](#) shows the Print Documentation dialog box.

Figure 109 Print Documentation dialog box



Print Documentation generation parameters

The dialog box for printing documentation in PDF format or to a printer 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.

The lower section of the dialog allows you to specify the destination for PDF documentation. Specify printer options and display a preview of the documentation prior to printing using the **Print Options** and **Preview** buttons at the bottom of the dialog.

- *PDF*: Enter the location for the generated PDF file, or select from the file chooser. The default location for PDF documentation is `$(TGH)/out/pdf/$(PROJECT_NAME)`.

Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

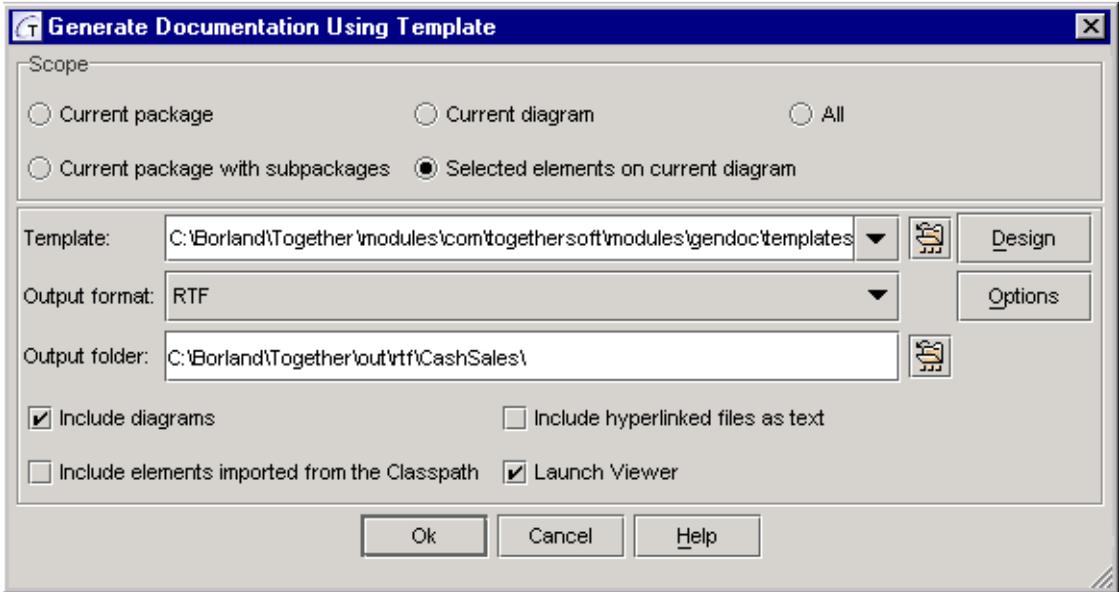
- *Printer*: Select to print documentation to a printer.
- **Buttons**:
 - *Print Options*: Click the **Print Options** button to refine printer parameters.
 - *Preview*: Click the **Preview** button to preview the documentation.

Generating documentation using a template

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 Generate Using Template command generates HTML, RTF, or text documentation using a custom documentation template or a template that ships with Together. [Figure 110](#) shows the Document Generation Using Template dialog box.

Figure 110 Document Generation dialog box



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.
- *Selected elements on current diagram*: Generated output only for the elements that are selected in the current diagram.
- *All*: Generated output covers the entire project.

The lower section of the dialog box has options to specify the type of template from which the documentation is to be generated, output format and destination, and optional actions.

- *Template*: Specify the template for the documentation generator to use. 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 23](#), “[Designing Custom Documentation Templates](#).”)
- *Output format*: Select HTML, RTF, or TXT 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 `$(TGHS)/out`. The subfolder corresponds to the *Output format* selected. For example, if you choose HTML as the *Output format*, then the generated output location of the documentation will be `$(TGHS)/out/html/$(PROJECT_NAME)`.

Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

- **Checkboxes**:
 - *Include diagrams*: Check to include diagram images in the output (according to the instructions in the template).
 - *Include elements imported from the classpath*: Check to expand the generated documentation to include the classpath elements.
 - *Include hyperlinked files as text*: Leave this box unchecked to embed all hyperlinks to external documentation as URLs. Check the box to embed the hyperlinked documentation in the generated document. With RTF documentation, the embedded documentation is as follows:


```
INCLUDE TEXT:<URL>
```

 To update the field, select it and press F9 or press Ctrl-A, F9 to update all fields. Mac OS X users should press Open Apple-A, F9.
 - *Launch viewer*: Check to load the documentation into the appropriate application as soon as the documentation is generated. This starts the application if necessary.

Note HTML documentation launches in the default web browser of the Operating System. RTF documentation launches in the default RTF viewer of the Operating System. Text documentation launches in the Together Editor.

RTF documentation options

With RTF as the output format, you can click the **Options** button to refine the parameters for generating the documentation. The available options (shown in [Figure 111](#)) are as follows:

- *Formatting template*: Enter 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*: Check to translate HTML tags into appropriately formatted text in the printed documentation. The following tags are supported:

``, `<i>`, `<u>`, `<h1>`, ..., `<h6>`, `<code>`, `<tt>`, ``, ``, `<pre>`, `<p>`, `
`, ``, ``, ``

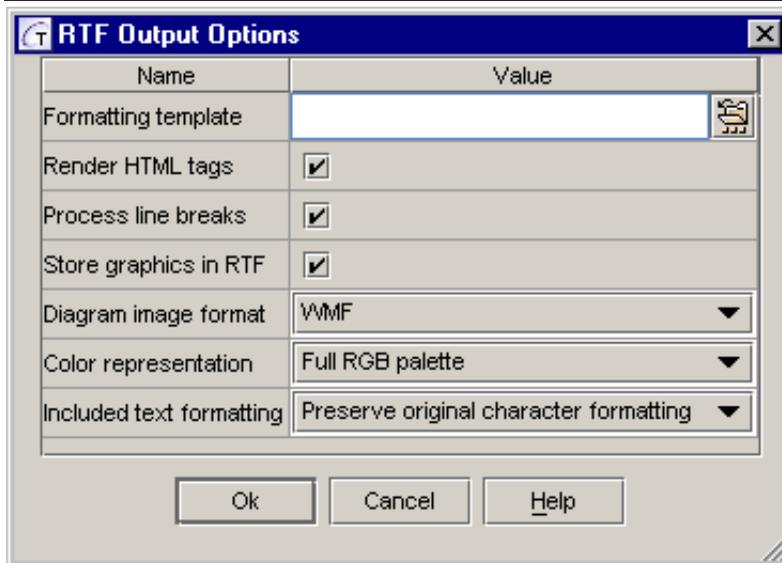
Note Descriptions for model elements may contain HTML tags. This setting is useful for documentation templates that process descriptions.

- *Process line breaks*: Check to preserve line breaks even if the Render HTML tags is checked on.
- *Store graphics in RTF*: Check to embed 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*: Select WMF or GIF format.
- *Color representation*: Select RGB color or 16-bit color.
- *Included text formatting*: Select whether to preserve the original formatting or apply the one from the *Formatting template* that is specified in the first option.

Figure 111 RTF Output format options



Using a custom formatting template (.dot file) with DocGen

To use a custom formatting template (.dot file), the DocGen template must be modified to contain the same set of styles that are defined in the .dot file. If the set of styles contained in the .dot are not defined in the DocGen template, those styles will not be used by DocGen when generating documentation.

To use a custom formatting template (.dot file) with DocGen:

1. Specify the path to the .dot file in the RTF documentation options. (For more information, see [“RTF documentation options” on page 386.](#))
2. Open the Documentation Designer. From the main menu, choose **Project | Documentation | Design Templates.**
3. In the Documentation Template Designer, choose **File | Properties.**
4. Choose the **Formatting Styles** tab.
5. Click the **New** button.
6. Choose the **Main** tab.
7. In the **Name** textbox, specify the style name defined in the .dot file.

To apply the style, it must be associated with an area property or control in the DocGen template. The style for each area property or control must be set separately.

Note Area property style settings override style settings that have been made to controls within the same area.

To set the Style for an Area Property:

1. Right click within a **Static Section** on the template.
2. Choose **Area Properties** from the right-click menu.
3. Choose the **Other** tab.
4. Choose the appropriate **Formatting Style** from the drop-down list.
5. Click **Ok.**

For more information on area properties and controls, see [“Setting area properties”](#) and [“Creating controls and setting control properties” on page 413.](#)

Generating reports with the Rational® RequisitePro® plug-in

The RequisitePro integration allows you to generate reports for your Together project. You can use DocGen to generate the reports in HTML, RTF, or text.

Setting up the reporting tool

The RequisitePro integration enhances DocGen, and requires some additional setup before you can use the reporting tool. Use a text editor to complete the instructions in this section.

To set up the reporting tool for use with RequisitePro, you need the following files:

- \$TGH\$\modules\com\togethersoft\modules\gendoc\templates\MetaModel.mm
- \$TGH\$\modules\com\togethersoft\modules\reqpro\ReqProMetaModel.mm

Using a text editor, add the contents of the ReqProMetaModel file to the end of the MetaModel file.

Running reports

The instructions in this section assume that you have completed the steps to set up the reporting tool in the previous section. To run reports for RequisitePro, the document generation tool in Together uses the template, report.tpl, as described in the instructions that follow. Note that you can also modify this template for your specific needs.

To generate a report using the standard template:

1. From the main menu, choose **Project | Documentation | Generate Using Template**.
2. Browse to select the template file report.tpl located in the `TGH\modules\com\togethersoft\modules\reqpro\report.tpl` directory.
3. Click **Ok**.

Your browser should appear with a report. For more information on report generation, see , [“Generating documentation using a template” on page 384](#).

To modify the existing template:

1. Save the default template as a backup.
2. Choose **Project | Documentation | Design Templates**.
3. Open the template file.

Refer to [Chapter 23, “Designing Custom Documentation Templates” on page 395](#) for more information on designing custom documentation report templates.

Documentation Generation Tips

This section provides tips that may be used when generating documentation for your projects.

Adding package-level documentation

When generating HTML documentation in Together (**Project | Documentation | Generate HTML**), documentation for any diagram can be added by updating the description property for the diagram. However, you can also document the physical package in your project by creating a separate file named, `package.html`. Place the `package.html` file into the appropriate source code directory, and it is read during the documentation generation process.

To add package level documentation, perform the following steps:

1. Create the `package.html` file. An example is shown below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

```

<html>
<body bgcolor="white">
<h2>Package Specification</h2>
<ul>
  <li><a href=""># Refer to any framemaker specification here #</a>
</ul>

<h2>Related Documentation</h2>

For overviews, tutorials, examples, guides, and tool documentation,
please see:
<ul>
  <li><a href=""># Refer to non-spec documentation here #</a>
</ul>
</body>
</html>

```

The documentation generator processes `package.html`, and copies everything between the `<body>` tags to the destination file, `package-summary.html`.

2. Save `package.html` to the appropriate source code directory.
3. Generate HTML documentation through Together. From the main menu select, **Project | Documentation | Generate HTML**.

Line breaks in PDF documentation

By default, Together preserves line breaks in the PDF documentation (**File | Print Documentation**). To disable this, edit `TGH\config\print.config` as follows:

```
print.doc.process.line.breaks = false
```

Automating documentation generation

You can update project documentation as part of a periodic automated build process by having the process script call the documentation generator in Together through the command line interface. To generate documentation from the command line without launching Together, use the following launchers:

- `TGH/bin/TogetherCon.exe` (Windows)
- `TGH/bin/Together.bat` (Windows)
- `TGH/bin/win32/umldoc.exe` (Windows)

Both `TogetherCon.exe` and `Together.bat` produce RTF and HTML documentation using modifiable template files (*.tpl files) as well as regular HTML documentation. `umldoc.exe` produces HTML documentation only. All of the launchers bypass the Together window. Table 44 lists each of the launchers to be used for automating documentation generation and the type of documentation that the launcher generates.

Table 44 Launchers and documentation type

Launcher	Documentation Type
TogetherCon.exe Together.bat	<ul style="list-style-type: none"> Documentation generated as RTF, HTML, or text using a modifiable template such as <code>ProjectReport.tpl</code> Documentation generated as regular HTML
umldoc.exe	Documentation generated as regular HTML

Launching the documentation generator from the command line

To generate RTF or HTML documentation from the command line without launching Together use the following command:

```
<launcher> [module] [options] <project>
```

where:

- `<launcher>` is `TogetherCon.exe`, `Together.bat`, or `umldoc.exe`.
- `[module]` is a command for the specific utility:

`-script=com.togethersoft.modules.genhtml.GenerateHTML` for generating HTML documentation. Using this module is analogous to using **Project | Documentation | Generate HTML** from within Together.

`-script=com.togethersoft.modules.gendoc.GenerateDocumentation` for generating RTF or HTML documentation. Using this module is analogous to using **Project | Documentation | Generate Using Template** from within Together.

Note The `script` parameter is not required when using `umldoc.exe` since it executes the `GenerateHTML` module only.

- `[options]` are those for the `GenerateDocumentation` or `GenerateHTML` modules. Options are listed in Table 45 and Table 46.
- `<project>` is the absolute path to the project, for example:
`C:/Borland/Together/samples/java/CashSales/CashSales.tpr`

At a bare minimum, the command needed to launch the `GenerateDocumentation` or `GenerateHTML` modules using `TogetherCon.exe` or `Together.bat` is:

```
<launcher> [module] <project> -d <output directory>
```

Tip Using the `GenerateDocumentation` module in the above command will generate RTF documentation by default. To change the output format to HTML or text, use the **format option**: `-format HTML` or `-format TXT`.

At a bare minimum, the command needed to launch the `GenerateHTML` module using the `uml.doc.exe` launcher is:

```
<launcher > <project> -d <output directory>
```

Where, `output directory`, is the absolute path to the output directory.

[Table 45](#) lists the available command line options for generating documentation using a template. [Table 46](#) lists the available command line options for generating regular HTML documentation.

Note The `Together.exe` and `Together.sh` launchers do not presently run fully in console mode. The Together shell opens displaying the Generate Documentation dialog box for setting the documentation generation options.

Documentation generation options

Generating automatic HTML or RTF documentation from a template (for example, `ProjectReport.tpl`) or generating regular HTML documentation allows for specific command-line options. Use the options described in these sections when generating documentation from the command line.

Options for generating documentation using a template

The options described in this section pertain to documentation generated as RTF, HTML, or text by using a modifiable template file, such as `ProjectReport.tpl`. Generating this type of documentation is the same as generating it from within Together by using **Project | Documentation | Generate Using Template**.

Each 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 documentation using template options and custom options. The regular options are listed in [Table 45](#).

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 the command line and adjust the template behavior dynamically.

Table 45 Command line options for generating documentation using a template

Command	Description
-template <path>	The template file. If omitted, the default "ProjectReport.tpl" template is used: ..modules\gendoc\templates\ProjectReport.tpl
-metamodel <path>	The 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 images.
-recurse	Create output for packages specified in [packagenames] and their subpackages.

Options for generating HTML documentation

The options described in this section pertain to regular HTML documentation. Generating this type of documentation from the command line is analogous to generating it from within Together by using **Project | Documentation | Generate HTML**.

The options in [Table 46](#) are valid for the `uml.doc.exe` launcher. These options also apply when using the `GenerateHTML` module with the `Together.bat` or `TogetherCon.exe` launchers.

Table 46 Command line options generating regular HTML documentation

Command	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.
-d <directory>	Destination directory for output files. Note: Not required for <code>Together.exe</code> , since this initiates the Together documentation generation dialog.

Table 46 Command line options generating regular HTML documentation (continued)

-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>	File to change style of the generated documentation.
-togethersettings	Use settings from \$TGH\$/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	Include diagram images.
-navtree	Generate navigation tree.
-nopackagedependencies	Do not show package dependencies in all class diagrams.

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 covers the details of documentation template creation.

The contents of this chapter are:

- “[Template Organization](#)” on page 395.
- “[Metamodel types](#)” on page 398.
- “[Creating new templates](#)” on page 400.
- “[Designing the template body](#)” on page 402.
- “[Accessing model elements through iterator sections](#)” on page 404.
- “[Reusing templates and stock sections](#)” on page 408.

Template Organization

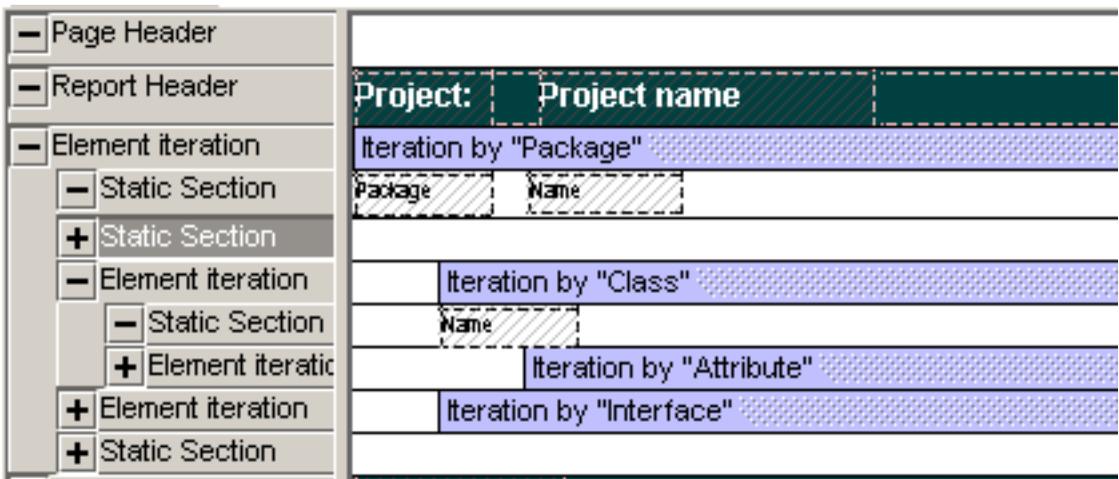
To open the Designer, select **Project | Documentation | Design Templates** from Together’s main menu. Alternatively, click the **Design** button when generating documentation from a template.

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 112](#).)

Figure 112 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 documentation; they are ignored for HTML and text 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 112](#) 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 112](#) 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 24, “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 iteration 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 400.

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 `TGH\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="[gendoc/gen_doc_by_template1/full_name_ELEMENT]"
  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="[gendoc/gen_doc_by_template1/full_name_DIAGRAM]"
  big_icon = "DiagramTypes/Default.gif"
  properties = { %package; stereotype; alias }
  contained_metatypes = { DIAGRAM_REFERENCE; DIAGRAM; NODE; LINK }
</metatype>
#-----
<metatype>
  name=NODE
  extends=ELEMENT
  rwi_entity=node
  full_name="[gendoc/gen_doc_by_template1/full_name_NODE]"
  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
- **UI Builder Model:** for reports based on the GUI builder model

and two kinds of templates:

- **Document Template:** for RTF, text, or non-frame HTML documentation
- **Frameset Template:** for HTML frame-based documentation

All of the topics discussed in this chapter and the next ([Chapter 24, “Documentation Template Controls”](#)) are appropriate for document templates. Frameset templates for multi-frame HTML are covered in [Chapter 25, “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 113](#) 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.

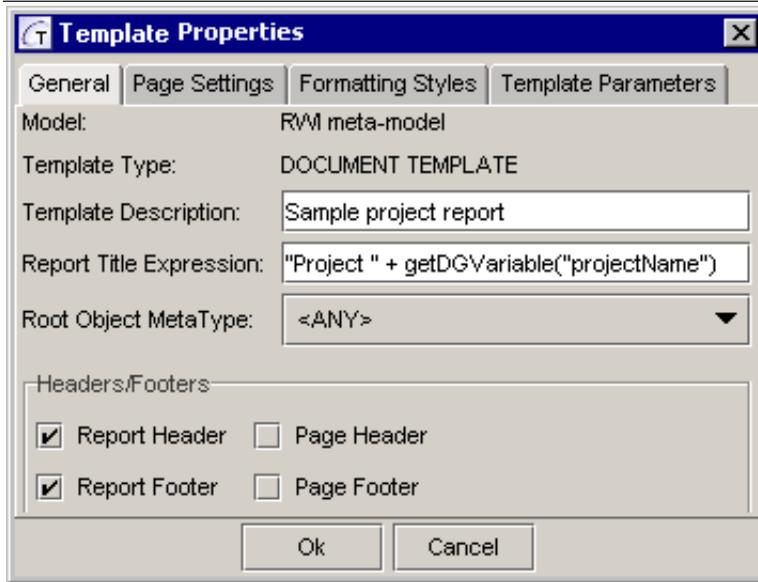
- **Page Settings** enables you to specify page size, margins, and landscape or portrait orientation. Measurements for the page size and margins are in inches.
- **Formatting Styles** are named text formatting styles in this template. When you create a new style by clicking the **New** button, a Style dialog window displays options for the style's properties such as font, size, color, and border. (See ["Text format settings" on page 419](#) for 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.

Tip Create as many Formatting Style types as you would like, then assign the desired Formatting Style to a Control (for more information on Controls see ["Creating controls and setting control properties" on page 413](#)). Once the Formatting Style has been assigned to a control(s), to change any style properties for these controls, e.g., font style, change the property in the appropriate Formatting Style. Once the Formatting Style is updated, the change will show immediately in the control(s).

- **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 113 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 **Insert Sibling Section** or **Insert 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 to create a sibling section.

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. Element iterators, property iterators, and folder sections each have a tab for **Output Style**, and a tab for **Other** options. (**Settings** displays for a folder section)

Output Style

The Output Style 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 (Settings tab for folder sections)

The Other tab (Settings tab for folder sections) 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.

Additionally, the folder section offers a commentary field in its Settings tab where you can enter a descriptive string to identify the folder section in the template.

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 26, “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.

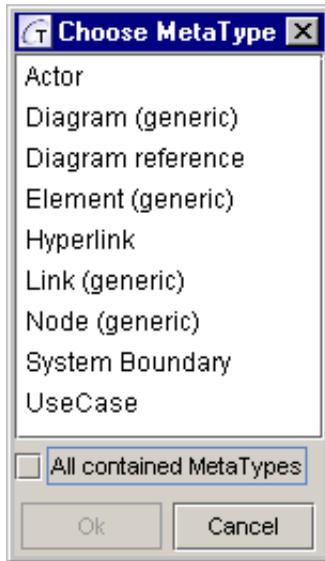
When you create a new element iterator, you must select its metatype from a list whose choices depend on the metatype of the parent section. If you check **All contained Metatypes**, the list expands to include the available metatypes of the items on the list as well. [Figure 114](#) shows the available 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 114 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 417](#) 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 417](#) for information on constructing expressions.)
- **Reverse scope order:** elements are always documented in ascending order. Check the **Reverse scope order** box to list elements in descending order.

Element property iterators

Element iterators traverse model elements. Element property iterators traverse element properties instead of elements. For instance, since an operation is a class property, you can use a property iterator to loop through all of the operations of a class.

A property iterator can reside inside an element iterator, folder, or property iterator. A property iterator must contain at least one static section, folder section, or call to a stock section or template. A property iterator may also contain an element iterator, or another property iterator.

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 398.)
- **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.
 - **Filter Expression:** Use this field to restrict the search elements that satisfy the filter properties.

A property iterator can iterate over multiple properties. Set the scope to **Set of Properties**. Use Ctrl+click to select multiple properties from the *Available properties* list, and then use the double-arrow button to move these properties to the *Selected properties* list. You can change the order in which the properties are documented by arranging the properties in the *Selected properties* list.

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 to 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

To create a stock section, select **Stock | New** from the main menu of the template designer. The resulting dialog box displays a choice of appropriate sections. Once the stock section has been created, 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, a field for the template parameter expression, and a field for setting the left indent. Check **Disabled** to skip the entire stock section.

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 Section** (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 also have the option to designate the stock section to use an **Intrinsic Property Iterator**. If you check this flag, then this means that the only available iteration scopes for an element iterator are *customized* and *programmed*. For folder sections, this means that the *metatype* chooser tab is absent.

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. This command displays each stock section 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.

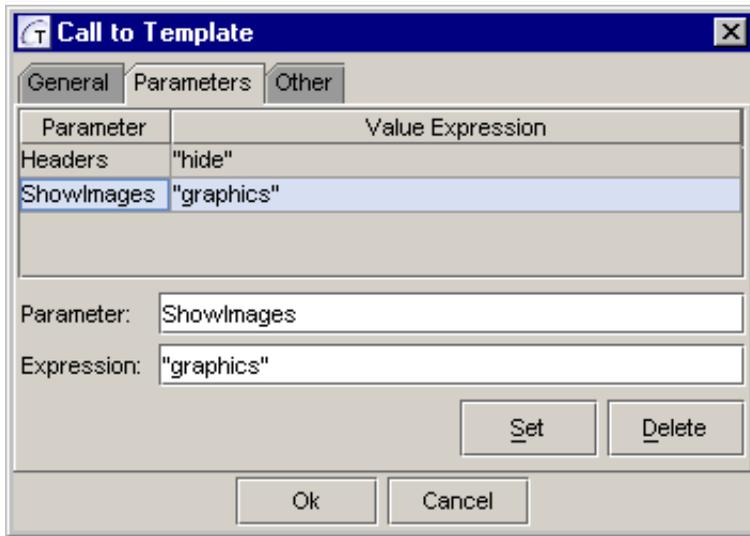
After creating a call to template section, right-click the new section and select **Properties**. This displays the Call to Template dialog box, where you specify the name of the called template. Clicking on the button to the right of the file name text field opens a dialog box for selecting the actual 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 115](#) 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 115 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 115](#):

```
getParam("ShowImages") == "graphics"
```

For more information, see [“Enabling Conditions” on page 403](#) and [“Formula and text controls” on page 417](#).

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.

Documentation Template Controls

The documentation generator 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 this chapter are:

- [“Using static sections, headers, and footers” on page 411](#)
- [“Creating controls and setting control properties” on page 413](#)
- [“Managing the display of output” on page 418](#)
- [“Hyperlinking controls to element documentation” on page 420](#)

Using static sections, headers, and footers

Documentation templates consist 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 396.](#))

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

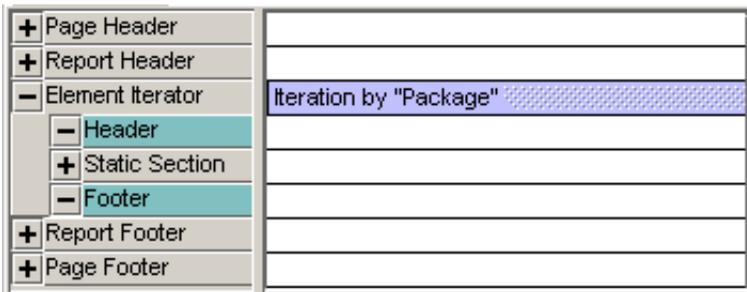
To create a static section, right click an existing section and choose **Insert Nested Section | Static Section** if the existing section is a folder or iterator or for any existing section, choose **Insert Sibling Section | Static Section**. Alternatively, select the existing section in the details pane and choose **Object** from the main menu.

The metatype for a static section is the metatype of its parent. (“[Body sections and the DocGen engine](#)” on page 396 has an introduction to metatypes.)

Report and page headers and footers lie outside the body of the template. You can turn report and page headers and footers on or off in the template options (use **File | Properties** from the Documentation Template Designer main menu).

Iterators and folders can also contain headers and footers. To add a header or footer to a folder or iterator, right click the section, and choose **Add Header** or **Add Footer**. [Figure 116](#) shows the aqua coloring of body section headers and footers in the scope pane.

Figure 116 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 403.)

Setting area properties

To set area properties for headers, footers, and static sections, choose **Area Properties** from the right-click menu of the details pane. 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:** for associating formatting styles with an area, setting style name expressions, and using a control delimiter.

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, choose **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
- **Formula:** for project-specific data (necessary when the data cannot be accessed with simple data controls)
- **Data Control:** for project-specific data
- **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 by right-clicking the control and choosing **Properties**, or by selecting the control and choosing **Object | Properties** from the main menu of the Documentation Template Designer. 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

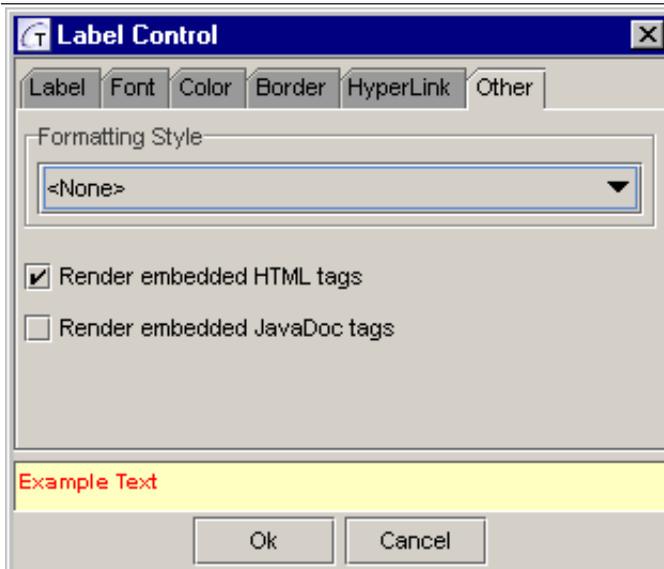
A label control generates static text that is independent of its containing section. The text does not depend on the metatype of the section or where the section

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 419 has more details on styles.

[Figure 117](#) shows the **Other** tab 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.

On the **Other** tab, you also have the option to *Render embedded Javadoc tags*. For example, enter the following text, `@link Class1`, on the **Label** tab, and check *Render embedded Javadoc tags* on the **Other** tab of the label control. You also need to mark the area that you wish to reference by the hyperlink as a location of the element's main documentation. To do this, you open the **Area Properties** for the static section under *Iteration by Class*, select the Hypertext Target tab, and check *Start of the current element's specific documentation*. Whenever you generate documentation, the label displays as a hyperlink to the documentation for `Class1`.

Figure 117 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 (URL):** Used 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 open a file selection dialog box.
- **Static (Resource):** Use the *Image Resource Expression* field and enter a path to the image file. For example, ". . \modules\com\togethersoft\modules\robustness\images\tv-boundary.gif"
- **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.
- **Element's Small / Large Icon:** Inserts a small or large icon representing the iterator's metatype.

Image controls can have borders. Use the **Border** tab of the image control to define borders around an image. Image controls can also be hyperlinked in the same way as label controls.

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 and choose **Insert Control | <control type>**. Alternatively, click the panel, and choose **Object | Insert Control | <control type>** from the main menu. If a copy of a control is in the clipboard, you can choose **Paste** from the **Object** main menu or the panel's right-click menu to insert it into the panel. You cannot drag a control into a panel from outside of the panel.

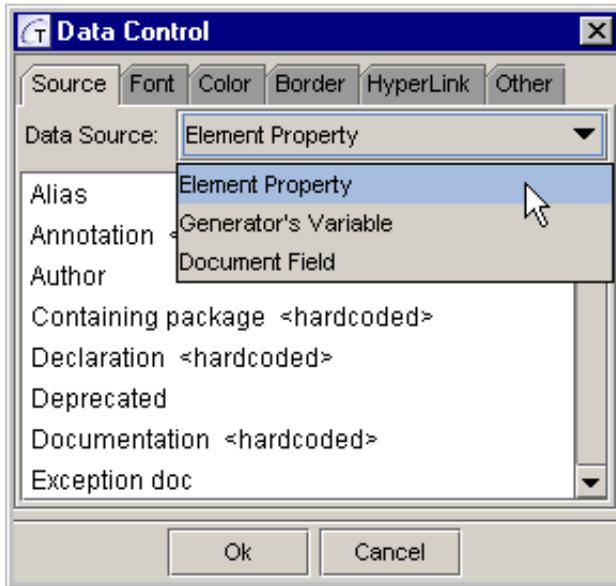
Data controls

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 396](#) 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 118](#) shows picking the data source from the drop-down menu in the Data Control dialog box.

Figure 118 Data Control dialog box



The menu includes 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 398.](#))

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.

To display custom properties:

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 26, "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 of 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 used 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:

1. Select the control. Notice that the mouse pointer changes to a cross with double-ended arrows.
2. With the control selected, drag the control to the desired position within the section.

To resize a control:

1. Place the mouse pointer on either the right or left edge of the rectangle. Notice that the mouse pointer changes to a double-ended arrow.
2. Drag the edge of the rectangle to a desired size.

Tip You can move multiple controls within a section at the same time. To select multiple controls, use Ctrl+click.

You cannot move a control from one section to another by dragging it. Instead, you must **Cut** using the control's right-click menu and **Paste** using the target section's right-click menu.

Text format settings

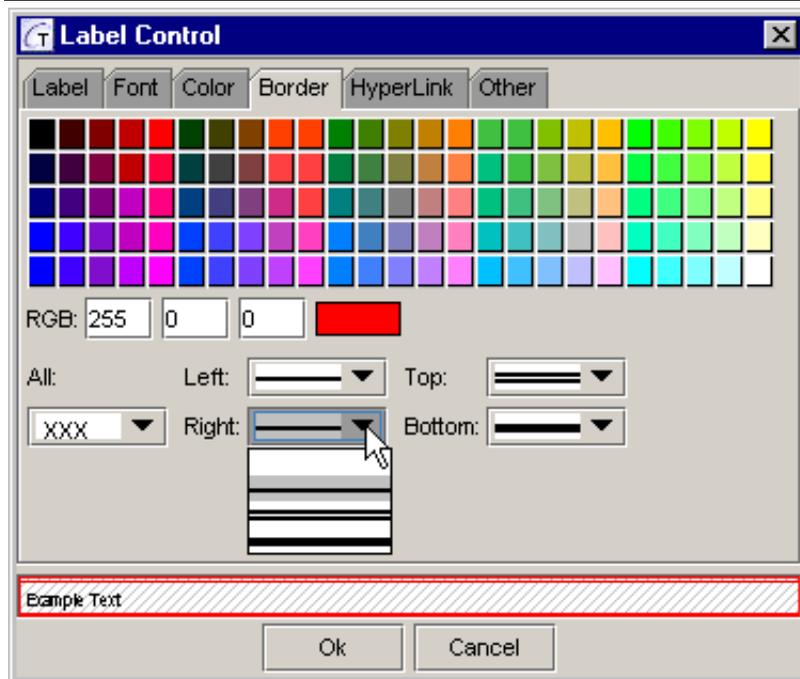
Properties for controls include tabs for formatting the presentation of text: **Font**, **Color**, and **Border**. Each page has a preview box at the bottom for evaluating the format settings.

On the **Font** tab, you can select the font family, size, style, and alignment within the rectangular area of the generated output.

On the **Color** tab, 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** tab, you can specify whether you want a border, and also determine its style, and color. The border styles are on drop-down lists, as shown in [Figure 119](#).

Figure 119 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 control's 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:** This is a text field.
- **Start of the current element's specific documentation:** This is 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:** This is 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 120](#) 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*.

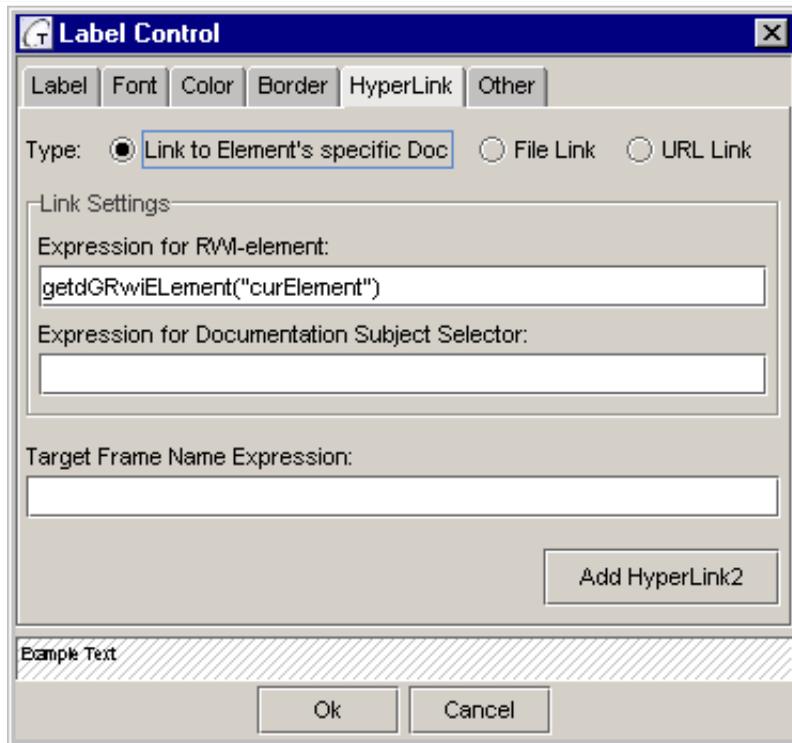
DocGen evaluates the *Expression for RWI-Element* to determine which element's documentation is the hyperlink target. For hyperlinking to the current model element, you can use the following expression:

```
getDGRwiElement("curElement")
```

This is the same expression as in [Figure 120](#). Expressions can be more complicated, such as:

```
findElement(getDGRwiProperty("curPropertyInstance")->  
    getSubproperty("$referencedElement"))
```

Figure 120 Hyperlinking a label to an element's main documentation



[“Creating hypertext links \(advanced\)” on page 429](#) has additional information on documentation hyperlinks.

Creating Multi-Frame HTML Documentation

Together can generate simple HTML, RTF, text, 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 23, “Designing Custom Documentation Templates”](#) and [Chapter 24, “Documentation Template Controls.”](#)

The contents of this chapter are:

- [“Multi-frame HTML document template basics”](#) on page 423.
- [“Defining the frameset structure”](#) on page 424.
- [“Designing the frameset template body”](#) on page 426.
- [“Creating hypertext links \(advanced\)”](#) on page 429.

Multi-frame HTML document template basics

Multi-frame HTML documentation divides project reports into 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 part describes the frameset file. The other part, 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 398.](#))

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 (**UI-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 templates 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 define 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 121](#).

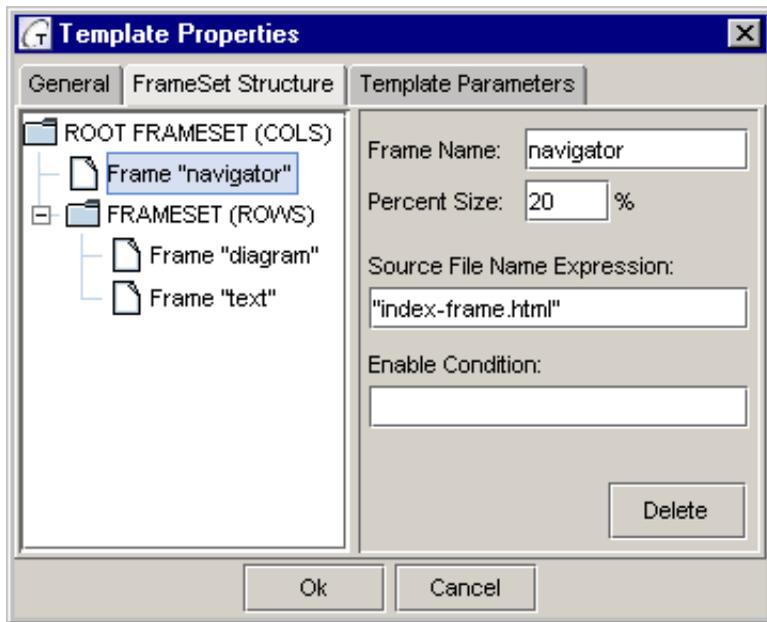
The FrameSet Structure tab shows the frameset as a tree on its left panel. The tree has two kinds of nodes:

- *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 121](#) 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 121](#) 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 121 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 121](#) 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 121](#) 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 403](#) 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 429](#) has additional details of this feature.

Enable Condition and Percent Size for a frame are identical to those for a frameset.

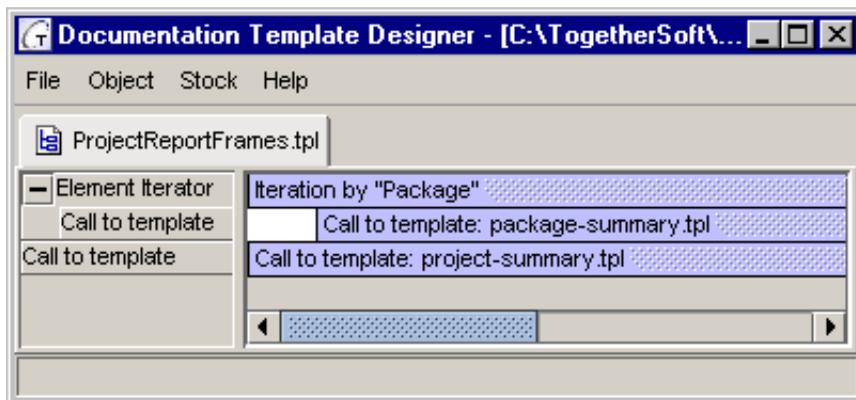
Designing the frameset template body

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 122](#) shows the “ProjectReportFrames” template, included with your Together installation.

Figure 122 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 123](#) shows the properties of the second call to template section from the "ProjectReportFrames" template of [Figure 122](#).

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.

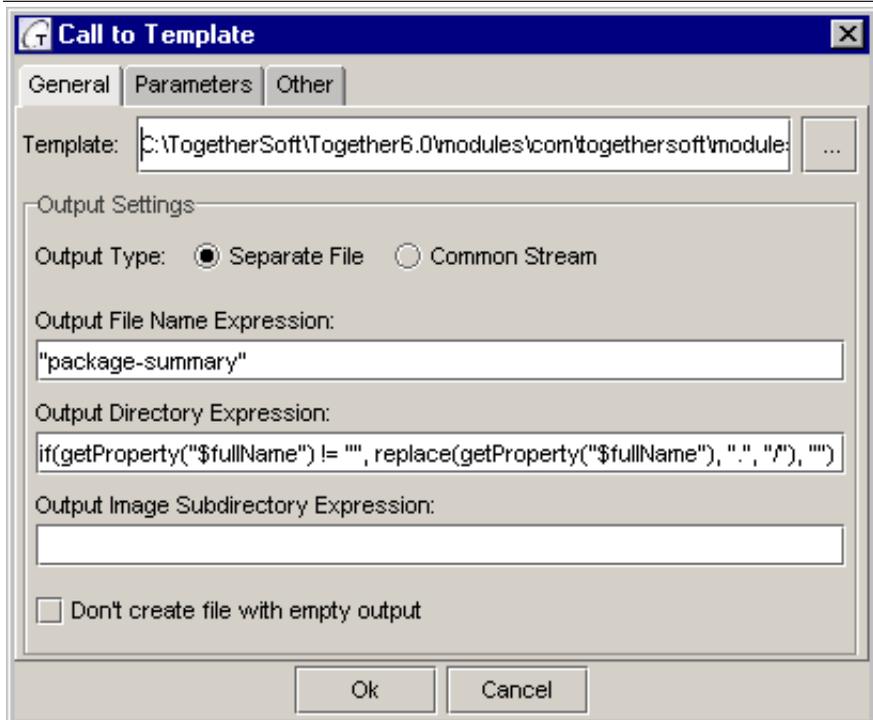
If the expression contains directories that do not yet exist, DocGen will create them when it processes the template.

The template in [Figure 123](#) 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 123 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 420](#) 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 420](#) 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 417](#) 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 *JDRefs*) 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 JDRefs into real hypertext links.

Each JDRef should conform to the rules described in the standard Javadoc documentation. 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: `label`
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 JDRefs into hyperlinks

A JDRef appears in one of two forms:

- inside `{@link}` tags embedded in documentation text. The JDRef 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**.

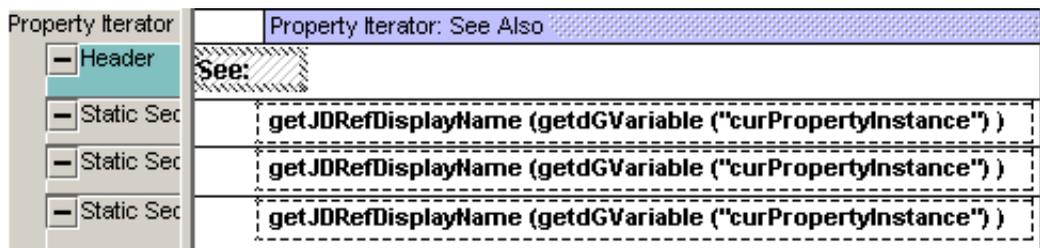
Converting the value of an element's property

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 124](#).

Figure 124 Iterator to display the `see` property values



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 JDocRef.

The following enable condition activates the first static section for element type JDocRefs only:

```
getJDocRefType (getDGVariable ("curPropertyInstance")) == "element"
```

Each static section area contains a formula control and nothing else. All formula controls use the same expression:

```
getJDocRefDisplayName (getDGVariable ("curPropertyInstance"))
```

The value of the expression is the text that will be displayed in the documentation.

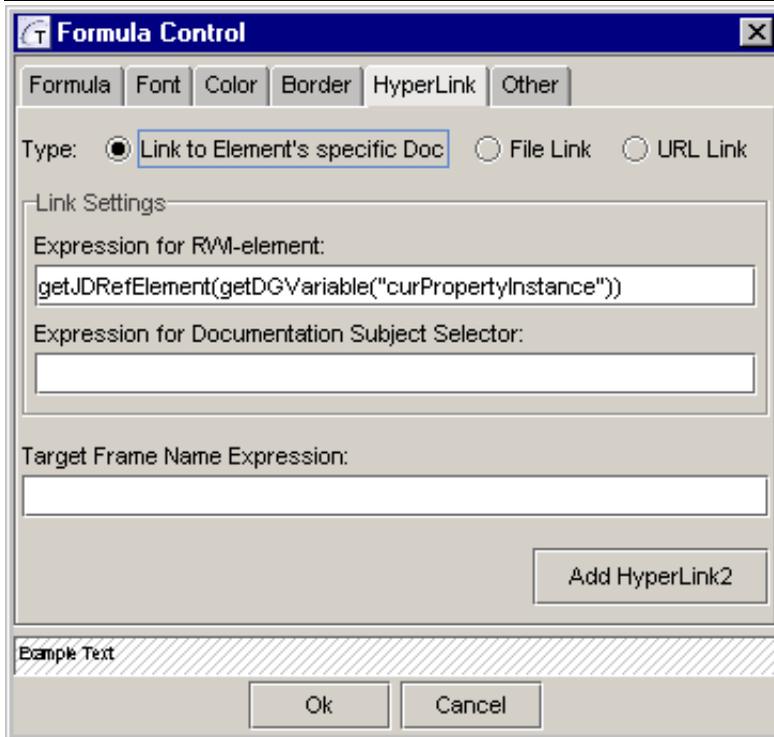
Figure 125 shows the HyperLink tab in the properties dialog of the formula control for the first static section. Note the value for Expression for RWI-element.

The function `getJDRefElement()` returns an RWI-element referenced by JDRef. The target of the link is the element's main documentation.

The second static section is activated when JDRef'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: `getJDRefURL(getDGVariable("curPropertyInstance"))`

The third static section is activated for JDRefs 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 JDRef does not refer to any element, the function `getJDRefElement()` always returns null, producing no hyperlinks.

Figure 125 HyperLink tab for a Formula Control



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 435.](#)
- [“Documentation generator functions” on page 438.](#)

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 398.](#)

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

DG variables and properties are case sensitive, however, DG functions are not case sensitive.

For example:

- getDGVariable("curItemNo") -- correct usage of curItemNo DG variable.
- getDGVariable("CURItemNo") -- incorrect usage of curItemNo DG variable.

[Table 47](#) lists the DG variables. [Table 48](#) lists the DG functions.

Table 47 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 398.) 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 398.) Availability: inside property iterators Access via: getDGVariable
curPropertyValue: String	Value of the current property. Availability: inside property iterators Access via: getDGVariable

Table 47 Documentation generator variables (continued)

Name and type	Description
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 <code>curPropertyInstance</code> 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: <code>getDGRwiProperty</code>
curPropertyInstance: String	Value of the current property instance. Availability: inside property iterators while iterating by instances of the specified property Access via: <code>getDGVariable</code>
curElement: RwiElement	Current model element. Availability: inside element iterators Access via: <code>getDGRwiProperty</code>
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: <code>getDGRwiProperty</code>
diagramMapElement: RwiElement	Diagram element; used to create images maps for diagrams. Availability: inside image controls Access via: <code>getDGRwiElement</code>
projectName: String	Project name. Availability: inside report / page header / footers Access via: <code>getDGVariable</code>
nowDateTime: String	Current date/time. Availability: inside report / page header / footers Access via: <code>getDGVariable</code>
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: <code>getDGOption</code>

Table 47 Documentation generator variables (continued)

Name and type	Description
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 48](#) lists the major functions that can be used in formula expressions and enabling conditions.

Table 48 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

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
<p>getDGOption</p>	<p>String getDGOption(String optionName)</p> <p>Parameter: name of a option</p> <p>Returns: value of the option with the specified name; the empty string if no such option is defined where the function is called</p> <p>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></p> <p>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.</p> <p>Example: default values for the options <code>inclSubpackages</code>, <code>inclDoc</code>, <code>DTLAdapter</code></p> <pre>DEFAULT_OPTIONS={inclSubpackages='yes';inclDoc='yes'; DTLAdapter='com.togethersoft.modules.doorslink.DTLAdapter'}</pre>
<p>getParam</p>	<p>String getParam(String paramName)</p> <p>Parameter: parameter name</p> <p>Returns: value of specified template parameter.</p> <p>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.</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
<p>invokeForName</p>	<pre>String invokeForName(String className, String methodName) String invokeForName(String className, String methodName, String param1) String invokeForName(String className, String methodName, String param1, String param2)</pre> <p>Parameters: className: fully-qualified name of the user-provided class. This class should not be abstract.</p> <p>The Documentation Generator creates an instance of <code>className</code> and calls the method <code>methodName</code> with this instance. An instance object is created for each entry of <code>invokeForName</code> 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 <code>className</code> parameter is changed.</p> <p>methodName: name of the method in the class <code>className</code>.</p> <p>The method should have the following signature:</p> <pre>String methodName (..gendoc.api.GenDocContext)</pre> <p>The last parameter is an instance of <code>..gendoc.api.GenDocContext</code>, which provides the following methods:</p> <ul style="list-style-type: none"> • <code>RwiReference</code> <code>getRwiReference</code> returns <code>RwiReference</code> if the current DG iteration element is an RWI-reference within a diagram. Otherwise the method returns <code>null</code>. • <code>RwiElement</code> <code>getRwiElement</code> returns an <code>RwiElement</code>. If the current DG iteration element is an RWI-reference, then returned element is <code>rwiReference.getElement()</code>. Otherwise, returned element is the current DG iteration RWI-element. • <code>String</code> <code>getParameter1()</code> returns the value of the first optional parameter passed to <code>invokeForName</code> function, or <code>null</code> if the parameter is omitted. • <code>String</code> <code>getParameter2()</code> returns the value of the second optional parameter passed to <code>invokeForName</code> function, or <code>null</code> if the parameter is omitted. <p>Returns: value calculated by the method named <code>methodName</code> of the user-provided class.</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
getContainingDiagram	<p>RwiDiagram getContainingDiagram()</p> <p>Returns: RWI-diagram containing the primary reference to the current element</p> <p>Example: rwiElement -> getContainingDiagram()</p>
isDiagram	<p>boolean isDiagram()</p> <p>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.)</p> <p>Example calls:</p> <pre>rwiElement->isDiagram() getDGRWIElement("diagramMapElement")->isDiagram()</pre> <p>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 431.</p>
isImported	<p>boolean isImported()</p> <p>Returns: true if the current element in the diagram is a shortcut; false if it is not a shortcut</p>
getSubproperty	<p>String getSubproperty(RwiProperty rwiProperty, String subpropertyName)</p> <p>Parameters:</p> <p>rwiProperty: the element property subpropertyName: the name of its subproperty</p> <p>Returns: the value of subpropertyName contained in rwiProperty.</p> <p>Example call: rwiProperty->getSubproperty(subpropertyName)</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
hasSubproperty	<p>String hasSubproperty(RwiProperty rwiProperty, String subpropertyName)</p> <p>Parameters: rwiProperty: the element property subpropertyName: the name of its subproperty</p> <p>Returns: true if rwiProperty has the specified subproperty; false otherwise.</p> <p>Example call: rwiProperty->hasSubproperty(subpropertyName) String getJdrefType(String jdref)</p> <p>Parameter: JavaDoc reference</p> <p>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"</p> <p>Since: Together 5</p>
getJdrefType	<p>String getJdrefType(String jdref)</p> <p>Parameter: JavaDoc reference</p> <p>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"</p> <p>Since: Together 5</p>
getJdrefDisplayName	<p>String getJdrefDisplayName(String jdref)</p> <p>Parameter: JavaDoc reference</p> <p>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.</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
getJDRefElement	<p>RwiElement getJDRefElement(String jdref)</p> <p>Parameter: JavaDoc reference</p> <p>Returns: If the specified JavaDoc Reference is an "element" reference (if it has the form package.class#member label, where package.class#member represents some model element) and referenced element exists in the model, the function returns this element; otherwise returns null.</p> <p>Since: Together 5</p>
getJDRefURL	<p>String getJDRefURL(String jdref)</p> <p>Returns: If the specified JavaDoc Reference is a "url" reference (if it has the form label) the function returns the text URL#value; otherwise, returns an empty string</p> <p>Since: Together 5</p>
getCodeElement	<p>Object getCodeElement(RwiElement rwiElement)</p> <p>Returns: An object.</p> <p>Details: The function passes the call to rwiElement.getCodeElement() method declared in com.togethersoft.openapi.rwi.RwiElement interface.</p> <p>Usage: This function is used in the template expressions in conjunction with one of the following functions: findMember(), findNode(), findLink(), findPackage().</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
<p>getCodeElements</p>	<p>Enumeration <code>getCodeElements(RwiElement rwiElement)</code></p> <p>Parameter: an RWI-element.</p> <p>Returns: code elements associated with the parameter.</p> <p>Details: This function passes the call to the <code>rwiElement.getCodeElements()</code> method declared in <code>com.togethersoftware.openapi.rwi.RwiElement</code> interface.</p> <p>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.</p> <p>Example: When <code>Recognize JavaBean/C++ properties</code> is on, each JavaBean/C++ 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 430). But this RWI-element is actually a kind of a proxy. It is not identical to any of those 3 elements making up the JavaBean/C++ property. Thus, you cannot directly use the RWI-element representing a JavaBean/C++ property on the diagram to establish a hyperlink to anything contained in the generated documentation. The solution is to use this expression:</p> <pre>findDocumentedMember(getCodeElements(getDGRwiElement("diagramMapElement")))</pre> <p>The function <code>findDocumentedMember()</code> returns one of the RWI-elements associated with the JavaBean/C++ property and which is definitely presented in the generated documentation.</p> <p>Since a diagram contains ordinary elements as well as properties, the expression for diagram hyperlinks connecting RWI-elements may be more complicated:</p> <pre>if(getDGRWIElement("diagramMapElement")-> hasPropertyValue("\$shapeType", "BeanProperty"), findDocumentedMember(getCodeElements(getDGRWIElement("diagramMapElement"))), getDGRWIElement("diagramMapElement"))</pre> <p>Since: Together 5</p>
<p>findElement</p>	<p>RwiElement <code>findElement(String uniqueName)</code></p> <p>Parameter: string with the unique name of an RWI-element to be found</p> <p>Returns: an element of the given name.</p> <p>Details: This function passes the call to the <code>RwiModel.findElement()</code> method to do the search.</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
findMember	<p>RwiElement findMember(Object codeElement)</p> <p>Parameter: code element of interest.</p> <p>Returns: the member for codeElement.</p> <p>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>.</p> <p>Since: Together 5</p>
findNode	<p>RwiElement findNode(Object codeElement)</p> <p>Parameter: code element of interest.</p> <p>Returns: the node for codeElement.</p> <p>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>.</p> <p>Since: Together 5</p>
findLink	<p>RwiElement findLink(Object codeElement)</p> <p>Parameter: code element of interest.</p> <p>Returns: the link for codeElement.</p> <p>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>.</p> <p>Since: Together 5</p>
findPackage	<p>RwiElement findPackage(Object codeElement)</p> <p>Parameter: code element of interest.</p> <p>Returns: the package for codeElement.</p> <p>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>.</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
findDocumentedMember	<pre>RwiElement findDocumentedMember(Enumeration codeElements) RwiElement findDocumentedMember(Enumeration codeElements, String subjectSelector)</pre> <p>Parameters: Code elements and subject selectors to search.</p> <p>Returns: found <code>RwiElement</code> that satisfies these conditions:</p> <ul style="list-style-type: none"> • it is associated with the passed <code>codeElements</code> • it is an <code>RwiMember</code> • it will be presented in all generated documents by its Main Documentation or, if <code>subjectSelector</code> is specified, by its “specific” documentation associated with the passed <code>subjectSelector</code>. <p>Returns <code>null</code> if the requested element doesn't exist in the model.</p> <p>Details: This function should be used together with the function <code>getCodeElements()</code>. It calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findMember()</code>.</p> <p>Since: Together 5</p>
findDocumentedNode	<pre>RwiElement findDocumentedNode(Enumeration codeElements) RwiElement findDocumentedNode(Enumeration codeElements, String subjectSelector)</pre> <p>Parameters: See <code>findDocumentedMember</code>.</p> <p>Returns: See <code>findDocumentedMember</code>.</p> <p>Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findNode()</code>.</p> <p>Since: Together 5</p>
findDocumentedLink	<pre>RwiElement findDocumentedLink(Enumeration codeElements) RwiElement findDocumentedLink(Enumeration codeElements, String subjectSelector)</pre> <p>Parameters: See <code>findDocumentedMember</code>.</p> <p>Returns: See <code>findDocumentedMember</code>.</p> <p>Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findLink()</code>.</p> <p>Since: Together 5</p>
findDocumentedPackage	<pre>RwiElement findDocumentedPackage(Enumeration codeElements) RwiElement findDocumentedPackage(Enumeration codeElements, String subjectSelector)</pre> <p>Parameters: See <code>findDocumentedMember</code>.</p> <p>Returns: See <code>findDocumentedMember</code>.</p> <p>Details: The function calls the method <code>com.togethersoft.openapi.rwi.RwiModel.findPackage()</code>.</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
findDocBySubjectSelector	<p>String findDocBySubjectSelector(String subjectSelectorList)</p> <p>Parameter: list of subject selectors separated with semicolons.</p> <p>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.</p> <p>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.</p> <p>Note: A blank subject selector is allowed; it will refer to the Main Documentation of an element.</p> <p>Example: findDocBySubjectSelector("package-summary;summary") returns the first generated document for one of the subject selectors: "package-summary", "summary"</p> <p>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 424.)</p> <p>Since: Together 5</p>
findDocByTemplate	<p>String findDocByTemplate(String templateList)</p> <p>Parameter: list of template names (without file name extensions) separated with semicolons.</p> <p>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.</p> <p>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.</p> <p>Example: findDocByTemplate("all-classes;all-diagrams") returns the first document produced by one of the templates: "all-classes.tpl" and "all-diagrams.tpl"</p> <p>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 424.)</p> <p>Since: Together 5</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
checkStockSectionOutput	<pre>boolean checkStockSectionOutput(String stockSectionName, RwiElement rwiElement)</pre> <p>Parameters: name of a stock section and an RWI element passed to the stock section as the current model element.</p> <p> stockSectionName - name of the Stock Section to be tested.</p> <p> rwiElement - RWI-element passed to the Stock Section as the current model element.</p> <p>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 rwiElement passed to it as the current model element; false otherwise. No actual output is produced from a call to this function.</p> <p>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.</p> <p>Example:</p> <pre>checkStockSectionOutput("Included Diagram List", getDGRwiElement("curElement"))</pre> <p>Since: Together 5</p>
getPropertyExt	<pre>String getPropertyExt(String propertyName)</pre> <p>Parameter: name of a property</p> <p>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 398.)</p> <p>Example:</p> <pre>rwiElement->getPropertyExt(propertyName)</pre> <p>rwiElement is the element whose property is to be checked.</p> <p>See also: <code>getProperty()</code></p>
Utility String Functions	
substring	<pre>String substring(String str, int beginIndex) String substring(String str, int beginIndex, int endIndex)</pre> <p>Parameters: identical to those in the standard Java String.substring() methods.</p> <p>Returns: a new string that is a substring of str.</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
replace	<p>String replace(String str, String oldStr, String newStr)</p> <p>Parameters: three strings.</p> <p>Returns: a new string produced by replacing all occurrences of oldStr in str with newStr. This operation is case sensitive.</p> <p>Example: replace("aBC-BC-bc", "BC", "XY") returns "aXY-XY-bc"</p> <p>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:</p> <pre>replace(getContainingPackage()->getProperty("\$fullName"), ".", "/")</pre> <p>See also: "Creating hypertext links (advanced)" on page 429.</p> <p>Since: Together 5</p>
duplicate	<p>String duplicate(String str, int num)</p> <p>Parameters: a string and a non-negative number.</p> <p>Returns: a new string resulting from duplicating str num times. If num is 0, returns an empty string.</p> <p>Example: duplicate("aBC", 3) returns "aBCaBCaBC"</p> <p>Since: Together 5</p>
length	<p>int length(String str)</p> <p>Parameter: a string.</p> <p>Returns: the number of characters in the string (string length).</p>
str	<p>String str(Numeric N)</p> <p>Parameter: a number</p> <p>Returns: a string representation of the number.</p>
val	<p>Numeric val(String str)</p> <p>Parameter: a string.</p> <p>Returns: the number represented by the string. If conversion is impossible, the function returns 0.</p>
	<p>The following functions, commonly provided in Together formula queries, are also very useful in DG expressions.</p>
getProperty	<p>String getProperty(String rwiPropertyName)</p> <p>Parameter: the name of an RWI-property.</p> <p>Returns: the value of the specified RWI property of the (current) element, or the empty string if the element has no such property.</p> <p>Example call:</p> <pre>rwiElement->getProperty(rwiPropertyName)</pre> <p>.rwiElement is the element whose property is returned.</p> <p>See also: getPropertyExt()</p>

Table 48 Documentation generator functions (continued)

Function name	Signature and Description
hasProperty	<p>boolean hasProperty(String rwiPropertyName)</p> <p>Parameter: the name of an RWI-property.</p> <p>Returns: true if the (current) element has the specified property; false otherwise.</p> <p>Example call:</p> <pre>rwiElement->hasProperty(rwiPropertyName)</pre> <p>rwiElement is the element whose property is to be checked.</p>
hasPropertyValue	<p>boolean hasPropertyValue (String rwiPropertyName, String value)</p> <p>Parameters: an RWI-property and a possible RWI property value.</p> <p>Returns: true if the (current) element has specified property with the required value; false otherwise.</p> <p>Example call:</p> <pre>rwiElement->hasPropertyValue (rwiPropertyName, value)</pre> <p>rwiElement is the element whose property is to be checked.</p>
if	<p>type if(boolean condition, type value1, type value2)</p> <p>Parameters: any Boolean condition and two values of the same type. The type can be any data type allowed in queries.</p> <p>Returns: If the Boolean condition is true, the function returns value1. If the condition is false, the function returns value2.</p>
getContainingNode	<p>RwiNode getContainingNode()</p> <p>Returns: the RwiNode element that contains the current element. Can be called for RWI member or node current element.</p> <p>Possible call: rwiElement->getContainingNode()</p> <p>Example: The following expression calculates visibility modifier for the class/interface member:</p> <pre>if (hasProperty("\$private"), "private", if (hasProperty("\$protected"), "protected", if (hasProperty("\$public") && !getContainingNode()-> hasProperty("\$interface"),"public", "")))</pre> <p>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>

CONTEMPORARY SOFTWARE PRACTICES

- Chapter 27, “Refactoring”
- Chapter 28, “Using the Testing Framework”
- Chapter 29, “Templates and Patterns”
- Chapter 30, “Audits and Metrics”

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 453
- “Showing code structure” on page 454
- “Moving classes, interfaces, attributes, and operations” on page 456
- “Renaming” on page 458
- “Encapsulating attributes” on page 459
- “Extracting interfaces, superclasses, and operations” on page 460
- “Summary of refactoring commands” on page 465

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 | Refactoring** from Together's main menu.
- Select **Refactoring** from the right-click menu of the currently selected item in the Explorer or the Designer.
- Select **Refactoring** 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 *Refactoring* 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 | Refactoring menu contains the full list of refactorings. Only those refactoring that are appropriate for the currently selected element are enabled.

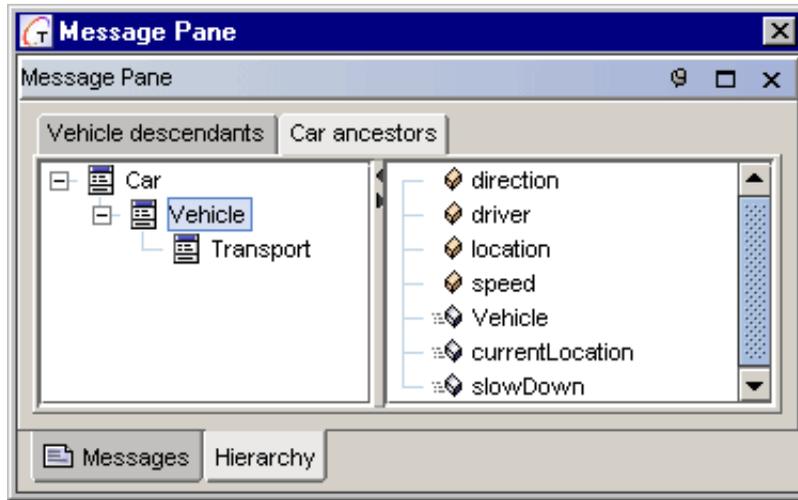
Showing code structure

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.

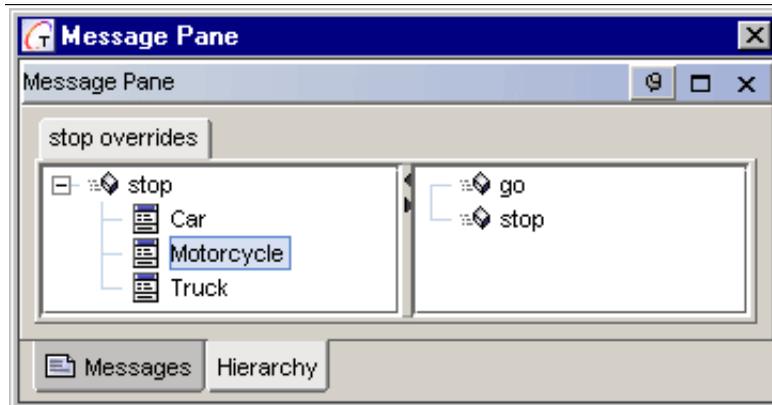
- *Show Ancestors*: (classes and interfaces) shows all of the ancestors of the selected class or interface.
- *Show Descendants*: (classes and interfaces) shows all of the descendants of the selected class or interface.
- *Show Implementing Classes*: (interfaces only) shows the classes that implement 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 126 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 127 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 Refactoring | 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 **Refactoring** | **Move Class** from the right-click menu of the class, *or*
 - Choose **Tools** | **Refactoring** | **Move Class** from the main menu, *or*
 - Move the cursor into the name of the class in the Editor, then choose **Refactoring** | **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 Refactoring | Pull Up Member or Refactoring | 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.

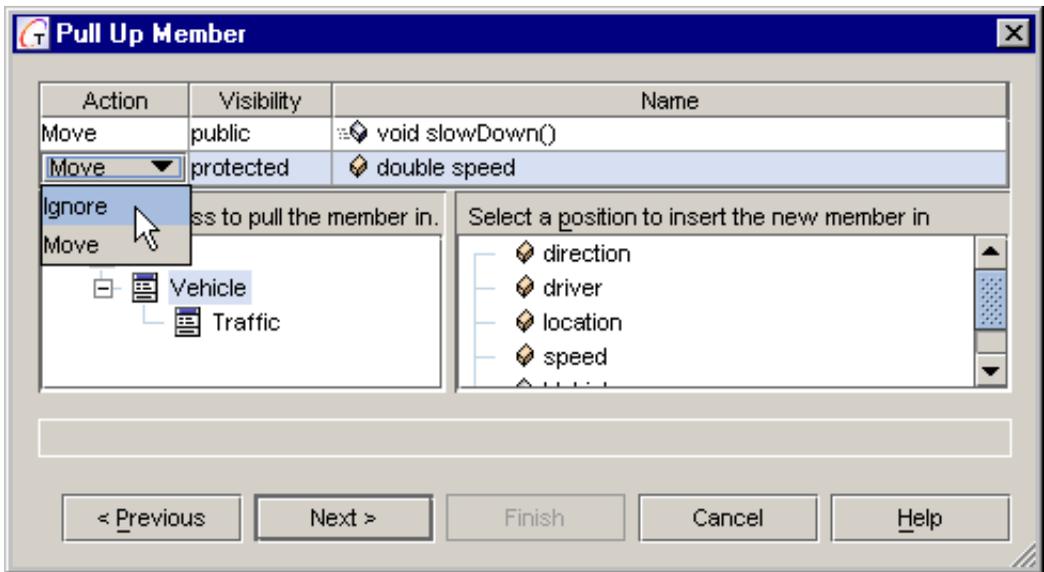
When you pull up or push down members, Together displays a dialog box for specifying additional information that it must know to make the move. The dialog box takes you through a series of pages:

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 128](#) 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 128 *Pull Up Member* dialog box displaying target classes. The operation will be moved but the attribute will be ignored.



Renaming

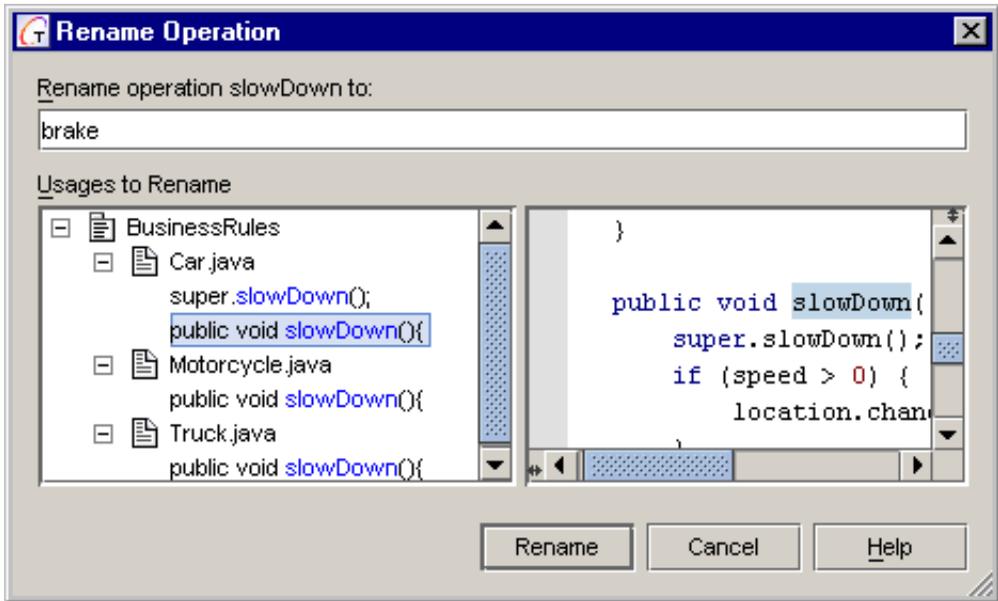
You can rename any code element: class, interface, operation, attribute, parameter, or local variable. Together propagates the name changes to dependant code in the project files.

To rename a local variable or parameter, put your cursor on the variable name in the Editor and right click. Choose Refactoring | 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 | Refactoring | Rename Property. For any other code element, you can use the Editor, the Designer, the Explorer, or the Tools menu.

Figure 129 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 129 Renaming an operation



Encapsulating attributes

Encapsulating an attribute means hiding it by making it private and providing setters and getters where the attribute is used.

You can encapsulate an attribute (or multiple attributes) through its right-click menu by selecting Refactoring | Encapsulate Attribute. The resulting dialog box has multiple displays. With the first display, you can specify:

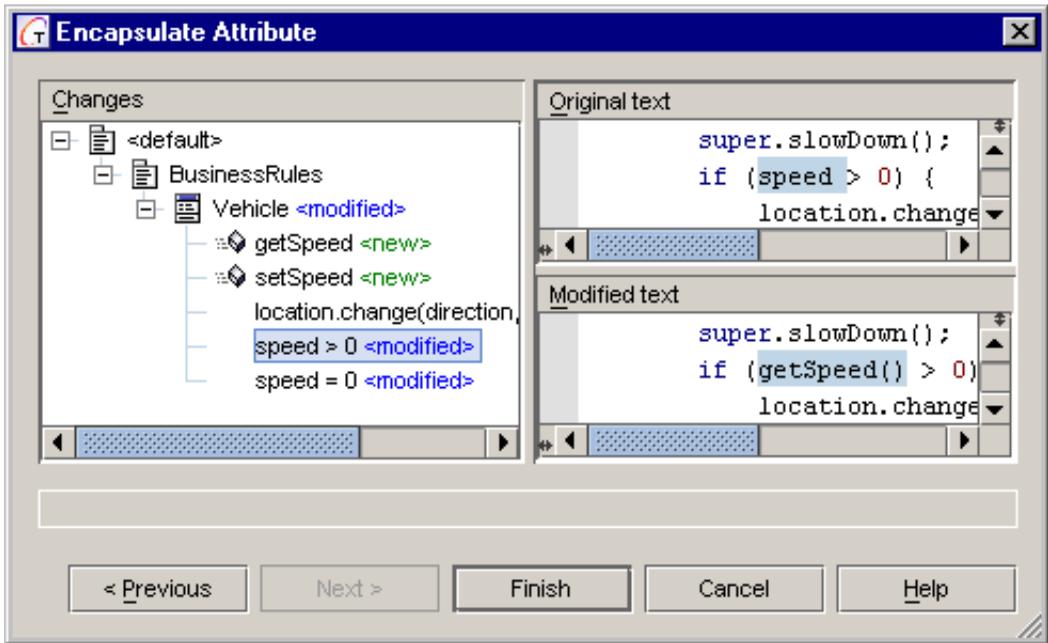
- **Visibilities:** Dropdown list to select the visibilities of the attribute and its getter and setter methods.
- **Names:** For getters and setters. You can change the names of either. (There
- **Self Encapsulate:** Check box for replacing all references to the attribute by calls to its getter and 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.

Note Together does not create setters for attributes that are final.

Figure 130 shows the results of encapsulating an attribute named `speed`. The changed lines of code are listed on the left frame. Selecting a line in the left frame displays the original and modified text on the right frame.

Figure 130 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 **Refactoring | Extract Interface** or **Refactoring | 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:

1. Select the code fragment that you want to extract and place your cursor in the selection.
2. Choose **Refactoring | Extract Operation** from the Editor's right-click menu. [Figure 131](#) 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.

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 131](#) 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;
}
```

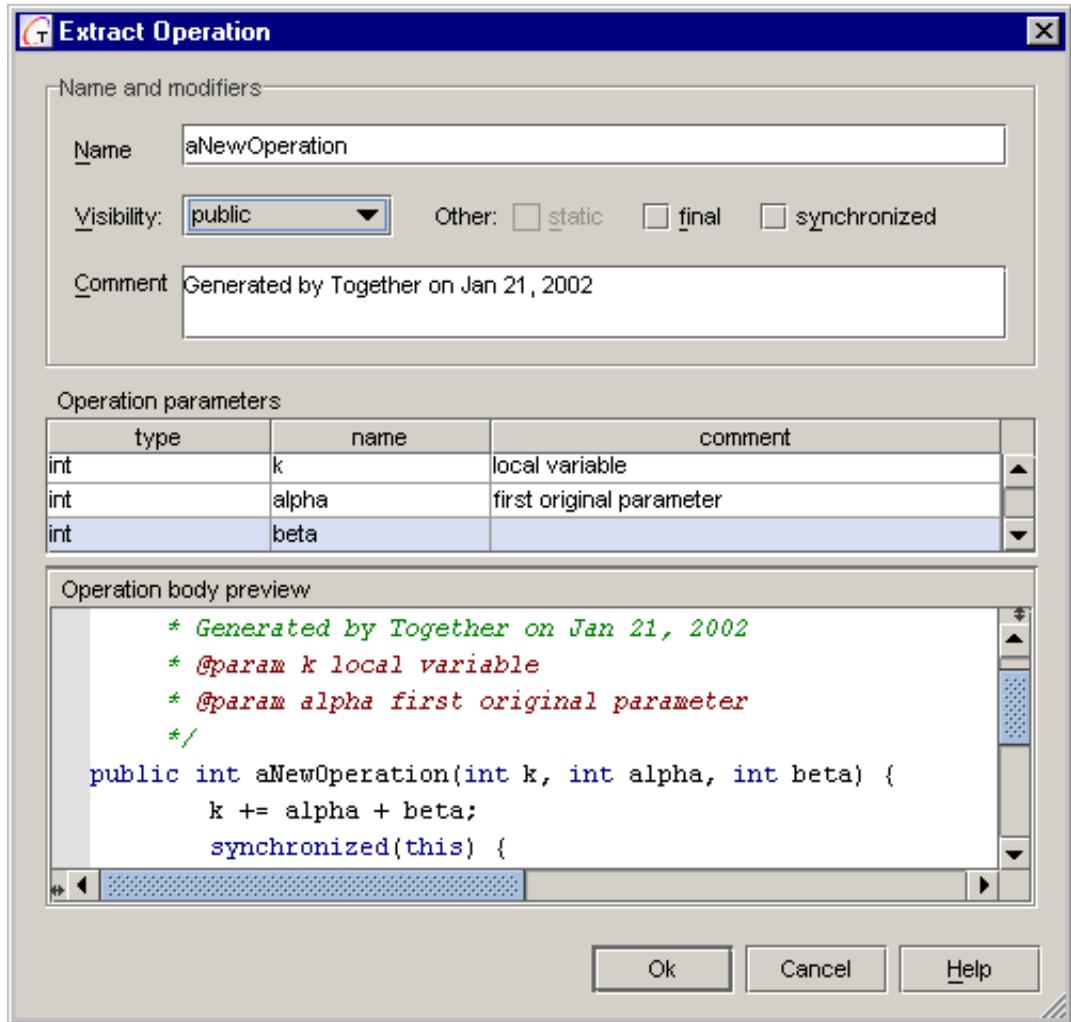
The resulting code is as follows:

```
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;
}
```

Figure 131 Extract Operation



Summary of refactoring commands

Table 50 lists all of the refactoring commands. You can invoke most commands through the Designer, and 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 49 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.

Table 49 (continued)Refactoring commands (continued) (continued)

Command	Applies To	Editor	Designer/ Explorer	Comments
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.

Table 50 Refactoring commands

Command	Applies To	Comments
Encapsulate Attribute	One or more attributes	
Extract Interface	One class	All class members are listed. Default action is 'Ignore.'
	Several classes	Only members with equal signatures are listed. Default action is 'Abstract.'
	Several members of a single class	Selected members are listed. Default action is 'Abstract.'
Extract SuperClass	One class or interface	All class members are listed. Default action is 'Ignore.'
	Several classes or interfaces	Only members with equal signatures are listed. Default action is 'Move.'
	Several members of a single class or interface	Selected members are listed. Default action is 'Move.'
Move Class	One or more classes	
Pull Up Member	One or more members of a single class	
Pull Down Member	One or more members of a single class	
Rename Attribute	One attribute	
Rename Class	One class	
Rename Interface	One interface	
Rename Operation	One operation	
Rename Property	One property	
Show Ancestors	One or more classes or interfaces	
Show Descendants	One or more classes or interfaces	

Show Implementing Classes	One or more classes or interfaces	
Show Overrides	One operation	

Using the Testing Framework

This chapter provides instructions for creating and running tests using the Together 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 470](#)
- [“Before you begin” on page 471](#)
- [“Creating a test project” on page 472](#)
- [“Configuring options” on page 474](#)
- [“Setting up version control” on page 477](#)
- [“Browsing test servers and working with profiles” on page 478](#)
- [“Developing unit tests for source code” on page 479](#)
- [“Working with unit tests” on page 485](#)
- [“Developing visual tests for a user interface” on page 486](#)
- [“Working with visual tests” on page 491](#)
- [“Viewing test results” on page 495](#)
- [“Distributed testing” on page 497](#)

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 set up 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 132, “Tests tab showing a test plan” on page 472](#).

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.

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 498](#) 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 option 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.

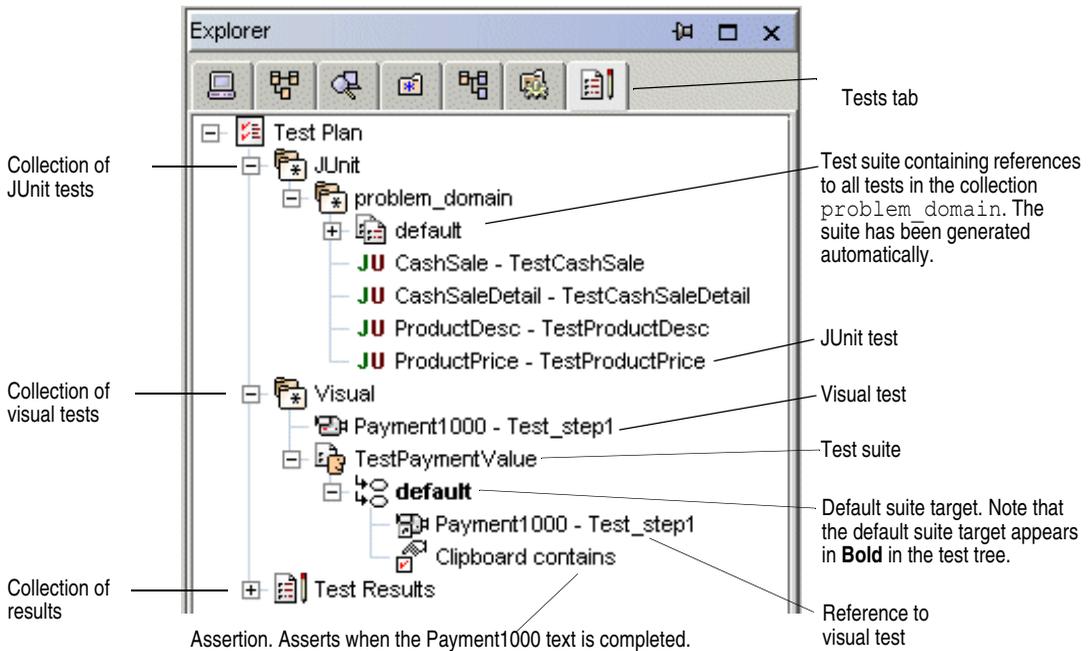
To access the sample test project for CashSales, follow these steps:

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 471](#) for more information.

3. Select the Tests tab to view the test plan as shown in [Figure 132](#).

Figure 132 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 471](#) 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
```

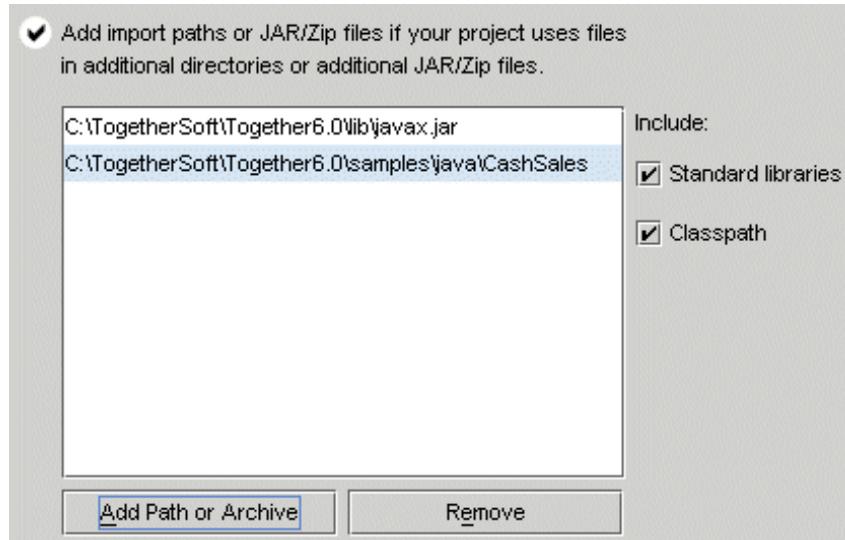
2. Create a new project using **File | New Project Expert**. (For additional information, see [“Using the New Project Expert” on page 98.](#))

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.

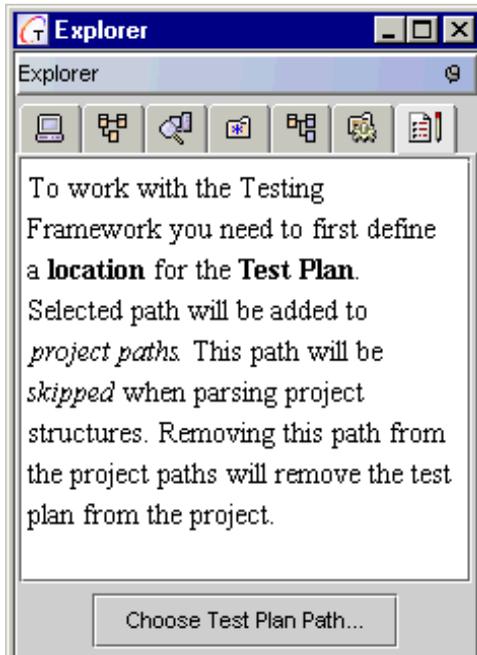
3. Add paths to the code being tested when the New Project Expert displays the screen shown in [Figure 133](#).

Figure 133 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 134](#) appears.

Figure 134 Tests tab for a new project



6. Click **Choose Test Plan Path** and select the project path for the test plan directory.

Tip To view or modify the project settings for your newly created project, open the Project Properties dialog (Project | Project Properties). The path to the test plan is included on the Project Paths tab, along with the paths for testing that you specified. Select the test plan path and observe that the *Skip path* box is checked automatically. Do *not* uncheck *Skip path* for the test plan path, because the test plan sources should not be parsed.

After completing these steps, you can proceed to [“Developing unit tests for source code” on page 479](#) or [“Developing visual tests for a user interface” on page 486](#).

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, choose **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 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 483](#).

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 480](#) 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 470](#).

To set JUnit options, choose **Testing Framework | Building Unit Tests | JUnit Family**. By default, the *Active* option for JUnit is selected, allowing you to create JUnit tests.

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. For more information about JUnitX, visit the JUnit website at <http://www.junit.org/index.htm>.

To set JUnitX options, choose **Testing Framework | Building Unit Tests | JUnitX Family**. By default, the *Active* option for JUnitX is selected, allowing you to create JUnitX tests.

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**. By default, the *Active* option for Cactus is selected, allowing you to create Cactus tests.

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**. By default, the *Active* option for HttpUnit is selected, allowing you to create HttpUnit tests.

Setting up version control

You can set up version control for a test project in order to share tests and test results.

To set up version control for a project, follow these steps:

1. Open a test project (* .tpr).
2. From the main menu, choose **Project | Version Control | Setup for this Project**.
3. On the Project Paths tab, check *Version control project path* and click **Options**. The Project Options: Version Control dialog opens.

4. Select Version Control in the tree and then check the options that apply to your project. Click **Apply** to continue.
5. Expand Version Control in the tree to configure options for CVS, ClearCase, or a generic provider. Click **Ok**.

Browsing test servers and working with profiles

When a project is open, the Server Explorer tab  appears in the Explorer pane. It shows a treeview of the server nodes respective to the activated modules. The server nodes include Application, Database, and Test Servers.

The Test Servers node contains the test server profiles required to run the tests in the appropriate environment. On the Test Servers level, you can create new profiles and browse the running servers. On the profile level, you can edit and delete profiles.

To view the test server at a specified host:

1. Choose **Browse** on the right-click menu of the Test Server node. The Test Server Browser opens. The browser automatically scans the ports of the specified machine and displays the detected servers.
2. Stop or restart the scanning process as required.

To add a new profile:

1. Choose **Add** on the right-click menu of the Test Servers node. The New Profile dialog opens.
2. In the *Profile name* field, specify the name. Note that the default string is highlighted in red if such a name already exists and you cannot click OK until you specify a different name.
3. Choose *Remote Profile* or *Local Profile*:
 - For distributed testing on a remote server, choose *Remote Profile* and enter the HTTP address of the server you want to run the tests on.
 - To perform testing locally:
 1. Choose *Local Profile* and click the **Edit Local Profile** button. If you want to use the default main class from the project for this profile, check the box *Set default main class*. In this case, it is not necessary to edit the profile and you can skip the following steps if desired.
 2. In the Arguments and Parameters dialog, specify the main class to be tested in the *Select class with 'main'* field. To set or change the main class, click the browse button next to the field. If the box *Set default main class* was checked in the New Profile dialog, the default main class is already selected for you. If it was not checked, the *Select class with 'main'* field is empty and you must specify a class.
 3. Set *Program arguments* and *VM options* as desired.

4. Click **OK** to return to the New Profile dialog.
4. Click **OK** to create the profile.

To set the profile for an existing test step or suite, open its Inspector (right click | Properties) and choose from the available profiles in the *Test server profile* property.

When you create a new test step or suite under the Test Plan, it uses one of the existing test server profiles by default. If necessary, you can change the defaults.

To change the default profiles used for new test steps and suites:

1. Go to the Server Explorer tab and expand the Test Servers node.
2. Right click the test server profile you want to set as a default and choose **Set Default For**.
3. From the submenu, choose which type of test to set this profile as the default for.
 - **Suite** sets the selected profile for any new test suite (New | Suite in the test plan).
 - **JUnit** sets the selected profile for any new default JUnit test step (New | JUnit Step | Default TestCase or Default TestSuite in the test plan).
 - **Visual** sets the selected profile for any new visual test step (New | Visual Step | Record or Empty Step in the test plan).

Note Only one profile can be set as the default for any given type of test. If you reset the default to a new profile, the previous setting is automatically turned off.

Developing unit tests for source code

The information in this section assumes you have completed the instructions for [“Creating a test project” on page 472](#). 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 483](#)
- [“Using templates to create test cases” on page 483](#)
- [“Testing the default ClassLoader of an application” on page 485](#)

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:

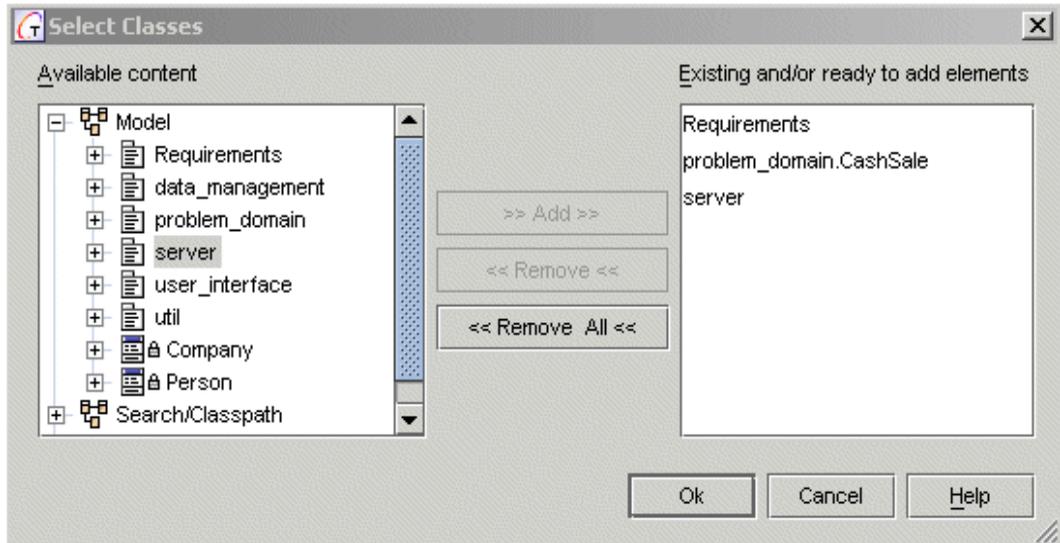
1. From the Tools menu, choose **Tools | Options | Project Level**.
2. From the tree-view of Project Options, expand **Testing Framework | Building Unit Tests | Test_Family_Name | Default TestCase**.
3. Select **Package Filters** and configure options as appropriate.
4. Select **Class Filters** and configure 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 (*.tpr). See [“Creating a test project” on page 472](#) 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 135](#).
6. Click **OK**.

Figure 135 Select Classes dialog



Writing test cases

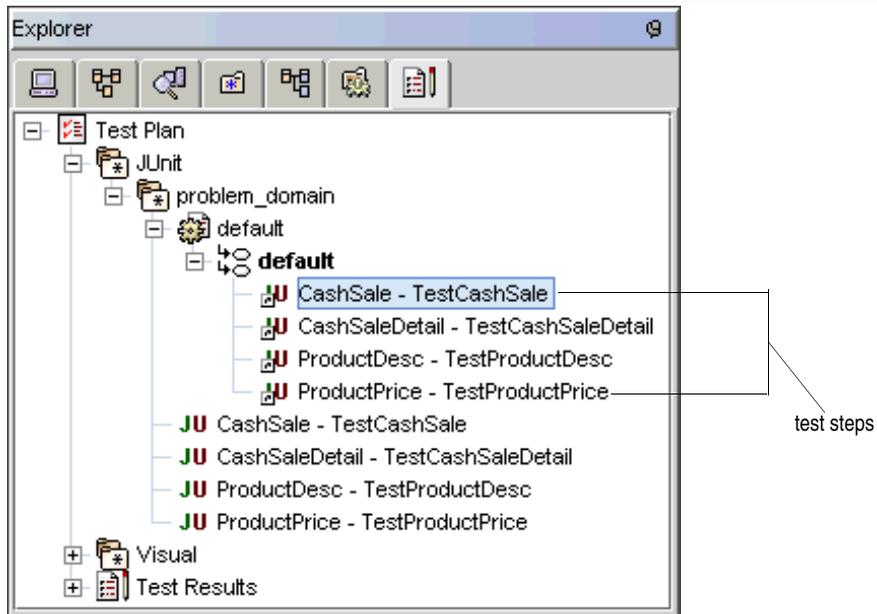
The default behavior of a generated test stub is for each method test to fail. This code must be replaced by code implementing the actual, intended method test.

These instructions assume that you have followed the steps for [“Setting up filter options for the test builder”](#) on page 480 and [“Generating stubs for test cases”](#) on page 480.

To write test cases, edit the code by following these steps:

1. Open a test project (*.tpr).
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. See [Figure 136](#) for examples of test steps.

Figure 136 Sample test steps



4. Using the Editor pane, replace each test method body with the desired test code.

Tip To view the properties of a test step, right-click on a test step and choose **Properties**. The inspector for the selected test step appears showing the **Test Step Properties** tab. This tab indicates the name of the test step. In addition, you can enter a one-line description for the test step in the *fulfills* field. The description does not affect the text execution.

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 481.

To create a test suite, follow these steps:

1. Open a test project (*.tpr).
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**.

Tip To view the test suite as an activity diagram, right-click the test suite node in the test plan and choose **Show Activity Diagram**. You can edit the activity diagram for the test suite, and your changes are reflected in the test suite on the Tests tab.

Importing JUnit tests

To import a JUnit test, follow these steps:

1. Open a test project (*.tpr).
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 (*.tpr).
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 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\togethersoft\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 51](#).

Table 51 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 51](#):

```
<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\togethersoft\modules\testingframework\plugin / \testbuilder\config\unittestbuilderconfig.xml`.
3. The XML editor tab  appears in the Explorer pane. Click the tab to view the file.

For instructions on using the XML editor, see [“Integrated XML Editor” on page 259](#).

Testing the default ClassLoader of an application

You can create JUnit tests that test the default ClassLoader of an application (for example, the classes initialized during the loading stage). To use this feature, specify the property `-Dtestserver.defaultclassloader=true` in the Java virtual machine (JVM) arguments.

Important If you change the classes of an application, you need to restart the test server. Otherwise, application classes will not reload.

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

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  icon. You may need to expand a test collection and test suite in order to locate a JUnit test.

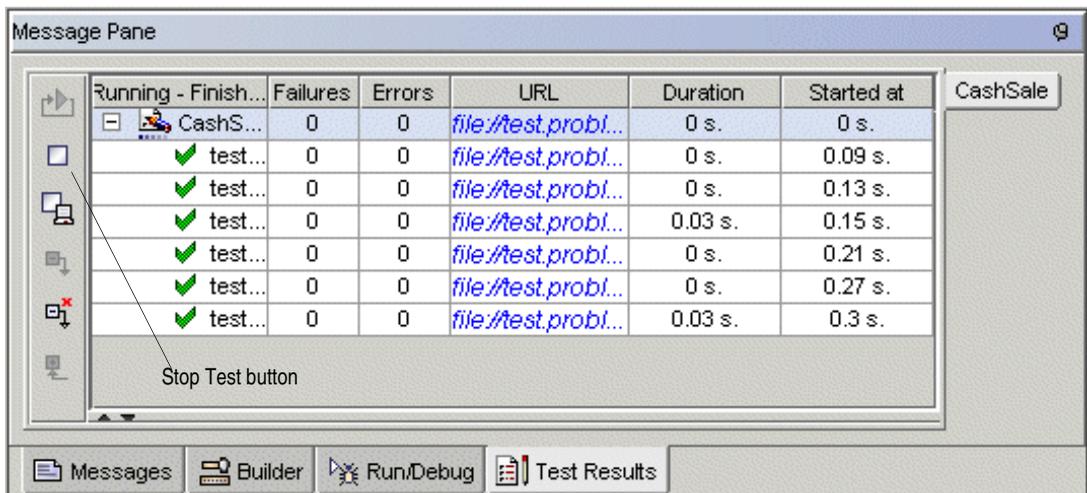
3. Choose **Run Test** from the right-click menu.
4. The results of the test appear in the Message pane.

Tip You can also run a test suite or collection of tests using the right-click menu. Right-click on either a test suite or test collection and then choose **Run Suite** from the menu.

Stopping the test server

The toolbar of the Test Results pane includes a **Stop Test** button as shown in [Figure 137](#). Use the **Stop Test** button to safely stop the test server, ending the currently running test.

Figure 137 Stopping a test



Regenerating test cases

After you edit or delete a test step (and package) of a JUnit test case, you can regenerate the test case by right-clicking the test case and choosing **Regenerate**.

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 472.

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 132, “Tests tab showing a test plan” on page 472](#) illustrates the organization of unit test collections and visual test collections.

To create a test collection, follow these steps:

1. Open a test project (*.tpr).
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.

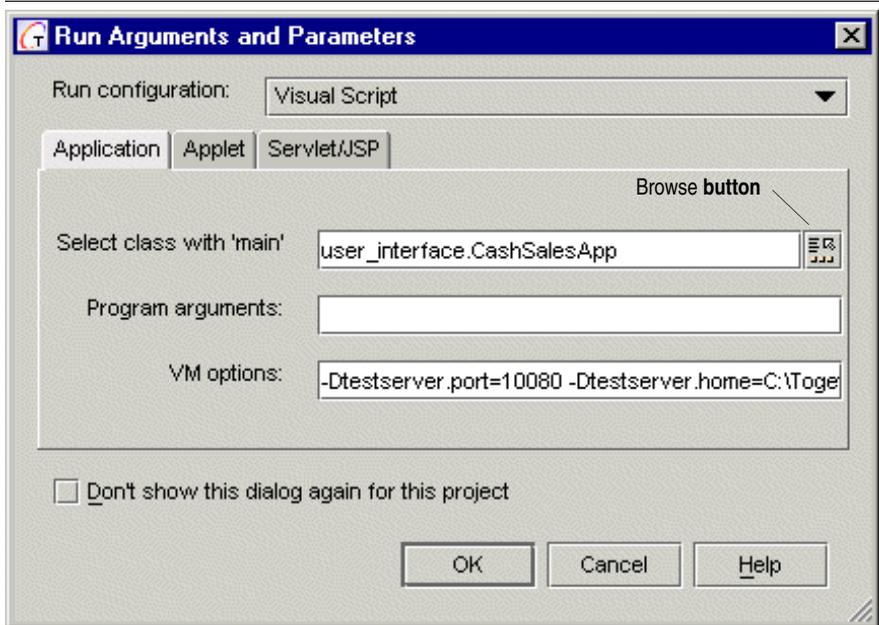
Recording a visual script

The testing framework includes a recorder (as shown in [Figure 139](#)) 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 489](#).

To record a visual script, follow these steps:

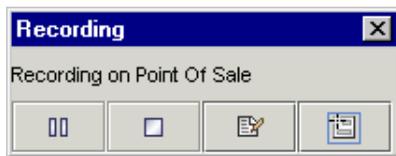
1. Open a test project (*.tpr). If you have not created a test project, see [“Creating a test project” on page 472](#).
2. From the Explorer pane, select the **Tests** tab.
3. From the tree-view, right-click on **Test Plan** and select **New | Visual Step | Record**.
4. From the Run menu, choose **Run** to invoke the Run Arguments and Parameters dialog as shown in [Figure 138](#). The default shown for *Run configuration* is *Visual Script*.
5. Click on the browse button to bring up the Select main class dialog.

Figure 138 Run Arguments and Parameters dialog



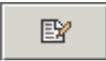
6. From the Select main class dialog, select the application or applet to run and then click **OK**.
7. From the Run Arguments and Parameters dialog, click **OK** to invoke the Recording window as shown in [Figure 139](#).

Figure 139 Recording window



8. 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:

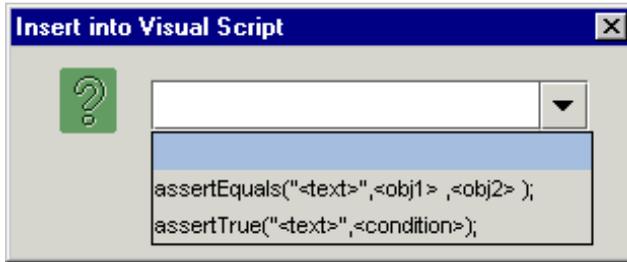
1. Click the assert button  on the Recording window.
2. From the Insert into Visual Script dialog shown in [Figure 140](#), select a pre-defined assertion statement or enter a comment to be placed in the script.

If necessary, edit inserted statements and comments by using the instructions in [“Editing a visual test” on page 490](#). For more information about assertions, see [“Creating an assertion” on page 493](#).

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 140 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.

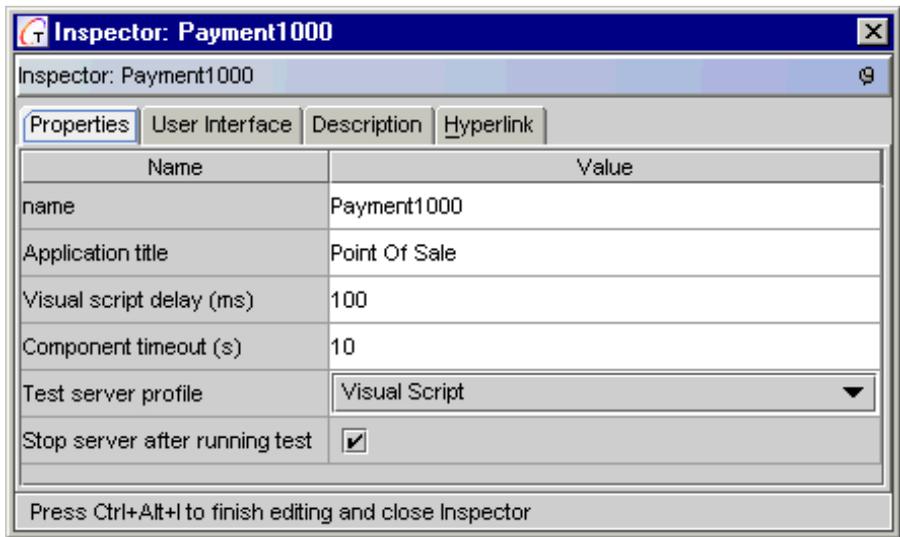
2. Use the cross-hair to select a component.
3. Click **OK**.
4. Click the stop button  on the Recording window.

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 141](#).
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 141 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.

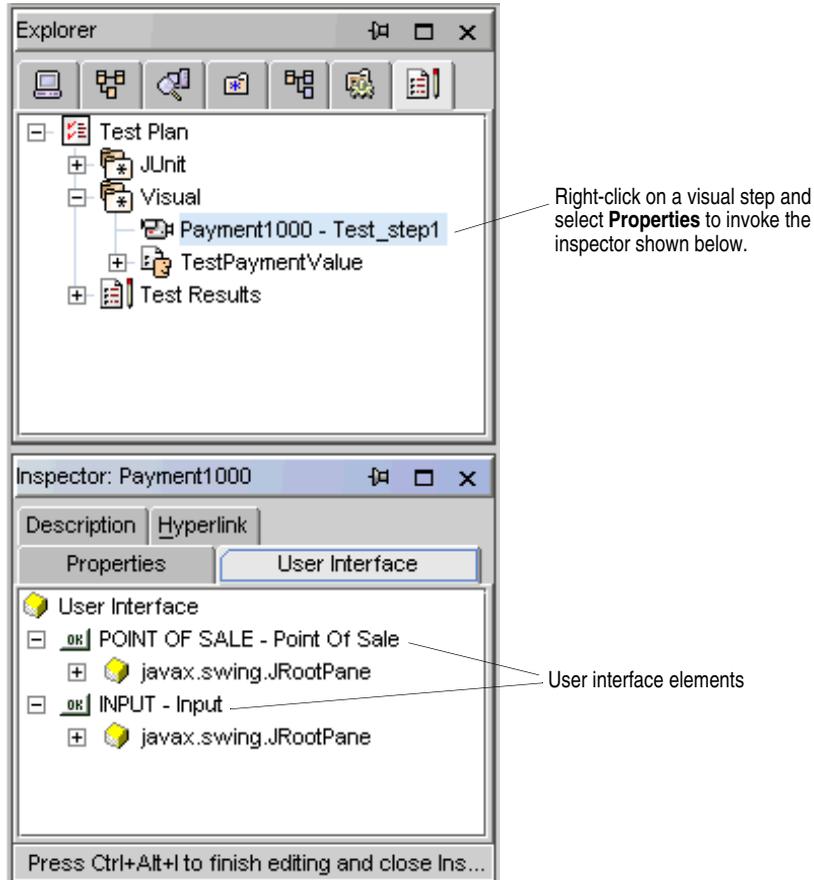
To change the user interface elements referenced by a visual step, follow these steps:

Note The instructions that follow use the XML editor and require that you activate XML support. See [“Using the XML editor” on page 485](#) 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 142](#), 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.

- Using the XML editor, you can change the values for `<tag>` elements or add your own `<tag>` elements under the `<ui_element>` element.

Figure 142 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

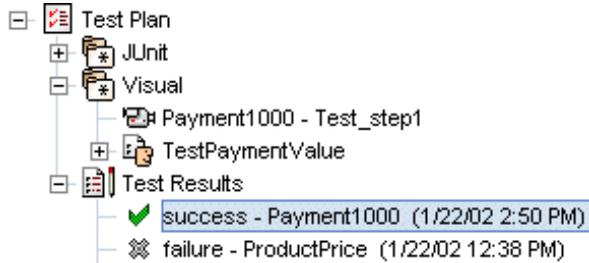
To run a visual test, follow these steps:

- From the Explorer pane, select the **Tests** tab.
- Expand **Test Plan | Visual**.

3. Right-click on a visual test, as marked by the movie-camera  icon and select **Play Script**. The Run Arguments and Parameters dialog appears.
4. Enter information for *Select class with 'main'*. The test executes. When finished, Test Results should indicate a successful status, as shown in [Figure 143](#) for the Payment1000 sample test.

Note When a visual script is played for the first time, the run configuration dialog, Run Arguments and Parameters appears.

Figure 143 Results for visual test indicating success

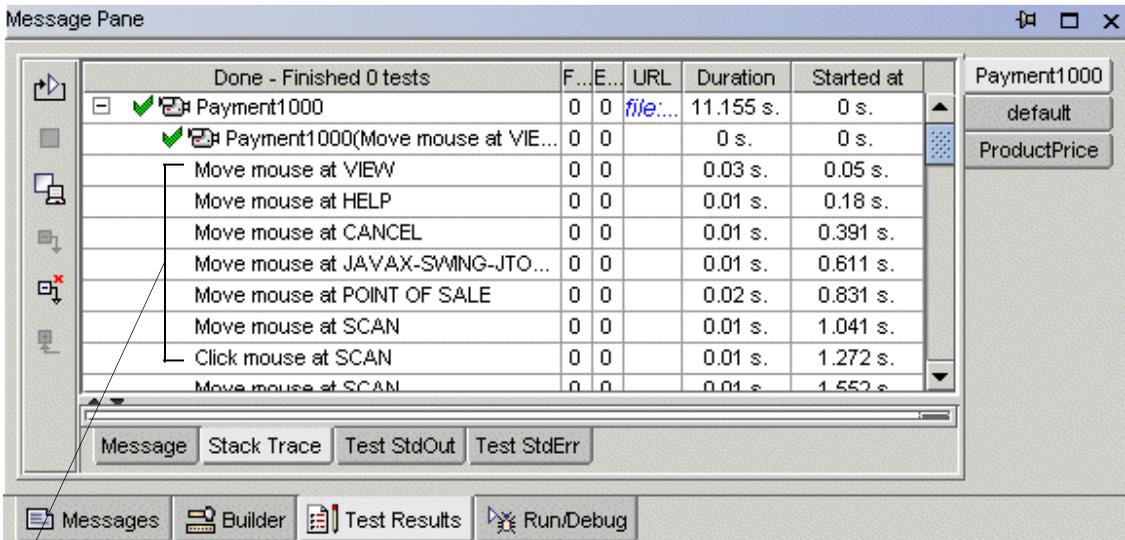


In addition, note that after the test completes, the Message Pane lists each step that comprises the visual test, and provides test results for each step, as shown in [Figure 144](#).

To safely end a visual test that is running, see [“Stopping the test server”](#) on page 486.

Tip You can also run a test suite or collection of tests using the right-click menu. Right-click on either a test suite or test collection and then choose **Run Suite** from the menu.

Figure 144 Results for each step of the visual test, as shown in the Message pane



Steps for the visual test script.

Creating an assertion

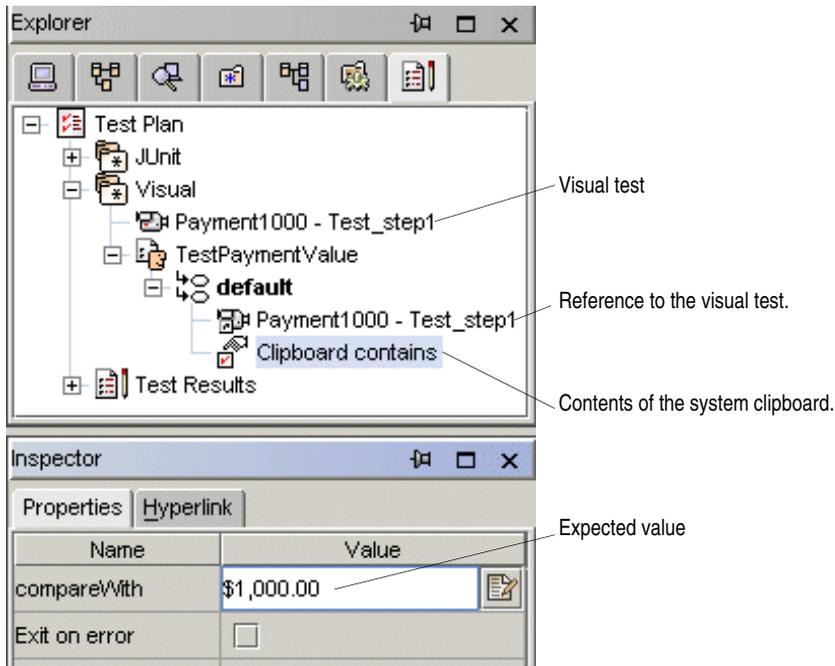
Together enables you to use four types of assertions:

- 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 145](#) 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 487](#).

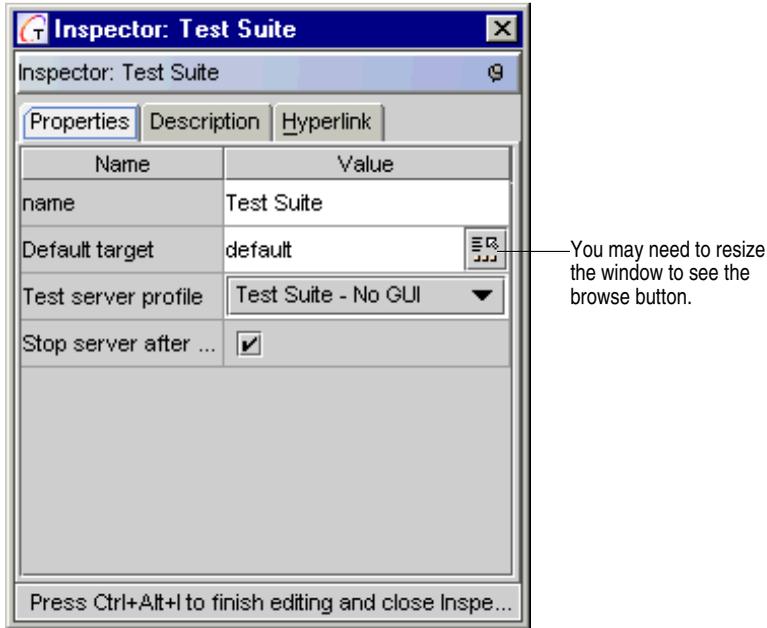
Figure 145 CashSales test suite showing clipboard and visual script



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 146](#).
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 146 Using the test suite inspector to set a default target



Viewing test results

The testing framework provides several formats for viewing test results: the results table in the Message Pane, an HTML report, or XML format.

Working with test results in the Message pane table

When a test is completed, the Test Results tab automatically opens in the Message pane and displays the test steps and results in a table.

- The first column in the table lists the test assets hierarchically. Each test step displays either a green checkmark if the test passed or a grey X if the test failed or produced an error.
- The *Failures* and *Errors* columns summarize the quantity of test steps failed and test steps with errors.
- The *URL* column displays a link to the corresponding test step file.
- The *Duration* and *Started at* columns display the length of each test step and the point at which each started.

To expand or collapse the nodes in the table, use one of the following methods:

- Right click any table row and choose **Expand/Collapse Branch** from the right-click menu. This expands or collapses the current branch if it has children, or the nearest branch up in the hierarchy if it does not.
- Click the plus (+) or minus (-) buttons in table rows that have children.
- Use the toolbar buttons to the left of the results table: *Expand Branch*, *Collapse Branch*, and *Expand all failure/error children*.

Tip You can sort the test results based on the data in the *Failures*, *Errors*, *Duration*, or *Started at* columns. To sort results according to the data in a column, click the column heading. To reverse the sorting order (ascending / descending), click the column heading again.

Navigating from test results to diagrams, test plan, or code

You can navigate from test results to the corresponding test step in the Designer, Explorer, or Editor pane.

To navigate from a test step in the test results table to the corresponding diagram:

Right click a row in the table and choose **Navigate to Diagram** from the right-click menu. This highlights the test step in the Designer pane.

To navigate from a test step in the test results table to the corresponding node in the test plan:

Right click a row in the table and choose **Navigate to Test Plan** from the right-click menu. This highlights the test step under the Test Plan on the Tests tab in the Explorer.

To navigate from a test step in the test results table to the corresponding code in the Editor:

Select the row in the table and click the link in the *URL* column. This automatically opens the file in the Editor pane and navigates to the corresponding code.

Generating a report in HTML

After running tests, you can generate a report showing the test results.

To generate a report, follow these steps:

1. From the Explorer pane, select the **Tests** tab.
2. Under **Test Plan**, expand **Test Results** and select a test result.
3. Right click the test result node and choose **Generate HTML** from the right-click menu. The report opens 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**.

Viewing test results in XML format

After running tests, you can view the XML version of test results.

To open the test results in XML format:

1. On the Tests tab in the Explorer, navigate to Test Plan - Test Results.
2. Right click a test results node and choose **Edit Test Asset** from the right-click menu.

An XML file containing the test results opens in the Editor. The same results are outlined on the XML Structure tab in the Explorer. You can edit the XML to add comments or other modifications.

[Table 52](#) lists some XML elements particular to the XML results files, along with their meanings.

Table 52 XML elements in results files

result	Top level element describing the test result.
step_reference	Test step executed.
suite_reference	Test suite executed.
detail	Provides details of execution.
stacktrace	Contains stack trace data.
stderr	Contains standard error stream.
stdout	Contains standard output stream.
comment	Contains any commentary provided by the user.
error	Signals a test error.
failure	Signals a failed test.

Distributed testing

To support testing in a distributed environment, the testing framework provides HTTP access to test assets and test results. Each test asset in the test plan is a separate resource that can be accessed by a URL. When you run tests, they are run according to the test server profile set in the properties (see [“Browsing test servers and working with profiles”](#) on page 478).

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/togethersoft/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:

- 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

Installing the bootstrap loader

To install the bootstrap loader:

1. Confirm that you have a copy of JDK 1.3 or higher installed on your system.
2. Navigate to your JDK home directory. If you have previously installed any version of testing extensions, remove the files `%JAVA_HOME%/jre/lib/ext/testingbootstrap.jar` and `%JAVA_HOME%/jre/lib/accessibility.properties`.
3. Navigate to the testing extension directory in your Together installation:
`%TG_HOME%/modules/com/togethersoft/modules/testingframework/lib/ext/`
4. Copy `testingbootstrap.jar` from the testing extension directory to `%JAVA_HOME%/jre/lib/ext`

5. Copy `accessibility.properties` from the testing extension directory to `%JAVA_HOME%/jre/lib`

Note On Mac OS X systems, the JDK home directory is slightly different (the `/jre/` directory does not exist). The correct Mac OS X file locations are:

```
%JAVA_HOME%/lib/ext/testingbootstrap.jar
%JAVA_HOME%/lib/accessibility.properties
```

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 / \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 preceding option 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:

1. Modify your application batch file to pass the 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 that 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

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 identifying a test server configuration (possible values are listed under [“Setting run configuration options” on page 499](#)).
5. Set the variable `TEST_SERVER_PORT` to a 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 the test server started successfully by pointing your browser to `http://localhost:<testserver.port>/tf/`.

Troubleshooting

Table 53 lists conditions to check when troubleshooting your test server.

Table 53 Troubleshooting your test server

If this occurs...	...then check this condition
If the test server exits with an error message	<ul style="list-style-type: none">• 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>junit.jar</code>, <code>junitx.jar</code>, <code>ant.jar</code> etc.) exist in the locations below or on your classpath:<ul style="list-style-type: none">• <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
An internal error message occurs when running test results	All classes required for running tests are present in 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	<ul style="list-style-type: none">• 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 498 .

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 503
- “Custom code templates” on page 508
- “Patterns” on page 513
- “Using templates and patterns” on page 517

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. 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. [Table 54](#) presents the properties available.

Table 54 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 attributes and return types of operations.
hideInChooseList	If this property is present in the <code>*.properties</code> 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.

Table 54 Code Template properties (continued)

Property	Description
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 55 Template types and corresponding macros

Template type	Macros handled
Class	<code>%Name%</code> , <code>%Class_Name%</code>
Member	<code>%Name%</code> , <code>%Type%</code> , <code>%Class_Name%</code>
Link	<code>%Name%</code> , <code>%Type%</code> , <code>%Dst%</code>

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” on page 507](#) below.)

Important Never edit the default templates. Improper modifications can result in incorrect functioning of Together, and make it impossible to create diagram elements.

Browsing available code templates

You can browse through the available code templates using the Directory tab of the Explorer. You do not need an open project. The categories correspond to different programming languages.

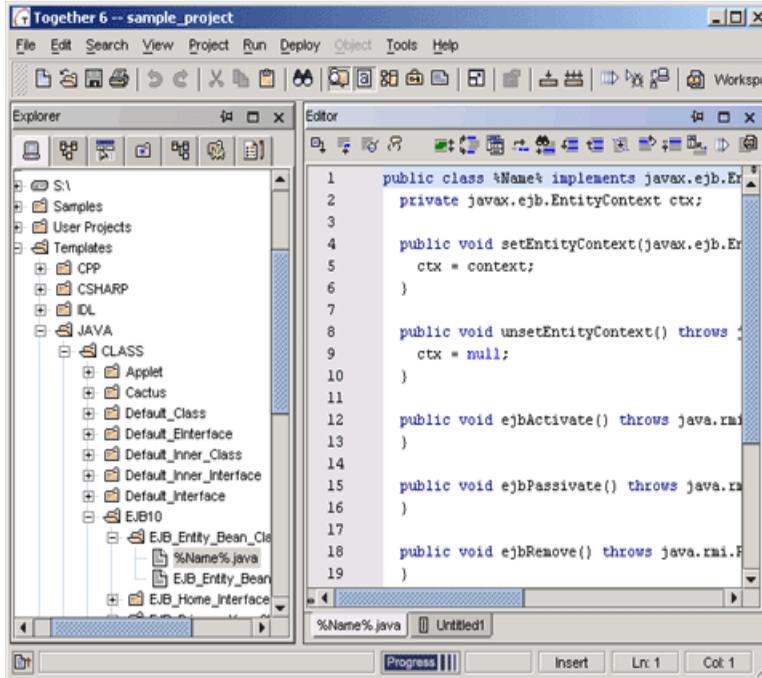
To see the main template categories:

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.

Note that while all templates are available to view and edit, not all templates function in all products. For example, Java templates do not work in products that support only C++. Functionality depends upon your product type.

Each main category has subcategories for class templates, link templates, and member templates. The figure below shows the structure of the *Templates* node on 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 in [Figure 147](#) you could add default declarations for finder or business methods.

Figure 147 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 code 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 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 category 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 convenient way to create custom code templates using the Code Template Expert. Open the expert from the Tools menu, or from the right-click menu of a diagram object. The expert creates template and properties files according to the rules described above.

In the Code Template 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” on page 518](#) and [“Creating members by pattern” on page 519](#) 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 modified, 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.

Important

The Code Template 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 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 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%.java` 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 be displayed 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 is displayed 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, you 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 is still displayed 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 is displayed in the Pattern Chooser.

Example

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 are 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 with custom file extensions

Together makes it possible to generate files with custom file extensions, using appropriate templates. This requires certain spade-work to be done.

Defining the source for extension of the newly generated files

When a class is generated by template, its extension can be taken from the two possible sources:

- textual templates extension, defined in `TGH/config/resource.config`,
- or
- codegen default extension, defined in `TGH/config/<language>.config`

By default Together uses the codegen default extension, but you can make Together use the template extension instead.

To use a textual template extension for the generated file:

1. Open the file `TGH/config/<language>.config` for the appropriate language.
2. In the section Codegen options, uncomment the following line:

```
;codegen.java.use_template_extensions = yes
```

Defining the custom template extension

To define the template extension:

1. Open the file `TGH/config/resource.config`.
2. Add the following lines to the appropriate language section:

```
resource.file.NewExtension.extension = "NewExtension"
resource.file.NewExtension.name = ["filetype/java.source"]
resource.file.NewExtension.type = "source"
resource.file.NewExtension.language = "java"
resource.file.java_source.extension.2 = "NewExtension"
```

Creating files with custom template extensions

When the source of the extension and its text are defined, you can create a new template and generate files with new extensions using this template.

To create a file with custom template extensions:

1. Start Together, and choose **Tools | Code Template Expert** command on the main menu
2. Create a new class template, as described in the section [“Custom code templates” on page 508](#).
3. In the Together installation, find the newly created template file, and change its extension as required:

```
For example, change $TGH$/templates/JAVA/CLASS/<new template>/%Name.java
to TGH$/templates/JAVA/CLASS/<new template>/%Name.NewExtension
```

4. In Together, click **Class by Pattern** button on the class diagram toolbar, and choose the new template with custom extension. The class with the specified extension is generated.

Patterns

Together delivers a unique capability to extend its functionality externally by using modules and patterns. In this topic you will learn about the basic concept of patterns, how to use the “pre-fab” patterns delivered with Together, and how to deploy your own patterns. Modules are discussed in [Chapter 45, “Together Open API”](#).

What are 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, and Singleton, a set of EJB classes for various specifications, a set of TagLib patterns, and many others.

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” on page 515. You will also need to study the documentation for the Together API. (For more information on the API documentation, see Chapter 45, “Together Open API”.) You can study examples of mature patterns in the `TGH\modules\com\togethersoft\modules\patterns` directory in your installation and you can also obtain the sources of patterns at the Together Community web site:

<http://community.togethersoft.com/>.

Working with various languages

Some patterns work only with a specific language (for example, the enterprise JavaBean pattern works only with the Java language), and some can be applied to any language (for example,

`com.togethersoft.modules.patterns.UNIVERSAL.MEMBER.Stub_implementation`).

Together determines the target language for a pattern automatically, based on its location in the installation. The package of a pattern defines the target language for the pattern (or elements produced by the pattern).

Table 56 Languages and corresponding patterns

Language	Package
Java	<code>com.togethersoft.modules.patterns.JAVA</code>
C++	<code>com.togethersoft.modules.patterns.Cpp</code>
All languages	<code>com.togethersoft.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 57 Elements generated by patterns

Element	Package
Class	<code>com.togethersoft.modules.patterns.JAVA.CLASS</code> , <code>com.togethersoft.modules.patterns.Cpp.CLASS</code> , <code>com.togethersoft.modules.patterns.UNIVERSAL.CLASS</code>
Link	<code>com.togethersoft.modules.patterns.JAVA.LINK</code> , <code>com.togethersoft.modules.patterns.UNIVERSAL.LINK</code>

Table 57 Elements generated by patterns

Element	Package
Member	com.togethersoft.modules.patterns.JAVA.MEMBER, com.togethersoft.modules.patterns.UNIVERSAL.MEMBER

Pattern behavior

Behavior of a pattern is defined by its properties set. The `SciPattern` interface defines the methods that you should implement in your pattern.

Table 58 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 or objects with the current values of the pattern properties.
<code>prepare</code>	Checks if it is possible to apply this pattern to the target object or 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 or 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 the package

```
com.togethersoft.modules.patterns.JAVA.CLASS.foo_Pattern
```

will be displayed in a separate group called “foo Pattern” in the class pattern chooser dialog (Together automatically replaces an underscore sign with a space).

The following example shows how to develop a simple pattern step by step, using Together:

1. Create the folder

```
foo_Pattern
```

under `com.togethersoft.modules.patterns.JAVA.CLASS`.

This pattern will work with Java classes only.

2. Create a Together project and name the project `foo_Pattern` (for example, `myprojects/foo_pattern`).

3. In the Project Properties dialog, choose the *Project Paths* tab, highlight the project path and enter the following prefix in the Package prefix field:

```
com.togethersoft.modules.patterns.JAVA.CLASS.foo_Pattern
```

Include `TGH\lib\together.jar` and `TGH\lib\openapi.jar` in the Search/Classpath. Alternatively, check the *Include Classpath* option.

4. Add a new class to the default diagram, naming it `Pattern.java`.
5. Add the following code in the Editor:

```
package com.togethersoft.modules.patterns.JAVA.CLASS.foo_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();
}
```

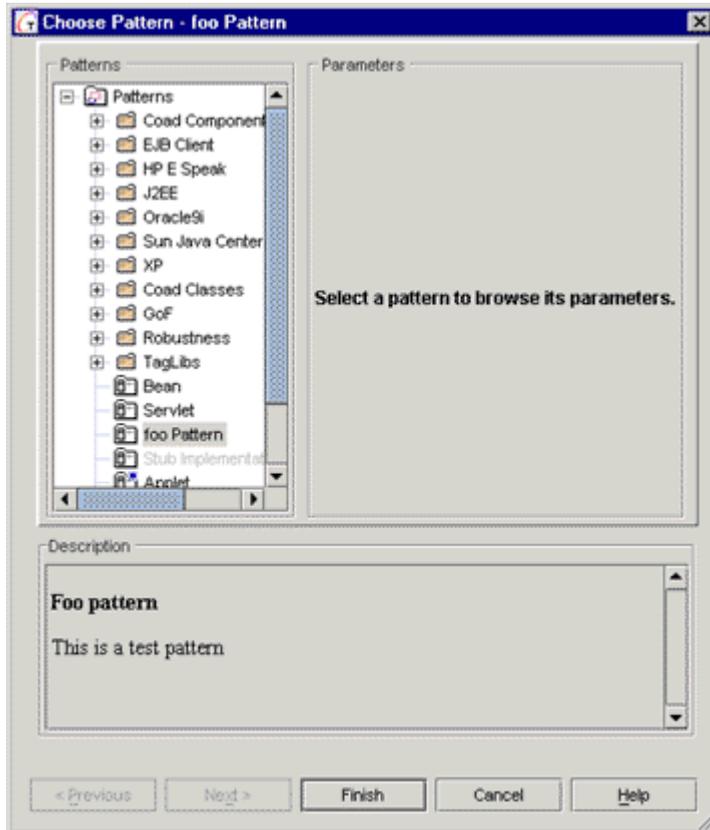
6. Adjust the compiler output on the project level:

- On the main menu, choose **Tools | Options | Project Options**.
- Navigate to *Builder - Built-in Javac - Compiler options*.
- In the *Destination directory* field, change the default destination folder to `TGH\modules\com\togethersoft\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 specified destination folder.

The Choose Pattern dialog displays the pattern. If you click the Class by Pattern button, the tree-view of available patterns and templates displays the new pattern as shown in [Figure 148](#).

Figure 148 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 descriptions

To create a description for your pattern, you need to place a file named `Description.html` in the directory with your pattern. The contents of this file are 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 chooser are marked with an asterisk, to distinguish 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 binary patterns, the preview is not available. You can find a 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 buttons in [Table 59](#) on the class diagram toolbar.

Table 59 Toolbar buttons that use templates and patterns

Tool-tip	Icon	Description
Class by Pattern		Creates a new class using selected pattern to define how code is generated.
Link by Pattern		Creates a relationship link using selected pattern to define how code is generated.

Both buttons launch the Choose Pattern dialog, displaying the available patterns for the respective operation. You can access Together predefined patterns, plus any that you implement yourself.

To create a class or link by pattern:

1. Click the appropriate toolbar button.
2. If creating a class, click on the Designer pane to open the Choose Pattern dialog for classes. If creating a link, drag it from the source class to the destination and drop it to open 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 If you are using the Microsoft JVM, note that `java.rmi` classes, required for compilation of the EJB pattern, are not present in this JVM.

Important 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 Designer 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 the Choose Pattern command from the class right-click menu enables refactoring of the class according to the selected pattern.

Creating a stub implementation

When creating an inheritance link between a class and another class or interface, the methods and members are not automatically added to the child class. This problem is solved using the Stub Implementation pattern.

If the destination of a link is an interface, the pattern makes the class-source implement that interface, and creates in a class the stubs for all the methods found in the interface and all its parent interfaces.

If the destination of a link is another class, then this pattern makes the class-source extend the class-destination, and makes stubs for all the constructors found in the class-destination. These constructor stubs simply call the corresponding constructors in the class-destination.

It is also possible to copy Javadoc comments of the inherited methods by checking the *Copy documentation* box in the pattern dialog.

To create an inheritance link with stub implementation:

1. On the diagram toolbar, click the Link by Pattern button.
2. Click on the source class and drag-and-drop the link to the destination class or interface.
3. In the Choose Pattern dialog for links, select *Stub Implementation*.
4. Check the *Stub constructors* and *Copy documentation* boxes, if necessary.
5. Click **Finish** to complete. The inheritance link is created, and the stubs for the inherited methods are generated in the source class.

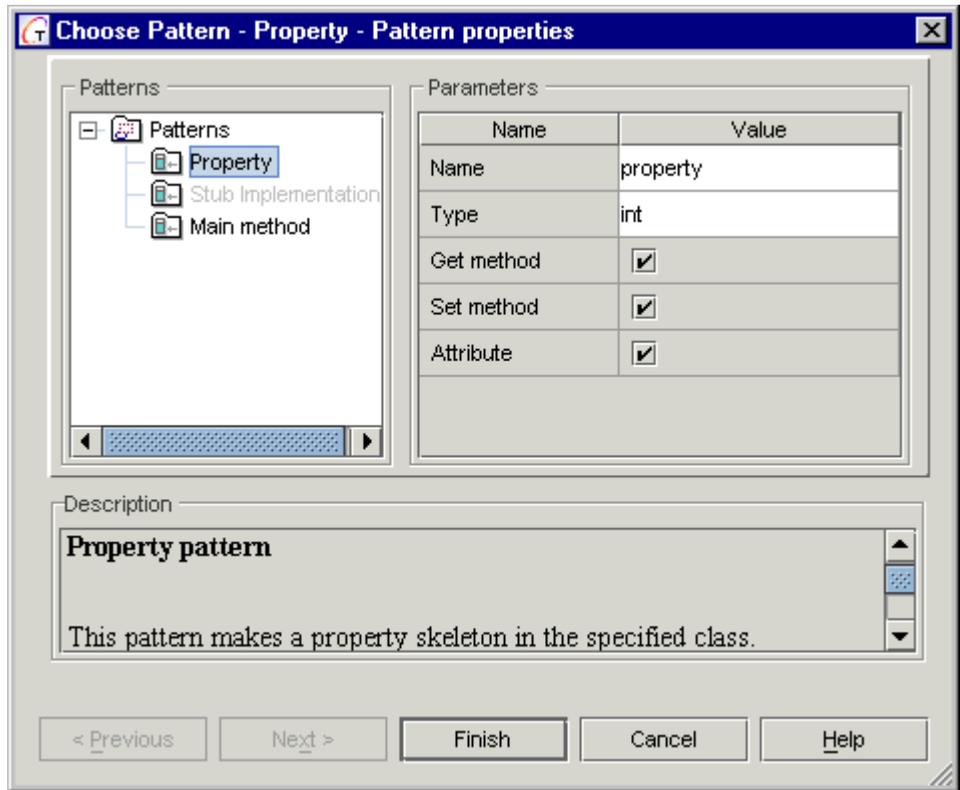
Creating members by pattern

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 149 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. 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 opens, displaying the available patterns for members.
3. Select the pattern you want for the member.
4. Click the **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 perform this singly, or to multiple selected elements.

To refactor classes or links:

1. Select the elements in the Designer 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.

Audits and Metrics

This chapter includes the following topics:

- “Overview of audits and metrics analysis” on page 523
- “Running audits and metrics in Together” on page 525
- “Language-specific metrics and audits” on page 539
- “Running audits and metrics from the command line” on page 547
- “Extending the QA module” on page 549
- “Additional information sources” on page 550

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 550 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 [Table 60](#).

Table 60 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 [Table 61](#).

Table 61 Rules of etiquette for applying software metrics

Management level	Rules
Functional management	<ol style="list-style-type: none">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.
Project management	<ol style="list-style-type: none">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	<ol style="list-style-type: none">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 535](#)). 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 feature 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 only 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 538](#).

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 539](#).

Some audits and metrics accompany specific Together features. Currently, these include audits and/or metrics that support the Real-Time and UI Builder features. These audits and metrics are available only if the corresponding feature 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 feature 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 feature 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.

Options for processing audits and metrics

The default processing settings are adequate for running audits and metrics. However, if you want to change the processing behavior, use the Options dialog (**Tools | Options | <level>**) to set options. The options are available in the QA node on the default and project levels.

The following options specify how to process audits and metrics:

- *Process in background* - This option is unchecked by default, because it is safer to run audits and metrics in the command thread. If you want to edit something while you are waiting for audits or metrics to be processed, check this option. However, use caution - never edit a package while it is being processed for audits or metrics (this leads to false results or exceptions in Together).
- *Process classes* - When running audits or metrics for an entire project, all the classes in the model are processed by default. It is possible to also process the classes specified in Search/Classpath in Project Properties. Check the options for the classes that you want to process: *Process classes in model*, and/or *Process classes from Search/Classpath*.

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 533).

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 529](#)).

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 530](#)).

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, as shown in [Figure 150](#).

- 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.

Figure 150 Comparing metrics results

29	51	486653	2325	779	95
10	5	6992	216	94	95
29	37	475529	2014	656	92
3	9	5270	108	74	
2	9	POSSFrame	HEff		
8	42	CashSales 1/31	475529		67
4	10	CashSales 2/28	475336		0
8	23	149094	857	300	21

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.

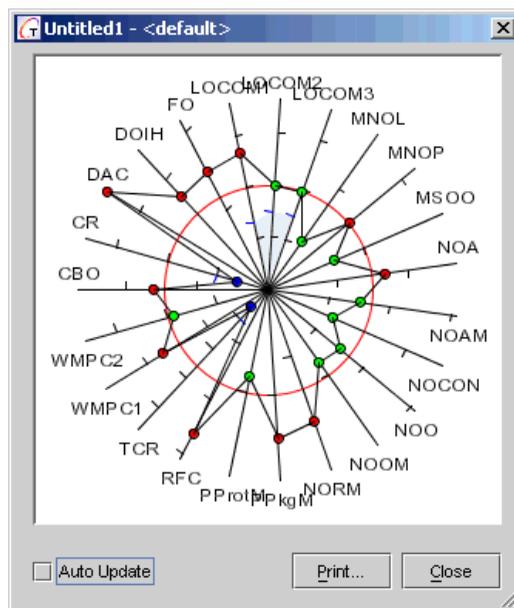
Kiviat graph

The Kiviat graph (see [Figure 151](#)) 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 upper limit values are equidistant from the center. In this way, limits and values are displayed using the following notation:

- Upper limits are represented by a red circle, showing that any points outside the red circle violate the upper limit.
- Lower limits are represented by blue shading, showing that any points inside the blue area violate the lower limit. Note that blue shading does not show up in areas of the graph with lower limits of 1 or 0.
- The actual metrics show up in the form of a star, with metric values drawn as points.
 - Green points represent acceptable values.
 - Blue points represent values below the lower limit.
 - Red points represent values exceeding the upper limit.
- Scale marks are displayed as clockwise directional ticks perpendicular to the Kiviati ray.
- Lower limit labels are displayed as counter-clockwise directional blue ticks perpendicular to the Kiviati ray.

Figure 151 Kiviati Graph



Bar graph

The bar graph (see [Figure 152](#)) 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.

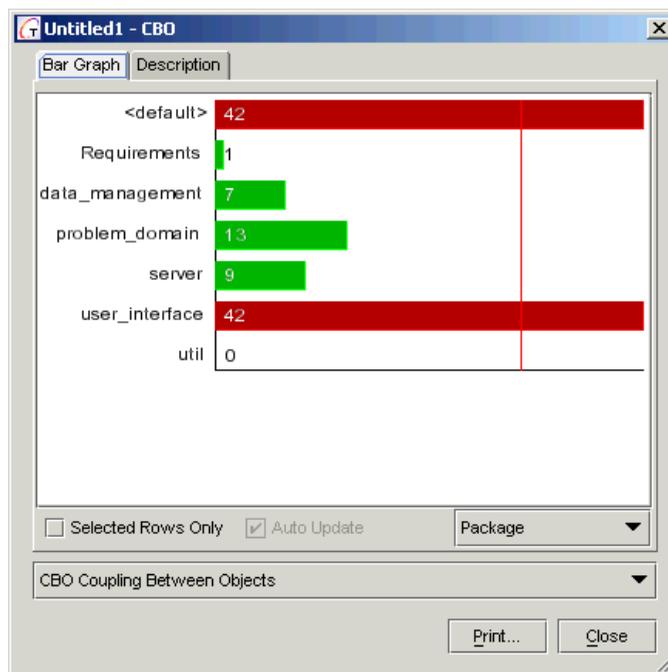
To change the metric to display results for, select from the drop-down list of metrics in the dialog box. You can refer to the explanation of the current metric on the Description tab.

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 and the corresponding element in the Designer pane. Double-click on a bar in the graph, or right-click and choose **Open**.

Figure 152 Bar Graph



Distribution graph

The distribution graph (see [Figure 153](#)) 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.

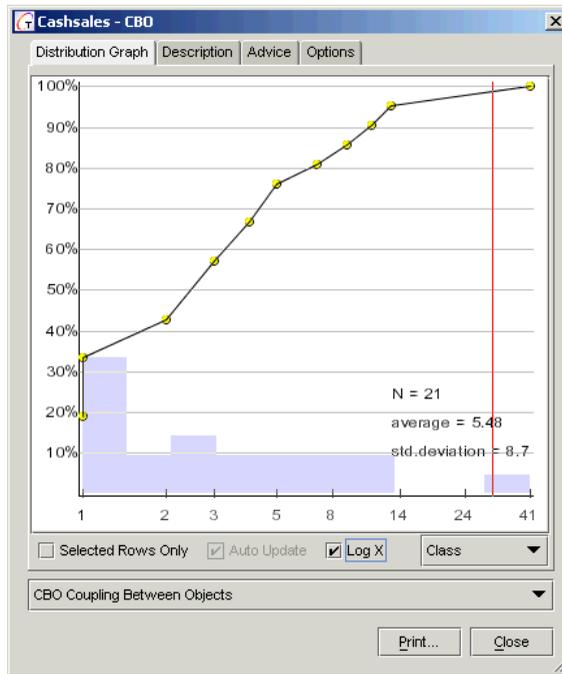
To view results only for the highlighted rows of the table (instead of all rows), check *Selected rows only*. To browse through the table and view the graph, check *Auto update*. If *Auto update* is unchecked, the graph displays the original selected rows and does not change if you select different rows 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 153 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.

[Table 62](#) lists the audits that currently provide automatic correction features.

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 62 Audits that have automatic correction features

Audit Group	Abbreviation	Audit Name
Declaration Style	CPASBF	Constant Private Attributes Should Be Final
Documentation	DCBA	Detect Collection-Based Associations
Possible Errors	CSF	Call super.finalize() from finalize()

Table 62 Audits that have automatic correction features (continued)

Audit Group	Abbreviation	Audit Name
Possible Errors	UL	Use 'L' Instead of '1' 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 525.
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 525](#).
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

The available audits and metrics depend on which language and other features (such as UI Builder) are activated in the current project. The Java feature supports a wide range of audits and metrics. Other languages have smaller sets of audits and metrics that have been adapted or created to fit the particular language.

Language-specific metrics and audits are listed in the following sections:

- [“Audits and metrics supported for Java” on page 539](#)
- [“Audits and metrics supported for C++” on page 544](#)
- [“Metrics supported for Visual Basic 6” on page 545](#)
- [“Metrics supported for C# and Visual Basic .NET” on page 546](#)

Audits and metrics supported for Java

Together provides over 100 audits and approximately 60 metrics for Java projects. [Table 63](#) lists the metrics supported for Java (which are also available for C++), and [Table 64](#) lists Java audits.

Table 63 Java 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 63 Java 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
MSSO	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
NOCF	Number of Controls in Form (UI Builder)*
NOIS	Number of Import Statements*
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
PPkgM	Percentage of Package Members*
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members

Table 63 Java Metrics (continued)

RFC	Response for Class
TAV	Total Audit Violations
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
WMPC2	Weighted Methods Per Class 2

Notes:

* This metric is specific to Java only and does not apply to C++.

Table 64 Java Audits

ACE	Assignments in Conditional Expressions
AFLV	Assignments to 'for' Loop Variables
AFP	Assignments to Formal Parameters
ASMDCN	Accessing Static Members by the Descendant Class Name
ASMO	Accessing Static Members through Objects
ASWL	Appending to String Within a Loop
BLAD	Bad Location for Array Declarators
BTJC	Bad Tag in Javadoc Comments
CA	Complex Assignments
CEV	Constants with Equal Values
CIUCFL	Complex Initialization or Update Clauses in 'for' Loops
CFPT	Comparing Floating Point Types
CLE	Complex Loop Expressions
CNSMFN	Class Name Should Match File Name
CPASBF	Constant Private Attributes Should Be Final (autofix)
CQS	Command Query Separation
CSF	Call super.finalize() from finalize() (autofix)
CVSBF	Constant Variables Should Be Final
DVIL	Declaring Variables Inside Loops
DVSS	Declaring Variables in Separate Statements
EBWB	Enclosing Body Within a Block
ECB	Empty Catch Blocks
EIAV	Explicitly Initialize All Variables

Table 64 Java Audits (continued)

EJB_CL	(EJB) Creating Class Loaders
EJB_CONSOLE	(EJB) Console Output
EJB_FDESCR	(EJB) Directly Reading or Writing File Descriptors
EJB_FILES	(EJB) Accessing Files and Directories
EJB_IO	(EJB) Using AWT, SWING, Other UI APIs
EJB_JDBC	(EJB) Using JDBC API from Session Beans
EJB_NATIVE	(EJB) Loading Native Libraries
EJB_REFL	(EJB) Using Reflection
EJB_SEC	(EJB) Obtaining Security Policy Information
EJB_SECOBJ	(EJB) Using Security Configuration Objects
EJB_SFACT	(EJB) Setting Socket Factory
EJB_SOCKET	(EJB) Listening on a Socket
EJB_SUBST	(EJB) Using Subclass and Object Substitution Features
EJB_THREADS	(EJB) Managing Threads
EOBA	Equality Operations on Boolean Arguments
HIA	Hiding Inherited Attributes
HISM	Hiding Inherited Static Methods
HN	Hiding Names
ILC	Import List Construction (autofix)
ICSBF	Instantiated Classes Should Be Final (autofix)
IMCM	Inaccessible Matching Constructors or Methods
LF	Long Files
LL	Long Lines
LPPMF	List Public and Package Members First
MLOWP	Mixing Logical Operators Without Parentheses
MSC	Multiple String Concatenations
MSOL	Multiple Statements on One Line
MVDWSN	Multiple Visible Declarations With Same Name
NC	Naming Conventions
NEC	Names of Exception Classes
NFSA	Non-Final Static Attributes
NLC	Numerical Literals in Code
NOIS	Negation Operator in 'if' Statement
ODCM	Order of Declaration of Class Members
OIM	Obsolete Interface Modifiers (autofix)
OM	Order of Modifiers
ONAMAM	Overriding a Non-Abstract Method with an Abstract Method

Table 64 Java Audits (continued)

OPM	Overriding a Private Method
OSNBU	Operator '?' Should Not Be Used
OVS	Overloading Within a Subclass
PCO	Parenthesize Conditional in Operator '?:'
PDBB	Place Declarations at Beginning of Blocks
PFC	Provide File Comments
PIFS	Provide Incremental in 'for' Statement or Use 'while' Statement
PJDC	Provide Javadoc Comments
PMFL	Place Main Function Last
POSNT	Place Operations with Same Name Together
PPA	Public and Package Attributes
PPCF	Place Public Class First
RIP	Referencing Implementation Packages
SAUI	Static Attribute Used for Initialization
SBCCS	Supply Break or Comment in Case Statement
SEB	Statements with Empty Body
SL	String Literals
SM	'synchronized' Modifier
SSSIDC	'switch' Statement Should Include a Default Case
DCBA	Detect Collection-Based Associations
TMSSC	Too Many Switch Statement Cases
UAAO	Use Abbreviated Assignment Operator
UC	Unnecessary Casts
UCVN	Use Conventional Variable Names
UE	Use 'equals' Instead of '=='
UIMM	Unnecessary Interface Member Modifiers (autofix)
UIOE	Unnecessary 'instanceof' Evaluations
UL	Use 'L' Instead of '1' at the End of Integer Constants (autofix)
ULVFP	Unused Local Variables and Formal Parameters
UPCM	Unused Private Class Members (autofix)
URSP	Unnecessary Return Statement Parentheses
UTE	Use 'this' Explicitly to Access Class Members
UI_CM	(UI Builder) Conflicting Mnemonics
UI_DM	(UI Builder) Disabling Menus when Items are Unavailable
UI_DCMI	(UI Builder) Duplicate Contextual Menu Items
UI_FSSC	(UI Builder) Font Size Specified in Code
UI_MDB	(UI Builder) Mnemonics for Default Buttons

Table 64 Java Audits (continued)

UI_MMT	(UI Builder) Multi-word Menu Titles
UI_SLS	(UI Builder) Second Level of Submenus
UI_UCM	(UI Builder) Use Common Mnemonics
UI_UCMO	(UI Builder) Use Common Menu Order
UI_UFL	(UI Builder) Use Flexible Layout
UI_UHC	(UI Builder) Use Headline Capitalization
UI_UM	(UI Builder) Unassigned Mnemonics
UI_USC	(UI Builder) Use Sentence Capitalization

Audits and metrics supported for C++

C++ support includes an extensive set of audits and metrics that have been adapted for C++, as well as several audits that are specific to C++ and are not supported for any other languages.

The metrics supported for C++ are the same as those listed in [Table 63, “Java Metrics” on page 539](#) with the exception that the following metrics do not apply to C++: Number of Controls in Form, Number of Import Statements, and Percentage of Package Members.

[Table 65](#) lists the audits currently supported for C++.

Table 65 C++ Audits

AUVK	Always Use “Virtual” Keyword (autofix)
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

Table 65 C++ Audits (continued)

MVDWSN	Multiple Visible Declarations With Same Name
NC	Naming Conventions
ONAMAM	Overriding a Non-Abstract Method with an Abstract Method
RO	Related Operators
OVS	Overloading with a Subclass
PPA	Public and Package Attributes
RINF	Redefining an Inherited Nonvirtual Function
UVBC	Using Virtual Base Class
UVD	Use Virtual Destructor (autofix)

Metrics supported for Visual Basic 6

Only declaration-based metrics are available for Visual Basic 6. Metrics related to inheritance and polymorphism are not provided because Visual Basic 6 lacks inheritance. [Table 66](#) lists the metrics currently available for Visual Basic 6.

Table 66 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

Metrics supported for C# and Visual Basic .NET

Only declaration-based metrics are available for C# and Visual Basic .NET. [Table 67](#) lists the metrics currently available for C#. The sets of metrics that have been adapted for C# and Visual Basic .NET are similar to each other. The differences in Visual Basic .NET are:

- 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).

Table 67 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
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

Table 67 Metrics supported for C# (continued)

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
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.

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 features 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.sh
%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 68 lists the options to use for running audits and metrics from the command line.

Table 68 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

Table 68 Command Line Options (continued)

Name	Description
<code>-fmt:[tab align html report loadable]</code>	Output format: <i>tab</i> = separate columns by tabs (default) <i>align</i> = align columns with spaces <i>html</i> = generate HTML file <i>report</i> = create HTML report (for metrics only) <i>loadable</i> = create loadable metrics results file (*.mtbl)
<code>-dcpy:[directory]</code>	Target folder where the descriptions are copied. This option is only valid for HTML format.
<code>-dref:[html-ref]</code>	Reference to this folder in HTML file. This option is only valid for HTML format.
<code>-cmt:[comment]</code>	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 (Tools | 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`).

Extending the QA module

To some extent, you can customize individual audits and metrics by setting the options in the right pane of the Audits or Metrics window.

However, if you have more specific needs, you can create your own custom audit and metric plug-ins to extend the QA module.

The QA module resides in `TGH/modules/com/togethersoft/modules/qa`. This directory also includes detailed API documentation, samples for writing custom QA plug-ins, and source code examples.

To write your own custom QA plug-in:

1. Write Java source code for your custom audit or metric. For instructions and tips, refer to the API documentation for the QA module (`TGH/modules/com/togethersoft/modules/qa/doc/index.html`).

Tip Remember to include the following required libraries in the project classpath:

```
$TGH$/lib/openapi.jar
```

```
$TGH$/modules/com/togethersoft/modules/qa/qa.jar
```

2. Compile the Java class and create a new directory for it under the appropriate QA module subdirectory (under `/qa/audit/` for audit plug-ins, or under `/qa/metrics/` for metric plug-ins).

The class name must be exactly the same as the name of the directory that you create for it. The package name for the plug-in must be equal to `com.togethersoft.modules.qa.<subsystem>.<plugin_name>`, where `<subsystem>` is either `audit` or `metrics` and `<plugin_name>` is the class name.

3. Write a description of your custom plug-in in an HTML file and place it in the directory that you created for the plug-in.

The name of the description file must be the same as the class file, but with the `.html` extension.

4. If your custom plug-in is a metric, write a results tip in an HTML file and place it in the directory that you created for the plug-in. The tip should give advice on how to interpret the results of your metric. This text is displayed on the Advice tab on the Distribution graph.

The name of the tip file must be equal to `<plugin_name>_Tip.html`, where `<plugin_name>` is the class name.

5. Launch a separate instance of Together for testing the plug-in.

Note To view messages for debugging the plug-in, you need to watch the Together console. Do not launch the `Together.exe` file, because it does not display the console. Use `TogetherCon.exe`, `Together.sh`, or `Together.bat` instead.

6. Run the corresponding QA feature (Audits or Metrics). Your custom plug-in should be displayed in the list of available audits or metrics.

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 31, “J2EE Support”
- Chapter 32, “Creating EJB Components”
- Chapter 33, “Assembling EJB Modules”
- Chapter 34, “Container Transactions and EJB References”
- Chapter 35, “Designing and Developing Web Applications”
- Chapter 36, “Designing and Developing Enterprise Applications”
- Chapter 37, “Designing and Developing Application Clients”
- Chapter 38, “Designing and Developing Resource Adapters”
- Chapter 39, “J2EE Platform Security Support”
- Chapter 40, “J2EE Deployment”

Chapter 31

J2EE Support

Together provides deep support for the Java™ 2 Platform, Enterprise Edition (J2EE). With Together, you can create servlets, JSPs, EJBs, and Web or enterprise applications, and then deploy them to your current application server. This chapter gives an overview of the features that enable developing such multitier distributed applications.

All chapters in this part of the *User Guide* assume that you are familiar with fundamental J2EE concepts. (See <http://java.sun.com/j2ee> for extensive information on J2EE specifications and usage.) The topics covered in this chapter are:

- “Supported J2EE technologies” on page 553
- “Special J2EE diagrams” on page 554
- “References support” on page 558
- “Security support” on page 558

Supported J2EE technologies

Multitier enterprise applications are typically comprised of J2EE components such as servlets and EJBs that link to databases or implement business logic and HTML pages, JSPs, and applets that interface with the user. Developing an application requires creating its component pieces, writing deployment descriptor *.xml files, and then deploying an archive of the resulting files to an application server.

Together supports creating and deploying EJB modules, Web applications, and enterprise applications by:

1. Providing special J2EE diagrams. The diagrams enable you to model applications, organize the contents of JARs, WARs, EARs, and RARs, use tag libraries, set EJB references, and specify security mechanisms.
2. Creating deployment descriptors and archives based on diagrams and your current application server.
3. Deploying from diagrams.

Together automates deployment of the following major components:

- Web files (such as HTML and GIF)
- Servlets and JSP files
- JSP tag libraries
- Web applications
- EJBs and EJB modules
- Classes required for servlets, JSPs and EJBs
- Application clients
- Resource adapters

Together's e-commerce feature provides its J2EE support. This feature is activated by default. If the e-commerce feature is not activated for your configuration, choose **Tools | Activate/Deactivate Features** from the main menu and check **E-Commerce**.

Special J2EE diagrams

Together has several types of diagrams to support the J2EE platform. These are special to Together and are not among the canonical UML diagrams.

Together's J2EE diagram types are:

- EJB assembler diagram: for JAR files of J2EE components.
- Web application diagram: for WAR files of J2EE and Web components.
- Enterprise application diagram: for EAR files.
- Taglib diagram: for JSP tag libraries.
- Application client diagram: for applications that run on client machines.
- Resource adapter diagram: for RAR files.

You can deploy enterprise applications from class diagrams, Web application diagrams, EJB assembler diagrams, and enterprise application diagrams.

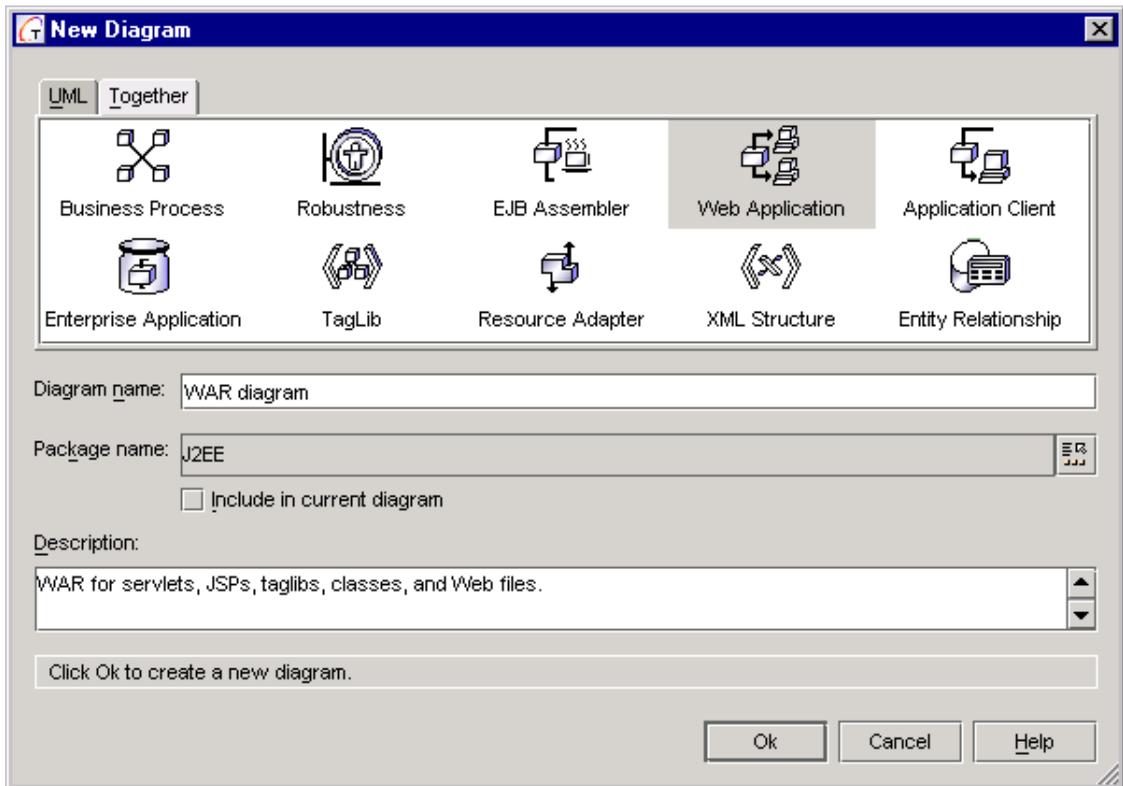
Creating a J2EE diagram

Creating a new J2EE diagram consists of four steps:

1. Click the **New Diagram** button () on the Designer pane's horizontal toolbar.
2. Click the **Together** tab to reveal the non-UML diagram types.
3. Select the desired J2EE diagram type from the icon display. [Figure 154](#) shows the dialog box for creating a new Web application diagram.
4. Fill in the diagram name and documentation as desired. Then click **Ok** to complete the process.

As an alternative to the steps listed above, you can use the Object Gallery to access “experts,” which generate J2EE diagrams based on your input. The Object Gallery is available from the **File | New** command on Together's main menu. Select **Enterprise** to access e-commerce experts; select **Web** to access Web experts.

Figure 154 Creating a new Web application diagram



EJB assembler diagram

An EJB assembler diagram provides a visual representation for a JAR archive. The JAR stores all EJBs (entity, session, and message-driven beans) as well as the other classes such as exceptions and utility classes required for the EJBs.

To start developing an EJB module, create an EJB assembler diagram. Alternatively, use the Object Gallery to create a new EJB module, which automatically generates an EJB assembler diagram. For complete instructions on how to create use an EJB assembler diagram for developing EJB modules, refer to [Chapter 33, “Assembling EJB Modules”](#). Once you have created an EJB module, use the J2EE Deployment Expert to generate deployment descriptors and deploy the EJB module to your current application server. The J2EE Deployment Expert is available from the Deploy | J2EE Deployment Expert command on Together’s main menu. See [Chapter 32, “Creating EJB Components”](#), and [Chapter 33, “Assembling EJB Modules”](#).

Web application diagram

A Web application diagram provides a visual representation of a Web application archive (WAR) that stores all JSPs, servlets, taglibs, classes, and Web files. Together uses the information on the Web application diagram to create the deployment descriptors for your current application server.

For complete instructions on how to create and use a Web application diagram to develop Web applications, refer to [Chapter 35, “Designing and Developing Web Applications.”](#) After creating the diagram, you can use the J2EE Deployment Expert to generate a deployment descriptor and deploy the Web application to your current application server.

Enterprise application diagram

An enterprise application diagram enables you to combine different JARs, WARs and RARs into an enterprise application archive (EAR). Enterprise application diagrams are also useful in modeling global security constraints via security roles in the EJB/WAR modules.

For complete instructions on how to create an enterprise application diagram and use it for developing enterprise applications, refer to [“Designing and Developing Enterprise Applications” on page 637](#). You can add shortcuts to application clients on an enterprise application diagram. From the diagram, you can use the J2EE Deployment Expert to generate a deployment descriptor and deploy the application to your current application server.

Taglib diagram

JSP technology has support for reusable modules called *custom actions*. This support is accessible in versions 1.1 and higher. Custom actions require custom tags; a *tag library* is a collection of custom tags. A taglib diagram is intended for using and developing JSP tag libraries. Such a diagram is convenient for tasks like message managing or accessing databases.

To develop a tag library or to use an existing one, start by creating a taglib diagram. For complete instructions on how to create and use a taglib diagram, refer to [“Working with tag libraries” on page 627](#).

You cannot deploy directly from a taglib diagram. However, you can place shortcuts to taglib diagrams on Web application diagrams for later deployment.

Application client diagram

Application clients are programs that execute on users’ machines. They typically have graphical user interfaces, and they depend on a container to provide system services. Application client diagrams are intended for design and development of application client programs.

For complete instructions on how to create an application client diagram and use it for developing application clients, refer to [Chapter 37, “Designing and Developing Application Clients” on page 645](#).

You cannot deploy directly from an application client diagram. Instead, you can place shortcuts of application client diagrams on EJB assembler or enterprise application diagrams for later deployment.

Resource adapter diagram

A resource adapter diagram describes the 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 diagrams are useful for integrating J2EE applications with Enterprise Information Systems (EISs) that are not relational databases.

Resource adapter elements provide access to any resources anywhere. Elements on a single resource adapter diagram can be archived into an RAR file. For complete instructions on how to create and use resource adapter diagrams, see [Chapter 38, “Designing and Developing Resource Adapters” on page 651](#).

You can use the J2EE Deployment Expert to generate a deployment descriptor and deploy a resource adapter to your 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 two additional types of references:

- EJB local references (references to other EJBs located in the same container)
- Resource-environment references

A reference is a special attribute in the EJB class. You can add references to an EJB by choosing **New | EJB Reference** on its right-click menu. Alternatively, you can add references and make changes on the Reference tab in the EJB's properties inspector.

The deployment descriptor, which is generated in the deployment process, saves information about references. An EJB can request the current status of transactions, security, links between EJBs, and so on, via the descriptor context.

For instructions on how to work with references, refer to [Chapter 34, "Container Transactions and EJB References"](#) on page 601.

Security support

J2EE security mechanisms specify security policies in an operational environment. Together supports J2EE security through special elements on J2EE diagrams. When you deploy from a J2EE diagram, Together's J2EE Deployment Expert places the information from these security elements into the deployment descriptor.

Most of Together's J2EE-oriented diagram contain security design elements.

- 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*

For instructions on how to work with security information, refer to [Chapter 39](#), “J2EE Platform Security Support” on page 657.

Creating EJB Components

This chapter explains how to design and develop enterprise Java beans. The chapter includes the following topics:

- “Setting the EJB properties for a project” on page 561
- “Creating EJB components” on page 562
- “Using EJB Inspectors to set EJB properties” on page 569
- “Working with EJBs” on page 581
- “Sharing home and remote interfaces” on page 588
- “Customizing the default EJB code” on page 589

Setting the EJB properties for a project

Every Together project is associated with an application server. The choice of application server determines the EJB properties available to a project. That choice does not determine which application server will be the ultimate target of deployment.

To choose an application server for an open Together project:

1. From the main menu, choose **Project | Project Properties**.
2. Click the **EJB** tab.
3. Choose the desired application server from the *Application Server* list.
4. Click **Ok** to quit the dialog and save the selection.

Note You can also set the EJB properties of a project through the options dialog. Choose **Tools | Options | <level>** from the main menu. Click the *EJB* node on the left, and pick the server in the *Application server* list on the right.

The EJB properties that the choice of application server impacts include:

- If the application server is EJB 2.0 compliant or Generic 2.0:
 - The class diagram toolbar contains a message-driven EJB button.
 - You can create local interfaces.
 - You can create application client diagrams and resource adaptor diagrams.
- The skeleton classes and interfaces that Together generates for EJBs and J2EE patterns vary according to whether the application server is EJB 1.0, 1.1, or 2.0 compliant.
- Some application server choices result in special EJB Inspector tabs.

To use encoding:

1. Choose **Tools | Options | <level>** from the main menu, and click the *EJB* node on the left.
2. Make sure that *Do not write encoding declaration for deployment descriptors* checkbox on the right is checked. Otherwise, you should know that encoding used for your project probably is not supported by the selected application server.

The J2EE Deployment Expert uses the application server for a project by default for the deployment target. You can choose a different target when you actually deploy. This makes it possible to deploy to two or more different application servers with the same project.

Creating EJB components

This section discusses the basics of viewing, editing, and creating EJB components.

Note Before creating EJBs, you should verify that the project's EJB properties are appropriate for your requirements. The code that Together generates depends on the choice of server and corresponding EJB specification.

Setting component editing and display options

The interfaces and classes of an EJB component act as a single entity, which is represented by the single implementation bean class.

By default, when you right-click an implementation bean class and choose **Delete**, **Cut**, **Copy**, or **Clone**, the interfaces and primary key classes are deleted, cut, copied, or cloned too. You can change that default behavior.

To change the behavior of Delete, Cut, Copy, Clone, or Paste of an EJB:

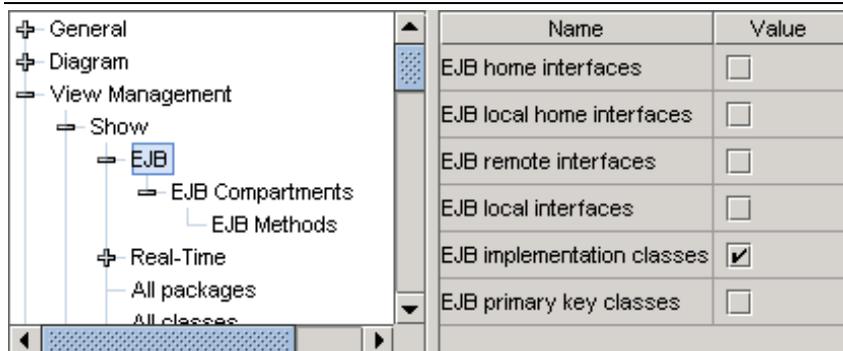
1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, choose *EJB - Auto update related elements*.
3. On the right pane, check which component elements to clone, copy, paste, delete, or cut when applying the same action to the implementation bean.
4. Click **Ok** to save your choices.

By default, class diagrams display EJB implementation classes, but they hide interfaces and primary key classes. You can change the diagram options to display any of the EJB component parts.

To show or hide EJB component classes or interfaces:

1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, expand *View Management - Show*.
3. Choose *EJB*.
4. On the right pane, check which component elements you want to display. [Figure 155](#) shows the Options dialog box for displaying only the implementation classes.
5. Click **Ok** to save your choices.

Figure 155 Showing and hiding EJB component elements in the Options dialog



Although some EJB component part may be hidden in the Designer pane, the Explorer pane lists all parts. You can edit source code for any part by selecting it on the Explorer.

Setting component synchronization options

When you change the remote or home interface of an EJB, Together synchronizes the corresponding changes with the bean implementation class methods by default. For each method in the new interface that is not part of the original bean, Together creates a method stub in the bean class.

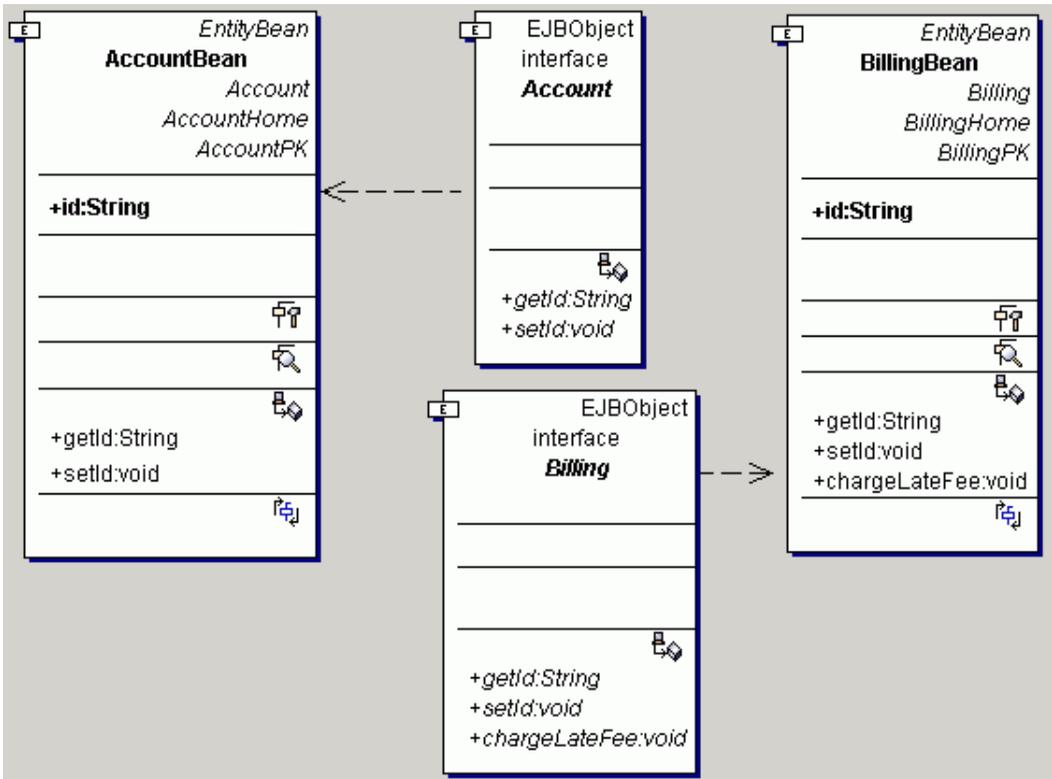
You can change that default behavior for automatic synchronization through the project or default options.

To change the EJB component synchronization properties:

1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, choose *EJB - Synchronize methods when changing interfaces*.
3. On the right pane, un-check the component methods that you do not want to synchronize (bean class methods, remote or home interface or local interface methods).
4. Click **Ok** to save your choices.

The following example illustrates how Together synchronizes the changes. [Figure 156](#) shows two entity EJBs, `AccountBean` and `BillingBean`. Notice the `chargeLateFee` business method for `BillingBean`.

Figure 156 Two entity beans with their corresponding remote interfaces



Continuing the example in [Figure 156](#), suppose we change the remote interface for `AccountBean` to `Billing` by following these steps:

5. Right click `AccountBean` and choose **Properties** to open its Inspector.

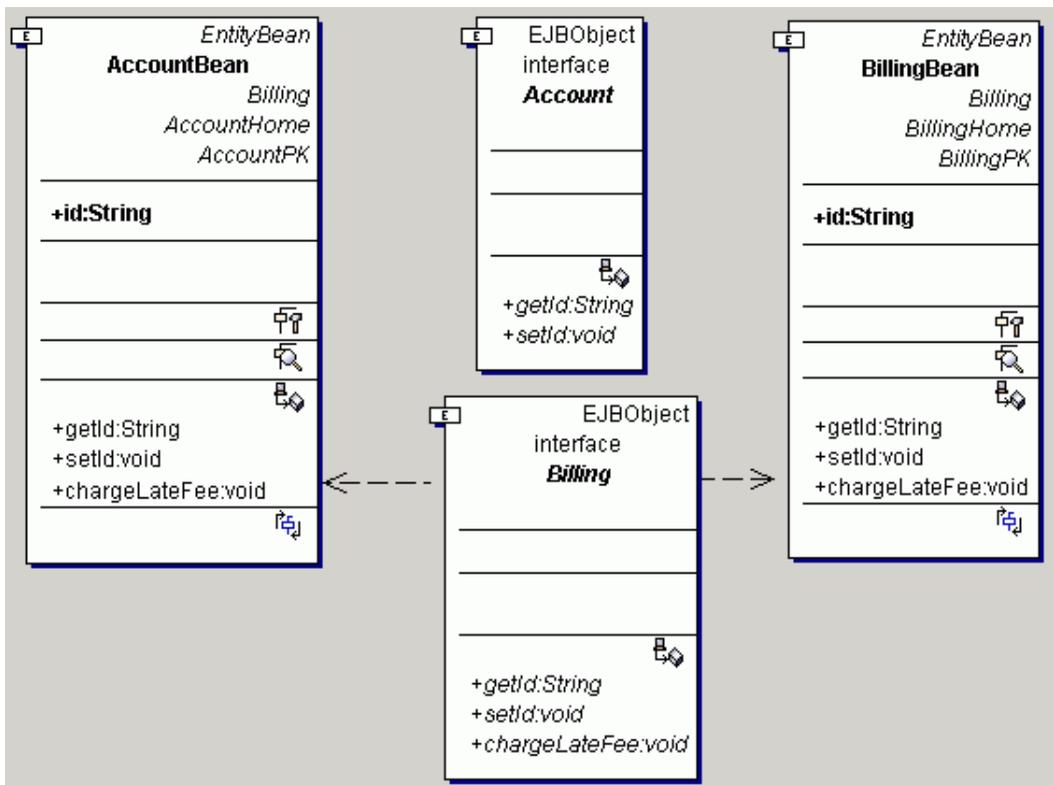
6. Choose the **Entity EJB** tab on the top, then choose the **General** tab on the bottom.
7. Click **Do not synchronize names**.
8. For the *Remote name* field, choose **Billing**.
9. In the resulting dialog box, click **Change class**.

Figure 157 shows the resulting class diagram with the new remote interface for AccountBean. Since synchronization with remote interfaces is turned on, Together provides this stub for the method `chargeLateFee` in the AccountBean:

```
public void chargeLateFee() throws RemoteException,
    javax.ejb.EJBException {}
```

If the synchronization is turned off, Together does not create new methods such as `chargeLateFee` in the implementing bean to correspond to the methods in the new interface.

Figure 157 Entity EJBs sharing a remote interface



Creating EJB components from the class diagram

The class diagram toolbar has three different EJB buttons, corresponding to the three different kinds of EJB components. The toolbar buttons are listed in [Table 69](#).

When you create an EJB component from the toolbar, Together provides skeletons for the home and remote interfaces, the bean, and the primary key class (for entity beans). The code is consistent with the application server specified in the project properties.

Table 69 EJB buttons on the class diagram toolbar

Button	EJB component type
	Entity EJB.
	Session EJB.
	Message-driven EJB. This button displays only when the project application server is EJB 2.0 compliant (or Generic 2.0).

Entity and session EJBs that satisfy the EJB 2.0 specification can have local interfaces. Together does not generate these interfaces when you create a new EJB.

To create local interfaces, right click the bean on the class diagram and choose **New | Create Local Interfaces** from the right-click menu. The two new local interfaces have the same method signatures as their non-local counterparts.

Creating EJB components using the Object Gallery

The Object Gallery provides EJBs among its collection of objects.

To create a new EJB using the Object Gallery:

1. On the main menu, choose **File | New**. This displays the Object Gallery.
2. Choose **Enterprise** among the categories listed on the left.
3. Choose the **EJB** icon on the right, and click **Next**.

The resulting pages provide an expert to guide you through the process of creating the EJB, where you can specify the following values:

- **EJB name.** The name of the EJB to be shown on the diagram.
- **EJB display name.** The name of the EJB to use in the deployment descriptor.
- **EJB type.** Entity, session, or message-driven.
- **Package location.** The <default> package is the default location. You can click the file-chooser button to specify an existing package.

- **Options.** Check this to place a shortcut to the bean on an existing EJB assembler diagram. (A complete discussion on EJB modules is in [Chapter 33, “Assembling EJB Modules.”](#))
4. Click **Finish** if the bean is message-driven. Otherwise, click **Next** to set type-specific properties for entity or session beans.
 - **JNDI name.** Common to entity and session beans.
 - **Session bean.** Choose **Stateful** or **Stateless**.
 - **Entity bean.** Choose persistence type and check for simple primary key. If the bean has container-managed persistence, enter the **Pool name** and **Table name**.
 5. Click **Finish** to close the dialog and generate the bean code.

Applying an EJB pattern to an existing class

You can change an existing class into an EJB implementation class by applying an EJB pattern.

To apply an EJB pattern to an existing class:

1. Right-click the class and choose **Choose Pattern**.
2. In the left pane of the resulting dialog box, expand *EJB*.
3. Choose the desired bean type: *Entity EJB*, *MessageDriven EJB*, or *Session EJB*.
4. Click **Finish** to close the dialog and generate the bean code.

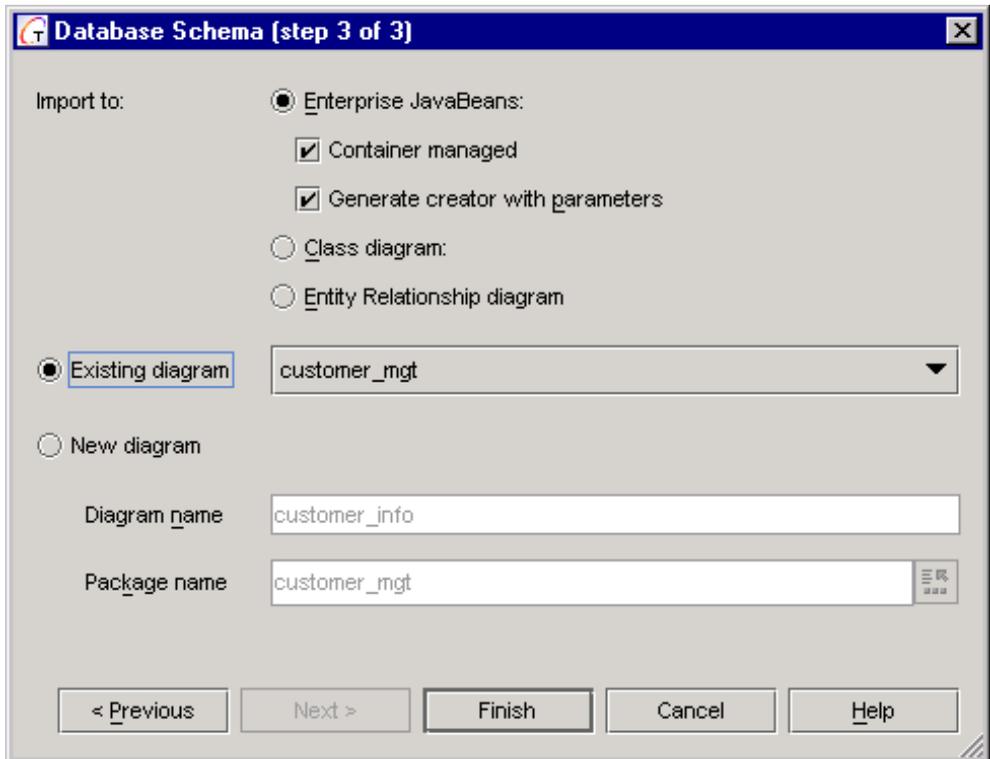
Applying the EJB pattern to a class appends `Bean` to the class name. Together creates remote and home interfaces and primary key classes for entity EJBs. Together also adds new members to the bean class appropriate for the project’s targeted application server.

Applying an EJB pattern to an existing class may modify some of the original class members. For example, Together removes constructors and adds the appropriate `throws` statements to other operations.

Importing entity EJBs from a database

You can import a database to create new entity EJBs in an existing Together project. [Chapter 7, “Importing and Exporting Information”](#) has instructions for importing a database. [Figure 158](#) shows the concluding dialog box for importing a database.

Figure 158 Importing entity EJBs from a database



There are two options for importing a database to EJBs:

- **Container managed.** Check to create a bean with container managed persistence. Leave it unchecked to create a bean with bean managed persistence.
- **Generate creator with parameters.** Check to generate an `ejbCreate()` method in which all columns of the database table are parameters. For example, if the source table has fields `CHAR column1` and `INT column2`, then the method signature is `ejbCreate(String, int)`.

When you import an EJB from a database, Together generates the code for an entity EJB component and places the component elements on the diagram. The diagram shows dependencies from the remote interface, the home interface, and the primary key class to the bean implementation class. The code that Together creates is consistent with the application server for the project.

Reverse engineering EJB source code

You can create a new project around existing EJB source code by reverse engineering. When you reverse engineer EJB source code, Together displays the EJB components on their package diagrams. (For instructions on how to re-engineer existing code, see [Chapter 6, “Reverse Engineering.”](#))

Using EJB Inspectors to set EJB properties

EJB inspectors have special EJB tabs. To open the Inspector, right-click the bean and choose **Properties** from the menu. When you click the **EJB** tab at the top, the special tabs display at the bottom. These tabs are for specifying general EJB properties, adding and removing fields and methods, and defining references, relationships, and deployment properties.

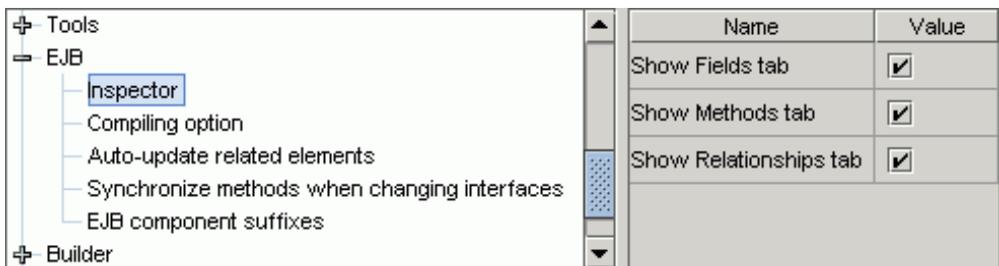
Configuring EJB inspectors

All EJB inspectors have **Deployment properties** and **EJB references** tabs. Session and entity EJB inspectors also have a **General** tab. You can hide or display three other EJB tabs: **Fields**, **Methods**, and **Relationships**.

To hide or display the optional tabs:

1. From the main menu, choose **Tools | Options | <level>**.
2. On the left pane, expand *EJB*. Then choose *Inspector*, as shown on [Figure 159](#).
3. On the right, check or clear *Fields*, *Methods*, and *Relationships* to display or hide the corresponding subtabs in EJB inspectors.

Figure 159 Setting EJB Inspector tabs in the Project Options



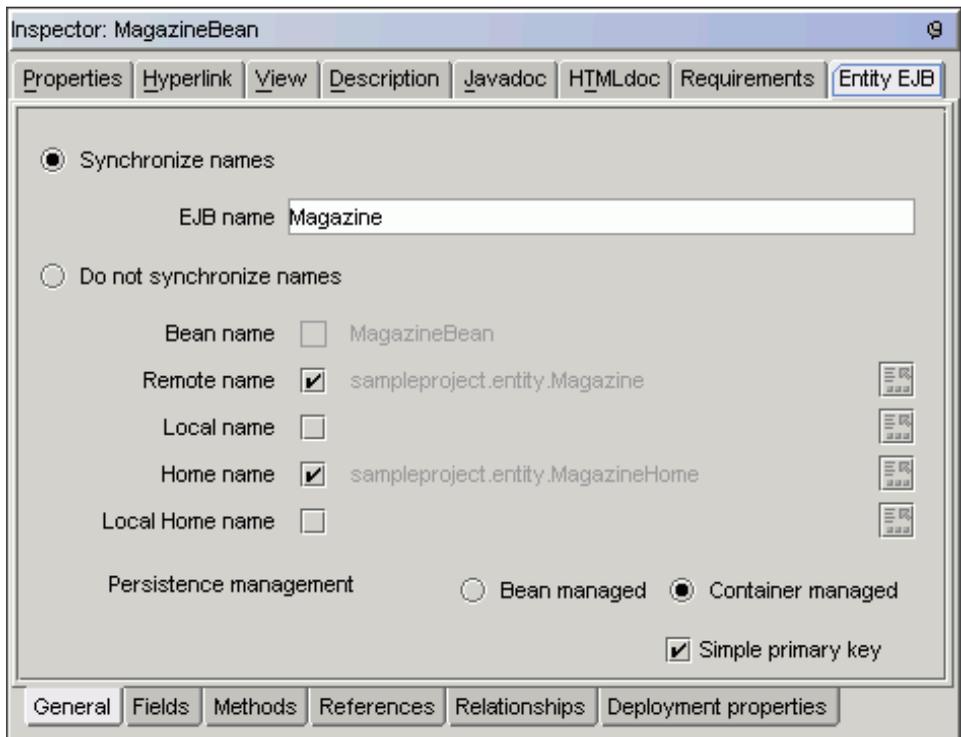
EJB Inspector tabs

The Inspector EJB tabs vary depending on the bean type, the current target application server for the project (EJB properties), and the Inspector configuration settings described above. This section discusses all tabs.

General tab

The **General** tab contains settings for naming component parts and for specifying persistence and primary key mapping. This tab is not present in inspectors for message-driven beans. [Figure 160](#) shows the **General** tab for a container-managed entity bean named `MagazineBean`.

Figure 160 General tab of an entity EJB Inspector



The **General** tab settings include the following:

- **Synchronize names.** Default naming convention. If you change the EJB name, the names of the interfaces change automatically to match the new name.
- **Do not synchronize names.** Alternative naming convention for specifying interface names independent from the bean name. The text fields for entering the names are available only when you choose this setting.

Remote interfaces are checked by default. *Local name* and *Local Home name* are enabled only when you are working with EJBs satisfying the EJB 2.0 specification.

- **Stateful.** (Session beans only) checkbox for specifying stateful or stateless session bean.
- **Persistence management.** (Entity beans only) default value is *Bean managed*.

- **Simple primary key.** A check indicates that there is a single field in the bean for the 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 entity bean class.
 - Primary key maps to multiple fields of the entity bean class (for implementation of compound keys via the user-defined `PKclass`).

To create a primary key of the first type, delete the created `PKclass`. Then check *Simple primary key*, which is enabled for container-managed beans if there is only one primary key field with the appropriate object type. This helps avoid wrapping of the simple type primary key (such as `String`) into a user-defined class.

References tab

The **References** tab is for viewing existing references, changing their properties, removing them, and creating new references.

Radio buttons at the top of the **References** tab indicate which type of reference is currently displayed. Each display consists of a table with a row for each reference of the current type and columns for the properties.

To add a new reference or remove an existing one:

1. Click the radio button corresponding to the kind of reference.
2. To add a new reference, click **Add**. This creates a new line in the table where you can edit the reference properties.
3. To remove a reference, choose its row in the table and then click **Remove**.

There are six kinds of references:

- **EJB:** EJB references. Some of the EJB reference fields are common to other reference types.
 - **Name.** Name of the bean attribute that corresponds to the reference.
 - **EJB reference name.** Specific name to use for look-up.
 - **EJB reference.** Type of EJB that is referenced (session or entity). If you use the remote interface or home interface for the referenced bean, then the value of this column is filled in automatically.
 - **Remote interface.** Remote interface of the referenced EJB. This column has a chooser button for selecting the remote interface of the referenced bean.
 - **Home interface.** Home interface of the referenced EJB.
 - **EJB link.** Parameter in the deployment descriptor.
 - **EJB JNDI name.** Parameter in the deployment descriptor.

- **EJB local.** Local references. EJB local is visible only if the EJB satisfies the EJB 2.0 specification (that is, if you choose Generic 2.0, or an EJB 2.0 compliant application server for the project). The columns are the same as those listed above.
- **Resource.** Resource references. There are five columns for each resource reference. The columns *Name*, *Resource reference name*, and *Resource JNDI name* are analogous to their counterparts for EJB references. The two additional columns are:
 - **Resource reference type.** Data source type of the resource. This column has a list of data source types: `javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.sql.XADataSource`, `javax.mail.Session`, and `javax.net.URL`.
 - **Resource reference authentication.** Specifies *Application* or *Container* authentication.
- **Resource-environment.** References to the container (local) resources. This type is visible only if the EJB satisfies the EJB 2.0 specification. *Name*, *Resource reference name*, and *Resource reference type* columns are analogous to their Resource reference counterparts described above.
- **Environment.** Environment references. The columns *Name* and *Environment entry name* are analogous to *Name* and *EJB Reference name* for EJB references. There are two additional columns:
 - **Environment entry type.** The type of an environment reference. This list consists of `java.lang` types: `Boolean`, `String`, `Integer`, `Double`, `Byte`, `Short`, `Long`, and `Float`.
 - **Environment entry value.** The value of the entry.
- **Security role.** Security role references. The column *Name* is analogous to the *Name* for ordinary EJB references. There are two additional columns:
 - **Role name.** The specific name to use for look-up.
 - **Role link.** An attribute in the deployment descriptor.

Deployment properties tab

The **Deployment properties** tab is for defining the major EJB properties required for the deployment process and used in the deployment descriptor. The properties on this tab vary according to the type of bean and the target application server for the project. See http://java.sun.com/dtd/ejb-jar_2_0.dtd for more details.

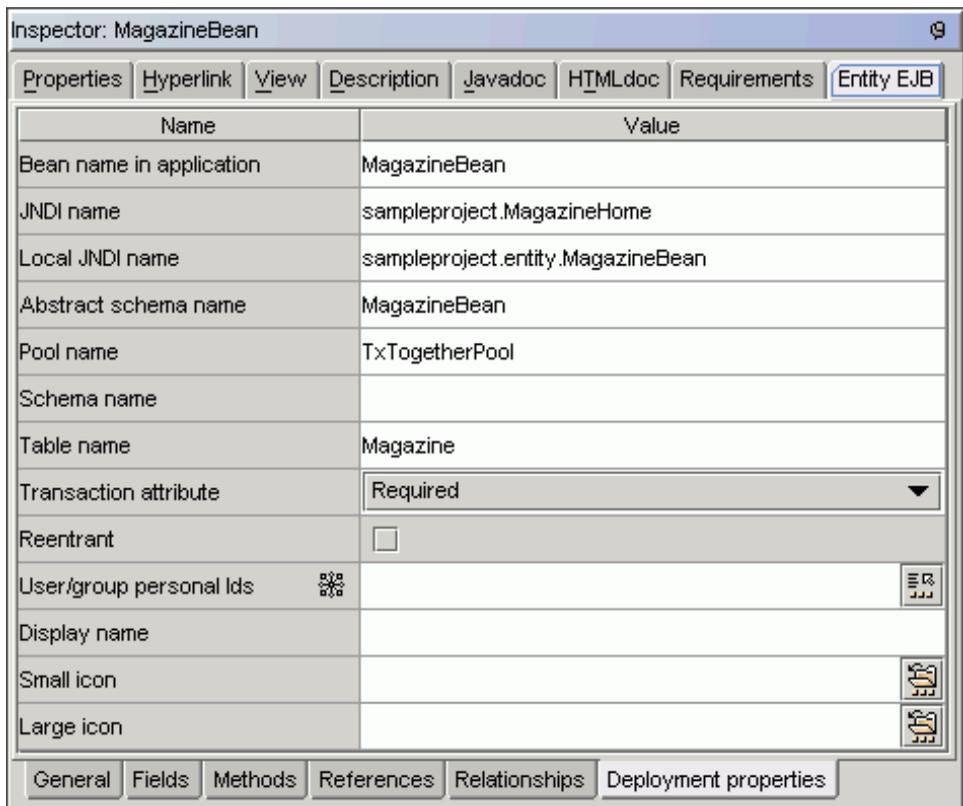
[Figure 161](#) shows the Deployment properties tab for an entity EJB in the Magazine sample that ships with Together.

There are several categories of Deployment properties:

- External access fields specify how the bean is named.

- **Bean name in application.** The bean name in the deployment descriptor. This has the same default meaning as *Bean name* on the General tab.
- **JNDI name.** The JNDI name for a bean's home interface.
- **Local JNDI name.** The JNDI name for a bean's local home interface. This property is enabled only for entity and session beans and if the project satisfies the EJB 2.0 specifications. If a bean has both a remote home and a local home interface, then you must specify two JNDI names, one for each interface.

Figure 161 Deployment properties tab in the entity EJB Inspector



- Database settings (entity beans only) are four fields whose meanings vary according to the particular application server. All of the fields are disabled for BMP beans and enabled for CMP beans. They include:
 - **Abstract schema name.** Defaults to the bean name.
 - **Pool name.**
 - **Schema name**
 - **Table name.**

- Transaction settings consist of three fields:
 - **Transaction attribute.** Has the following possible settings: *NotSupported*, *Supports*, *Mandatory*, *Required*, *RequiresNew*, *Never*.
 - **Transaction management type.** (Session and message beans only) has two settings: *Bean* and *Container*.
 - **Reentrant.** (Entity beans only) should be checked if this entity EJB allows reentrant calls.
- Display and user settings are optional fields. They include:
 - **User/group personal IDs.** For inserting personal identifiers. This field has a multi-string editor.
 - **Display name.** A parameter for the deployment descriptor.
 - **Small icon.** For selecting the small icon (16x16 pixels *.gif or *.jpg image) for the application server's icon.
 - **Large icon.** For selecting the large icon (32x32 pixels *.gif or *.jpg image) for the application server's icon.

Fields tab

The **Fields** tab is available for entity beans only. This tab is for adding, editing, or deleting fields in the entity bean. The different fields display in a table, with rows for fields and columns for field properties.

To add a new field, click **Add** at the top of the tab. To remove a field, select the field and click **Remove**.

Note You can also add a new field to an entity bean directly from the diagram by right-clicking the bean and choosing **New | Field**. You can delete a field using **Delete** on the right-click menu of the field.

Each field has six properties:

- **Type.** The type of the field. You can edit the type from the list (*String*, *boolean*, *byte*, *char*, *double*, *float*, *int*, *long*, *short*, *java.util.Hashtable*, *java.util.Vector*) or pick the type from the corresponding chooser button.
- **Name.** The name of field member in the bean class.
- **Column.** (CMP beans only) the name of the field in the database.
- **Label.** The representation in a JSP client.
- **Primary key.** Check if this field is used as a primary key.
- **Relationship.** (CMP beans only) check if the field is used in a container-managed relationship. See [“Creating container-managed relationships” on page 583](#).

Methods tab

The **Methods** tab is available for entity and session beans. It is for adding, modifying, or removing methods. You can add a new method on this tab by clicking **Add** at the top of the tab. You can remove a method by selecting the method and clicking **Remove**.

Note You can add a method to a bean directly from the diagram by right-clicking the bean and selecting **New | <type of method>**. You can delete a method using **Delete** on its right-click menu.

When you choose **New | Business method**, Together adds the business method to the remote interface as well as the bean class. If you do not want the method to be added to the remote interface, choose **New | Method** instead.

Radio buttons at the top of the Methods tab indicate which type of methods are currently displayed. There are five kinds of methods:

- **Creators.** Three creator properties are:
 - **Method signature.** Signature in the bean class.
 - **Exposed in Home.** Check if this method is accessible through the bean's home interface
 - **Exposed in Localhome.** Check if this method is accessible through the bean's local home interface. This property is enabled only if the target server is Generic 2.0 or EJB 2.0 compliant and if the bean has a local home interface.
- **Finders.** (Entity beans only) Properties are the same as creator method properties. There are two additional Finder properties:
 - **Multi.** Check if the finder can return many elements.
 - **Finder query.** The corresponding query.
- **Business methods.** Four business method properties are:
 - **Method signature.** Signature in the bean class.
 - **Exposed in remote interface.** Check if this method is exposed in the bean's remote interface.
 - **Exposed in local interface.** Check if this method is accessible through the bean's local interface. This property is enabled only if the target server is Generic 2.0 or EJB 2.0 compliant and the bean has a local interface.
 - **Transaction attribute.** A list consisting of *<bean default>*, *NotSupported*, *Supports*, *Mandatory*, *Required*, *RequiresNew*, *Never*.
- **Home methods.** (Entity beans only) Available only for entity EJBs satisfying the EJB 2.0 specification (the target server is Generic 2.0 or EJB 2.0 compliant). The home method properties are identical to the properties of creator methods.

- **Selectors.** (Entity beans only) Available only for CMP entity beans that satisfy the EJB 2.0 specification and that have simple primary keys. The Method signature and Finder query properties are the same as for finder methods.

Note According to the EJB 2.0 specification, all methods except selector methods can be exposed either in a remote interface or in a local interface.

Relationships tab

The **Relationships** tab is available only for CMP entity beans satisfying the EJB 2.0 specification. The relationships display in a table with rows for the relationships and columns for the relationship properties. There are seven properties:

- **CMR field name.** The name of the container-managed relationship as virtual field in the entity bean 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, with possible settings *one* and *many*.
- **Relationship destination.** The path to the related EJB element.

For additional information, see [“Creating container-managed relationships” on page 583](#).

Setting additional deployment properties

Some EJB inspectors have special tabs corresponding to the more popular application servers. The purpose of these additional tabs is to provide access to optional deployment properties for the EJB.

You can specify additional deployment properties for the following application servers:

- SunEE 1.3, Reference Implementation (*SunEE AS Properties* tab)
- Borland Enterprise Server 5.2 (*Borland Enterprise Server 5.2* tab)
- Be a WebLogic 6.0, 6.1, 7.0 (*WebLogic x.x* tab)

SunEE AS Properties Inspector tab

If the target application server for your project is SunEE 1.3, Reference Implementation, the EJB Inspector for CMP entity bean displays the **SunEE AS Properties** tab.

Check **Create a table on deploy** if you want to automatically create a table while deploying (`<table>` tag).

Check **Delete a table on undeploy** if you want to automatically remove a table while undeploying.

You can also write EJB QL queries in the following fields:

- **SQL statement used to create a table for an CMP EJB**
- **SQL statement used to delete a table for an CMP EJB**

The information contained in these fields is used while generating the EJB module deployment descriptor.

Borland Enterprise Server 5.2 Inspector tab

If the target application server for your project is Borland Enterprise Server 5.2, the inspectors for both EJBs and EJB modules display the **Borland Enterprise Server 5.2** tabs. The contents of these tab are dependent on the EJB type.

Note For additional information on the contents of the Borland Enterprise Server 5.2 tags, see <http://info.borland.com/techpubs/books/bes/pdfs52/>.

Properties is the common field for session, entity, message-driven beans, and EJB modules (EJB assembler diagrams). According to this information, Together generates corresponding custom `<property>` tags in the deployment descriptor.

Every property contains the following information:

- **Name** - a property name
- **Type** - a property type, for example, `java.lang.String`
- **Value** - a property value

Refer to the Borland Enterprise Server 5.2 documentation for a complete list of properties.

Borland Enterprise Server 5.2 Inspector tab for session EJBs

The Borland Enterprise Server 5.2 Inspector tab for a session EJB in addition contains the **Session timeout** (`<timeout>` tag) optional deployment property: time of saving the user session.

Borland Enterprise Server 5.2 Inspector tab for entity EJBs

The **Borland Enterprise Server 5.2** Inspector tab for CMP beans includes the following additional fields and checkboxes:

- **Create table** `<table>`. Check if you want a table to create automatically while deploying.

- **Table properties.** Using this field, you can define different parameters of the database table. Refer to the Borland Enterprise Server 5.2 documentation for a list of parameters.

Borland Enterprise Server 5.2 Inspector tab for message-driven EJBs

The **Borland Enterprise Server 5.2** Inspector tab for a message-driven EJB in addition contains the **Connection Factory name** (a tag `<connection-factory-name>`) optional deployment property: the JNDI name of the JMS `ConnectionFactory` that the bean uses for its queues and topics.

WebLogic Inspector tabs

If the target application server for your project is BEA WebLogic 6.0, 6.1, or 7.0, some of the EJB inspectors display special **WebLogic** tabs. The contents of these tabs as well as whether the tabs are even present vary according to the EJB type and the version of the BEA WebLogic application server.

Note For additional information on the contents of the WebLogic Inspector tags, see <http://e-docs.bea.com/wls/docs70/ejb/>.

WebLogic 6.0 and WebLogic 6.1 Inspector tabs

If the target application server for your project is BEA WebLogic 6.0 or 6.1, entity EJB inspectors display a tab with the label **WebLogic 6.0** or **WebLogic 6.1**. This tab is present for entity EJBs only. There are no such tabs for session or message-driven EJBs.

There are two properties on the WebLogic 6.0/6.1 tab for an entity EJB Inspector:

- **Max beans in free pool.** `<max-beans-in-free-pool>` The maximum size of the free pool of beans that WebLogic maintains for this bean class.
- **Initial beans in free pool.** `<initial-beans-in-free-pool>` The initial size of the free pool of beans that WebLogic maintains for this bean class.

WebLogic 7.0 Inspector tabs

If the target application server for your project is BEA WebLogic 7.0, all EJB inspectors, display **WebLogic 7.0** tabs. This is true for session beans, entity beans, and message-driven beans

The view of this tab and the number of pages with the corresponding subtabs inside the WebLogic 7.0 tab vary among the different kinds of beans.

WebLogic 7.0 Inspector tab for session EJBs

The **WebLogic 7.0** Inspector tab for a stateless session EJB contains exactly the same settings as the WebLogic 6.0/6.1 Inspector tab for an entity EJB. For more information, see [“WebLogic 6.0 and WebLogic 6.1 Inspector tabs”](#) on page 578.

The **WebLogic 7.0** Inspector for stateful session beans gives access to the following optional deployment properties:

- **Maximum beans in cache.** <max-beans-in-cache> The maximum number of stateful session beans that are allowed in memory. Beans removed from the cache are passivated.
- **Idle timeout seconds.** <idle-timeout-seconds> The number of seconds that a stateful session bean can remain idle in an LRU cache. The default value is 30.
- **Cache type.** <cache-type> Choose the cache type from the list:
 - **NRU** (Not recently used): Stateful session beans are removed from the cache when the number of beans in the cache approaches <max-beans-in-cache>.
 - **LRU** (Least Recently used): Stateful session beans are removed from the cache after reaching <idle-timeout-seconds>.

WebLogic 7.0 Inspector tabs for entity EJBs

The **WebLogic 7.0** Inspector tab for entity EJBs is divided into two subtabs for a BMP bean: *General* and *Cache*. CMP bean inspectors have those two subtabs plus two more: *Concurrency* and *Automatic key*. The *General* and *Cache* subtabs for BMP beans are the same as for CMP beans. All subtabs display at the bottom of the Inspector window.

The **General** subtab is identical to the WebLogic 6.0/6.1 tab for entity EJBs. It has the fields *Max beans in free pool* and *Initial beans in free pool*. For more information, see “[WebLogic 6.0 and WebLogic 6.1 Inspector tabs](#)” on page 578.

The **Cache** subtab gives access to the optional deployment properties associated with cache memory:

- **Maximum bean in cache.** <max-beans-in-cache> Maximum number of entity beans of this type that are allowed in memory.
- **Idle timeout seconds.** <idle-timeout-seconds> Maximum amount of time an entity bean of this type is allowed in memory.
- **Read timeout seconds.** <read-timeout-seconds> The number of seconds between `ejbLoad` calls on a read-only entity bean. If this value is 0, the WebLogic Server calls `ejbLoad` only when it brings the bean into the cache.
- **Cache between transactions.** <cache-between-transactions> A check to cache the persistent data of an entity bean between transactions. The default is unchecked, which means caching occurs only during an individual transaction.

WebLogic 7.0 Inspector tabs for CMP entity EJBs

The WebLogic 7.0 Inspector tab for CMP beans includes the subtabs *Concurrency* and *Automatic key* in addition to the *General* and *Cache* subtabs.

The **Concurrency** subtab gives access to the optional deployment properties associated with the strategy for managing concurrent accesses to an entity bean. This subtab has three fields, *Concurrency strategy*, *Verify columns*, and *Optimistic column*.

- **Concurrency strategy.** <concurrency-strategy> Choose the strategy from the list:
 - **ReadOnly:** (For read-only entity beans) Creates a new instance for each transaction.
 - **Exclusive:** Only the EJB container can update the underlying data during a transaction.
 - **Database:** Allocates a separate entity bean instance for each transaction. The database handles locking and caching.
 - **Optimistic:** No locks are held in the EJB container or database during a transaction.
- **Verify columns.** <verify-columns> Specifies the columns in a table that should be checked for validity when optimistic concurrency strategy is used. Choose from the list:
 - **Read:** Checks all of the columns in the table that are read during the transaction.
 - **Modified:** Checks only the columns that are updated by the current transaction.
 - **Version:** Indicates that a version column exists in the table. This column is used to check validity.
 - **Timestamp:** Indicates that a pseudo timestamp column exists in the table. This column is used to check validity.
- **Optimistic column.** <optimistic-column> The database column that contains a version or timestamp value used for optimistic concurrency:

The **Automatic key** subtab enables you to use the *Automatic Sequence/Key Generation* facility. You can set the following parameters:

- **Generator type.** <generator-type> The type of the key generator.
- **Generator name.** <generator-name> The name of the key generator in accordance with its type.
- **Key cache size.** <key-cache-size> The size of the key cache.

WebLogic 7.0 Inspector tabs for message-driven EJBs

The WebLogic 7.0 Inspector tab for a message-driven EJB gives access to the optional deployment properties associated with messaging. The first two fields, *Max beans in free pool* and *Initial beans in free pool*, are identical to those for WebLogic 6.0/6.1 tab for entity beans. (See “[WebLogic 6.0 and WebLogic 6.1 Inspector tabs](#)” on page 578.)

The remaining fields on the WebLogic 7.0 Inspector tab for a message-driven EJB are:

- **Initial context factory.** <initial-context-factory> The initial ContextFactory that the container uses to create its connection factories. If not specified, this defaults to `weblogic.jndi.WLInitialContextFactory`.
- **Provider URL.** <provider-url> The URL provider for the InitialContext. This value is typically `host:port`.
- **Connection factory JNDI name.** <connection-factory-jndi-name> The JNDI name of the JMS ConnectionFactory that the bean uses for its queues and topics. If not specified, this defaults to `weblogic.jms.MessageDrivenBeanConnectionFactory`, which must be declared in `config.xml`.
- **JMS pooling interval.** <jms-polling-interval-seconds> The number of seconds between each attempt to reconnect to the JMS destination.
- **JMS client id.** <jms-client-id> The client id for a topic connection with a durable subscription.

Working with EJBs

This section describes the details of working with entity EJBs, session EJBs, and message-driven EJBs.

Compiling EJBs

Since the interfaces and classes of an EJB component act as a single entity, compiling the implementation class is sufficient to compile the entire component.

To compile an EJB component:

1. Right click the implementation bean class on the diagram.
2. Choose **Compile** from the right-click menu.

This compiles the bean class, related interfaces, and primary key class (entity beans).

Entity beans

When you create an entity bean from the diagram toolbar button or the Object Gallery, Together generates skeletons for the implementation bean, the remote and home interfaces, and a primary key.

The entity bean Inspector, the Editor, and the Designer are helpful for modifying the code for an entity bean. We discuss the Inspector and the Designer here.

Default entity bean code

The default code for an entity bean consists of the following:

- Entity bean implementation class with:
 - `EntityContext` attribute
 - One `int` type field that serves as a primary key
 - Required method declarations:

```
setEntityContext and unsetEntityContext
ejbActivate and ejbPassivate
ejbCreate, ejbPostCreate, and ejbRemove
ejbStore and ejbLoad
ejbFindByPrimaryKey
getField and setField
```
- EJB home interface with two method signatures: `ejbCreate` and `findByPrimaryKey`
- EJB remote interface with two method signatures: `getField` and `setField`
- EJB primary key class with:
 - Default field (`int` type)
 - Primary key, hash code, and equals methods

Class diagrams with entity EJB components show dependency links from the displayed interfaces and primary key class to the implementation (bean) class.

When you create an entity bean, Together generates a primary key (`int field1`) and stubs for the general EJB methods such as `ejbCreate()`. Comments in the code indicate where you should provide the actual definitions: `// Write your code here.`

Note The top command on the right-click menu of a property, create method, business method, primary key, or field creates a new item of the same kind. For example, the top command on the right-click menu for a business method is **New Business Method**.

Entity beans that satisfy the EJB 2.0 specification can have local interfaces. For instructions on generating local interfaces, see [“Creating EJB components from the class diagram” on page 566](#).

Setting the persistence

Entity bean persistence can be bean managed (BMP) or container managed (CMP). Together generates BMP entity beans by default. To create a CMP bean, you can first create a BMP entity bean and then switch the persistence to CMP.

To switch the persistence of an entity bean from BMP to CMP:

1. Open the entity bean Inspector by right-clicking the bean in the diagram and choosing **Properties**.
2. Choose the **Entity EJB** tab in the Inspector.
3. Choose the **General** tab at the bottom of the Inspector.
4. In the *Persistence management* setting, click the *Container managed* radio button. See [Figure 160](#) as an example.
5. Click **Yes** in answer to the question: “All ejbFind methods will be deleted, continue?” CMP bean do not implement finder methods.

Creating primary keys

Any field that is serializable is a potential primary key.

To define a simple primary key:

1. Open the EJB Inspector (right click the bean and choose **Properties**).
2. Go to the **Fields** tab of the Inspector.
3. Choose a field with a serialized type. Then check its *Primary key* column. Make sure the *Primary key* column for any other field is clear.
4. Go to the **General** tab and check *Simple primary key*.

Deploying an entity bean requires a database and a table with fields corresponding to the bean.

Tip You can create a database from entity beans. Choose **Tools | Generate | DDL** on the main menu to open the Generate DDL Expert. 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. Test the connection with the database.

Creating container-managed relationships

The EJB 2.0 specification provides for container-managed relationships (CMRs) between CMP entity beans that have local interfaces.

To create a container-managed relationship in a project satisfying EJB 2.0:

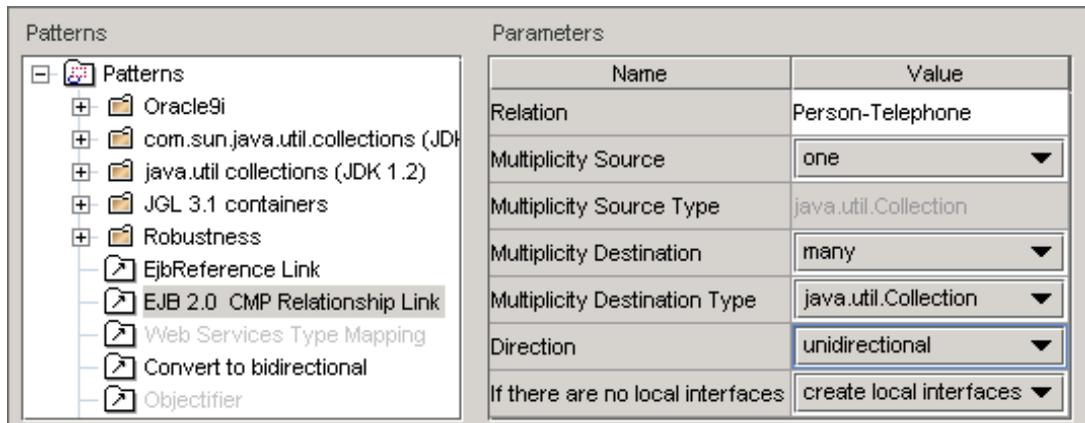
1. Open the class diagram containing the two CMP entity beans.

2. Click the **Association by Pattern** button on the toolbar. Connect the two CMP beans with the relationship.
3. In the left pane of the resulting dialog box, click *EJB 2.0 CMP Relationship Link*.
4. Fill in CMR parameters on the right pane.
5. Click **Ok** to finish the process and quit the dialog.

Figure 162 shows a unidirectional one-to-many CMR named *Person-Telephone*. This dialog setting specifies creating local interfaces if they do not exist rather converting remote interfaces.

The class diagram displays a new virtual field for the CMR on the source bean. It also displays a virtual association connecting the two beans, labeled with the relation name and the multiplicities. Together stores CMR properties as Javadoc comments in the code of the source bean.

Figure 162 Applying a CMR pattern



A many-to-many CMR assumes that there is an “intermediate linking table” that describes the many-to-many mapping. The table name as well as other properties of the CMR are listed in the CMR Inspector.

To change the properties of a CMR:

1. Right-click the virtual field member for the CMR in the diagram and choose **Properties**.
2. Choose the **EJB relationship** tab.
3. Change the properties as needed, including the *Relationship table name* to specify the name of the intermediate table.

Session beans

This section discusses how to modify the code for session beans. You can use the right-click menu of the bean or the session EJB Inspector to tailor much of the code. You must use the Editor to fill in some of the details such as business logic.

Default session bean code

When you create a session bean using the diagram toolbar or the Object Gallery, Together generates skeletons for the implementation class and remote and home interfaces. The default code consists of the following:

- Session bean implementation class with:
 - `SessionContext` attribute
 - `setSessionContext` method
 - `ejbActivate` and `ejbPassivate` methods
 - `ejbCreate` and `ejbRemove` methods
- EJB home interface that has a `create` method with no parameters
- EJB remote interface

The default code does not include business methods or additional creators. It also does not have local interfaces, which are possible if the project's application server conforms to EJB 2.0 specifications.

To create new business methods, creators, or local interfaces:

1. Right click the session bean in the diagram to open its right-click menu.
2. *Business method:* Choose **New | Business method**. Edit the name of the method using the in-place editor on the diagram.

Together writes the method in the implementation bean and puts its signature in the remote interface. Use the Together Editor to put business logic in the body of the method. Use the Together Editor to put business logic in the body of the method.

Note You can also use the session EJB Inspector to create new business methods.

3. *Creator:* Choose **New | Create method**. Edit the name of the creator using the in-place editor on the diagram.
4. *Local interfaces:* Choose **New | Create local interfaces**. Together gives the new local interfaces the same method signatures as their non-local counterparts.

Making the session EJB stateful or stateless

A stateful session bean can retain its state on behalf of an individual client. A stateful bean has three features:

- A serializable member variable for the conversational state
- A method `ejbcreate(<parameter>)` that has an input parameter
- Notation in the deployment descriptor to indicate the bean is stateful

You can use the Editor or the right-click menu of the bean to create the serializable member variable and the method. Together writes the deployment descriptor when you deploy the session bean.

To make the session bean stateful or stateless:

1. Right click the session bean in the diagram and choose **Properties**.
2. Choose the **Session EJB** tab at the top. Then choose the **General** tab at the bottom of the Inspector.
3. At the bottom of the **General** tab, check the *Stateful* box if you want a stateful bean and clear it if you want a stateless bean.

Javadoc comments corresponding to statefulness appear in the source Java code.

Message-driven beans

This section discusses how to modify the code for message-driven beans. You can use the right-click menu of the bean or the message-driven EJB Inspector to tailor much of the code. You must use the Together Editor for some details such as the `onMessage` content. You must use the Together Editor for some details such as the `onMessage` content.

Setting the project properties for message-driven beans

A project containing message-driven EJBs must satisfy two requirements:

- The application server for the project is Generic 2.0 or EJB 2.0 compliant.
- The Search/Classpath libraries include `j2ee.jar`.

To specify the appropriate Search/Classpath for a project with message-driven beans:

1. Choose **Project | Project Properties** from the main menu.
2. Go to the **Search /Classpath** tab. (Click **Advanced** if the tab is not visible.)
3. Clear the checkboxes *Include standard libraries* and *Include Classpath*. They are not necessary.
4. Click **Add Path, Library, or Archive**.
5. In the resulting file chooser dialog, navigate to `j2ee.jar` and click **Select**. You can find `j2ee.jar` in `$(TGH$)/bundled/j2ee/lib`.
6. Click **Ok** to complete the task.

Default message-driven bean code

When you create a message-driven EJB using the diagram toolbar or the Object Gallery, Together generates the bean skeleton. There are no interfaces. The default code consists of the following:

- `MessageDriven` bean context attribute
- `setMessageDrivenContext` method
- `ejbCreate` and `ejbRemove` methods
- `ejbActivate` and `ejbPassivate` methods
- `onMessage` method

The message-driven EJB Inspector is useful for modifying many properties of message-driven beans. It contains only two mandatory subtabs: *References* and *Deployment properties*.

The References tab is almost the same as session and entity EJBs except that there are no EJB security role references. The meanings of the other five types are the same as for session and entity beans. More details are in the section [“References tab” on page 571](#).

Setting deployment properties for message-driven beans

Deployment properties is a tab on the message-driven EJB Inspector. The special message-driven bean properties include:

- **JNDI name to bind to:** Specifies the JNDI name of the JMS ConnectionFactory that the message-driven bean accesses to create its queues and topics.
- **Acknowledge node:** Possible values are *Auto-acknowledge* and *Dups-ok-acknowledge*. This is used for beans with bean-managed transaction management.
- **Destination type:** Possible values are `javax.jms.Queue` and `javax.jms.Topic`.
- **Destination JNDI name:** JNDI name of the queue or topic.
- **Subscription durability:** Possible values are *Durable* and *NonDurable*.
- **Message selector:** A string to filter the incoming messages.
- **Security identity:** Possible values are *none*, *user-caller-identity*, and *run-as-specified-identity*.

For a discussion of deployment properties that apply to all EJB types, see the section [“EJB Inspector tabs” on page 569](#).

Sharing home and remote interfaces

Two or more entity beans or two or more session beans can share the same set of home and remote interfaces. This is potentially useful in deploying differing implementations of the same interfaces on different servers.

Creating a second implementation class

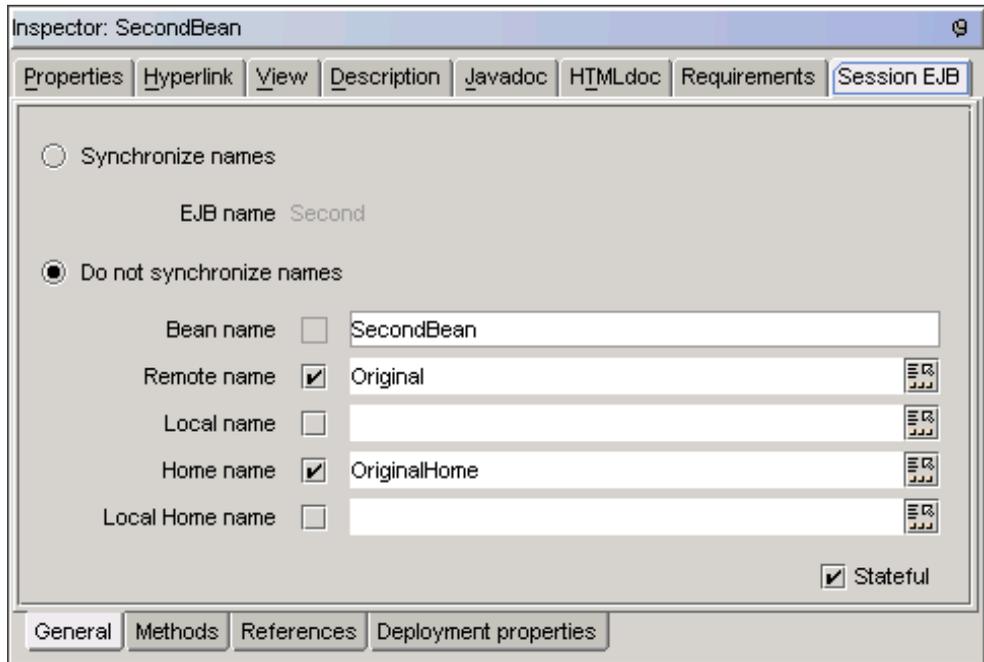
Before attempting to create a second implementation class, you should develop one complete EJB component. Then you can create a second class, refactor it into an EJB, and specify the home and remote interfaces of the first bean as those of the new one.

To create a second implementation class:

1. Create a new class in an existing class diagram or a new one.
2. Refactor the new class into an EJB:
 1. Right click the class node in the diagram and choose **Choose Pattern**.
 2. In the resulting dialog box, expand *EJB* and then choose *Entity EJB* or *Session EJB* in the left pane. Your choice should match the type of the first bean.
3. Right click the new bean and choose **Properties**.
4. In the Inspector for the new bean, click the **Entity EJB** or **Session EJB** tab on the top (depending on the type of bean). Click the **General** tab on the bottom.
5. Choose **Do not synchronize names**.
6. In the *Remote name* field, click the browse button .
7. In the resulting dialog, expand the *Model* node of the tree-view. Locate and choose the remote interface belonging to the first implementation (that is, the remote interface that you want the new bean to use).
8. Click **Ok** to accept the selection. Click **Change class** in response to the prompt.
9. Follow the same procedure in the *Home name* field, selecting the home interface for the first bean in the resulting dialog.

Figure 163 shows the Inspector for a `SecondBean`, which shares remote and home interfaces with `OriginalBean`.

Figure 163 Sharing remote and home interfaces



To correct business methods exposed from shared interfaces, uncheck **Synchronize remote interface** methods in the options dialog (Choose **Tools | Options | <level>** from the main menu, and expose the *EJB* node on the left).

Deleting implementation classes with shared interfaces

Normally when you delete an EJB implementation class from a diagram, Together deletes the home and remote interfaces as well. This includes the source files and their visual diagram elements.

In the case of shared interfaces, deleting elements from the diagram does not automatically result in deletion of the corresponding source code files. Instead, Together deletes only the visual diagram elements. In order to delete the source files, you must explicitly delete implementation classes and home and remote interfaces. You can do this by selecting them in the **Model** tab of the Explorer and choosing **Delete** from the right-click menu.

Customizing the default EJB code

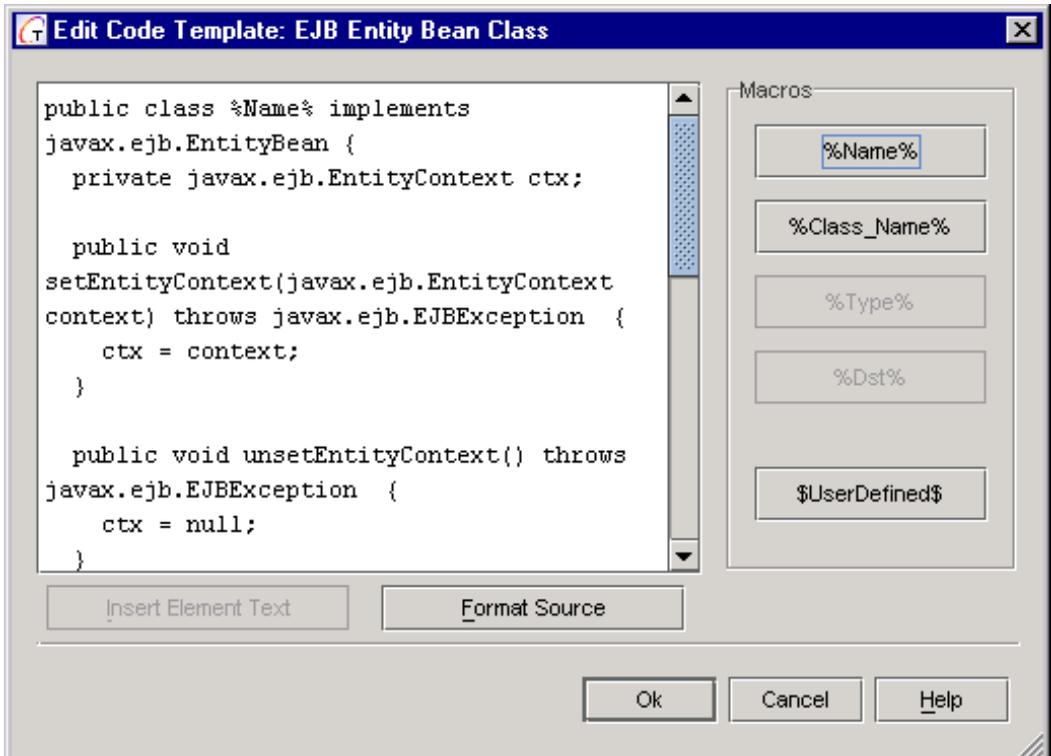
You can customize the skeletons that Together generates when you create an EJB by editing their code templates. There are templates for entity bean, session bean, and message-driven bean classes, primary key class, and home, remote, local home, and local interfaces.

To edit an EJB code template:

1. Choose **Tools | Code Template Expert** from the main menu.
2. On the resulting dialog box, choose *Java* as the *Template Language* and *Class* as the *Template Category*. Click **Next** to continue.
3. In the list of template folders on the left pane, expand *EJB10*, *EJB11*, or *EJB20* to edit EJB code satisfying the EJB 1.0, EJB 1.1, or EJB 2.0 specifications respectively. Choose the template to edit from the items shown and click **Next** to continue.
4. Click **Edit Template Code** to edit the template.
5. Edit the code and click **Ok** when you finish.

Figure 164 shows the dialog box for editing the code for an entity EJB satisfying EJB 2.0 specifications.

Figure 164 Editing the entity bean class template with the Code Template Expert



Assembling EJB Modules

Many multi-tier distributed applications use Enterprise JavaBeans (EJBs) to implement business logic. Together provides a means of visually modeling the assembly of EJB components into EJB modules, which can be subsequently archived into JAR files and deployed directly to application servers.

This chapter discusses assembling EJB modules on special EJB assembler diagrams. The chapter includes the following topics:

- “Visual assembly of EJB applications” on page 591
- “Creating and modifying EJB modules” on page 592
- “Displaying EJB shortcuts on EJB assembler diagrams” on page 594
- “EJB verification and correction” on page 597
- “Deploying an EJB module” on page 598

Visual assembly of EJB applications

Together provides special *EJB assembler diagrams* for modeling EJB applications. An EJB assembler diagram represents a single EJB module. An EJB module is a logical grouping of EJBs along with a corresponding deployment descriptor.

Note The Together EJB assembler diagram is a visual equivalent of the *Application Assembler* described by the EJB 2.0 Specification. The Together J2EE Deployment Expert corresponds to the *Application Deployer*. Refer to page 432 of the EJB 2.0 Specification for details.

An EJB assembler diagram enables you to group several EJBs in a JAR and deploy it to an application server.

Creating and modifying EJB modules

Creating an EJB module in Together is the same as creating an EJB assembler diagram. This section discusses creating EJB assembler diagrams and their elements.

Creating an EJB module

You can create an EJB assembler diagram (EJB module) using the Object Gallery.

To create an EJB assembler diagram:

1. Choose **File | New** on the main menu to open the Object Gallery.
2. On the left pane, choose *Enterprise*.
3. On the right pane, click *EJB Module*.
4. Click **Next**.

The resulting dialog box is an expert that guides you through creating and populating an EJB module.

5. Fill in the text fields.
 - **EJB module name.** The name of the new EJB assembler diagram.
 - **Display name.** The EJB module name for the deployment descriptor.
 - **EJB module location:** The folder for the module.
6. Click **Add from Project** to open the *Select EJBs* dialog.
7. On the left pane, open each folder containing EJBs. CTRL+click each bean class that you want to add to the module.
8. Click **Add** and then click **Ok** to close the dialog.
9. Repeat the same process to add application clients to the EJB module.
10. Click **Finish**.

The resulting EJB assembler diagram is an EJB module that contains shortcuts to all of the EJBs that you selected.

Note An EJB assembler diagram contains only EJB shortcuts. The actual EJBs belong on class diagrams instead of EJB assembler diagrams.

Placing elements on an EJB assembler diagram

The items on an EJB assembler diagram include shortcuts to EJBs, references, method permissions, security roles, principles, and links. EJB assembler diagrams enable you to refine the architecture of an EJB application by linking appropriate diagram elements together. For example, you can link method permissions to security roles or EJBs to resources.

The Designer toolbar at the left side of the diagram has buttons for most of the key EJB assembler elements. [Table 70](#) lists the special buttons.

Table 70 Special EJB assembler diagram design elements

Button	Description
	EJB Shortcuts: Adds or removes shortcuts to EJBs.
	Security Role: Defines security roles for EJB clients. (See Chapter 39, “J2EE Platform Security Support” on page 657.)
	Principal: Creates a user or a group of users separate from their security roles.
	Method Permission: Shows the required permissions on one or more methods.
	EJB Properties: Creates a separate EJB properties element. (Use this to represent EJB deployment properties separate from the EJB or for changing these properties.)
	Container Transaction: Defines transaction attributes for the methods of EJB home and remote interfaces. (See Chapter 34, “Container Transactions and EJB References” on page 601.)
	EJB Reference: Creates a reference to a corresponding EJB. Defines a remote type for EJB reference: EJB, or EJB local (for EJB 2.0 specification).
	Environment Reference: Creates an environment reference.
	Resource Reference: Creates a reference to a resource or resource-environment (for EJB 2.0 specification).
	Bundled Files: Creates a set of bundled files (such as Web files).
	Message Web Service: Creates a message Web service for BEA WebLogic 6.1,7.0 and Borland Enterprise Server. (See Chapter 41, “Creating Web Services” on page 705.)

You can access the properties of any element on an EJB assembler diagram by using its inspector. The inspectors vary according to the type of element.

Setting properties of an EJB assembler diagram

You can use the EJB assembler diagram inspector to set the diagram properties. To open the inspector, right click on the diagram background and choose **Properties**.

The tabs in the inspector for an EJB assembler diagram include the ordinary class diagram tabs as well as an **EJB** tab. The **EJB** tab lists properties used during the deployment process:

- **Module name.** Name of an archive *.jar file that is created during the deployment process. By default this name is the same as the *Name* on the **Properties** tab.
- **Display name, Small icon, Large icon.** Optional parameters for the deployment descriptor.

Displaying EJB shortcuts on EJB assembler diagrams

EJB assembler diagrams contain shortcuts to EJBs. You can place shortcuts on an EJB assembler diagram when you first create it using the Object Gallery. You can also add them to the diagram later by using the toolbar or the diagram right-click menu.

Creating EJB shortcuts

A shortcut is a node for an element that resides somewhere else in the project.

To create an EJB shortcut on an EJB assembler diagram:

1. Open the EJB assembler diagram.
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. Click the **EJB Shortcuts** button from the toolbar. This opens a dialog for selecting shortcuts.
4. In the dialog, expand the *Model* node and locate the EJB classes and interfaces from the project that you want to display in the diagram. CTRL+click the desired classes and interfaces on the left and click the **Add** button.
5. In the 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. CTRL+click them on the left and click the **Add** button.
6. Click **Ok** to add the selected EJB shortcuts on the diagram.

The EJB assembler diagram displays the shortcuts to the items that you selected. The shortcuts retain all of the characteristics of the actual items. For example, if the original item has a hyperlink, the item in the corresponding shortcut also has the hyperlink.

If you modify a shortcut on the diagram, the modifications apply to the actual bean as well with one exception. Deleting a shortcut removes it from the diagram; deleting the shortcut does not remove the actual bean.

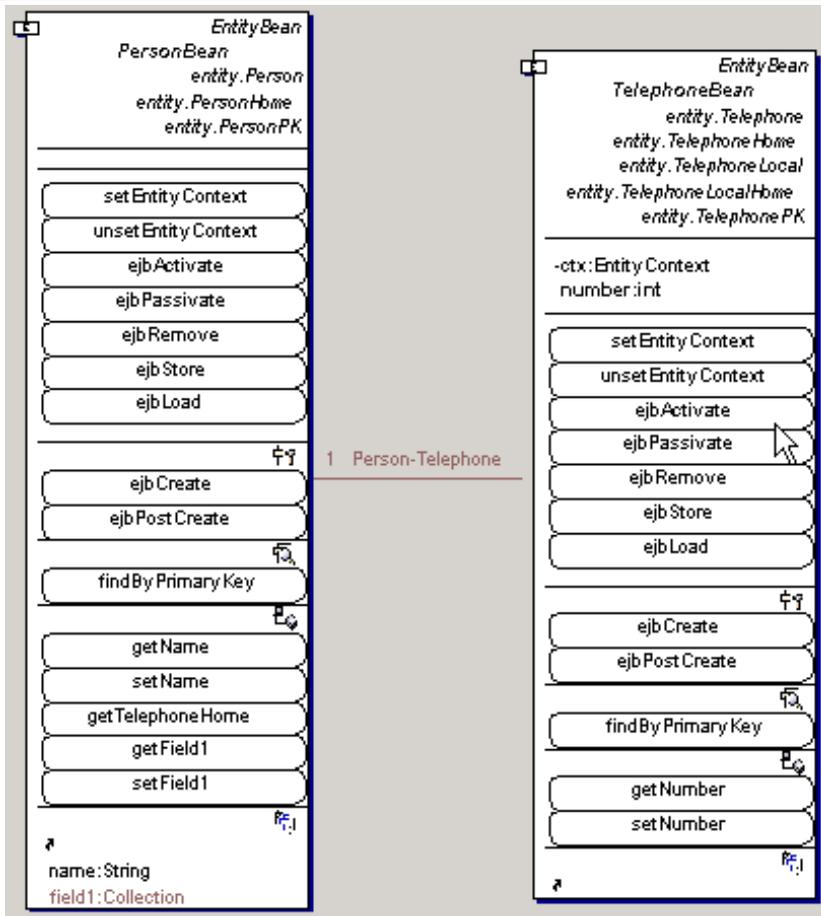
Displaying EJBs on EJB assembler diagrams

The display of an EJB shortcut on an EJB assembler diagram is similar to the display of an EJB component on a class diagram. The main visual distinction is that in an EJB assembler diagram, the methods have elliptical borders.

Figure 165 shows shortcuts for two entity nodes on an EJB assembler diagram with the following features:

- A container-managed relationship named `Person-Telephone` links the nodes.
- The `PersonBean` has hidden attributes. (To hide attributes, right click the bean shortcut and choose **Hide | Attributes**.)
- The fields for the `PersonBean` show at the bottom of the node. (This is the result of hiding the attributes.)

Figure 165 EJB shortcuts on an EJB assembler diagram.



Showing and hiding EJB component parts

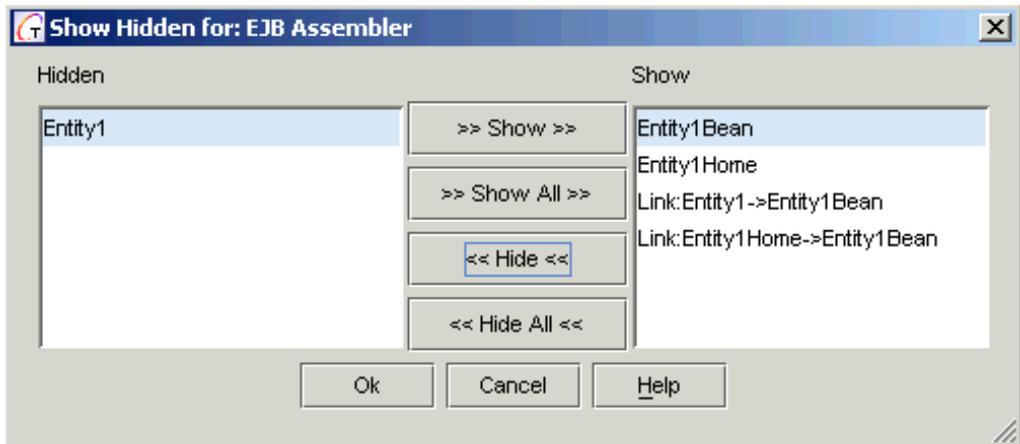
When you create EJB shortcuts, you can choose interfaces or primary key classes as shortcuts. If they do not appear on the EJB assembler diagram, the *View Management* option settings are hiding them. For instructions on how to change those settings, see “[Setting component editing and display options](#)” on page 562.

You can hide or show individual elements on the diagram using the **Show Hidden** command from the right-click menu.

To show or hide elements on an EJB assembler diagram:

1. Right-click the diagram background and choose **Show Hidden**.
2. In the resulting dialog box (shown in [Figure 166](#)), choose the hidden elements on the left that you want to see in the diagram. Then click **Show**.
3. Choose the shown elements on the right that you want to hide and click **Hide**. Click **Ok** to quit the dialog and change the diagram.

Figure 166 Show or hide elements on a EJB assembler diagram.



How to copy and paste EJB shortcuts

To create a copy of an EJB shortcut on an EJB assembler or Web application diagram:

1. Open the EJB assembler diagram that contains the shortcut you want to copy.
2. Right click the EJB shortcut and choose **Copy**.
3. Open the diagram where you want to place the copy.
4. Right click the diagram and choose **Paste**.

If you paste the shortcut to a different diagram, the new shortcut has the same name as the original. If you paste the shortcut to the same diagram, the copy has a different name from the original. In that case, Together creates the bean class corresponding to the new shortcut on the class diagram.

If you want to cut a shortcut and then paste it on another diagram, right-click the shortcut and choose **Cut**. Click on the selected diagram, and then choose the **Paste** (or **Paste Shortcut**) option from the options of the right-click menu. The mechanism of copying is the same: the objects will be created.

Note A single diagram can contain only one shortcut to a particular object.

EJB verification and correction

Entity and session EJBs must conform to EJB 1.0, EJB 1.1, or EJB 2.0 specifications. Message-driven EJBs must conform to EJB 2.0 specification only. Descriptions of these specifications are beyond the scope of this document and are available on the Sun website at <http://www.java.sun.com>.

Verification overview

Together provides verification for the EJB components in a project including:

- Conformance to the common requirements for all specifications: the correctness of inheritance, home and remote interfaces, testing the primary key type, and so on.
- Conformance to the specific requirements of the project's target application server (EJB 1.0, 1.1, or 2.0). EJB 2.0 verification includes checking conformance for local interfaces and message-driven beans.

Note The syntactic rules for verification are defined in configuration files under `$TGH/modules/com/togethersoft/modules/ejbbase/impl/verification/config`. The rules are organized in three folders corresponding to the EJB specifications. You can add new requirements or create customized syntactic rules by editing the appropriate configuration files.

Using verification and correction

You can verify EJBs from class diagrams or from EJB assembler diagrams. When you verify an EJB, the Message pane displays the results in an EJB Verification tab.

The **EJB Verification** tab displays error messages that are colored blue or red:

- Blue error messages report errors that Together cannot fix.
- Red error messages report errors that Together can fix.

Double clicking an error message highlights the place the error was detected in the Editor.

To verify a single EJB:

1. Right-click on the EJB in the diagram and choose **Verify EJB**.
2. Open the **EJB Verification** tab in the Message pane.
3. Together displays messages for any errors it detected according to the current project specification. Together does not automatically correct the errors.

To correct the errors:

1. Right click one of the error messages colored red.
2. Choose **Correct All**. Together corrects the code and turns the color of the message to blue.

You can use the verification and correction process during deployment of EJB modules. (See [Chapter 40, “J2EE Deployment” on page 669](#).)

Deploying an EJB module

Together provides an expert dialog that simplifies the process of deploying EJB modules. You can run the J2EE Deployment Expert from a class diagram that contains an EJB component or an EJB assembler diagram.

Set deployment properties in the *EJB* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for an EJB module displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see <http://info.borland.com/techpubs/books/bes/pdfs52/>.

To invoke the J2EE Deployment Expert:

1. Open a class diagram containing an EJB or an EJB assembler diagram.
2. From the main menu, choose **Deploy | J2EE Deployment Expert**.

The difference between deploying from a class diagram and deploying from an EJB assembler diagram is as follows:

- **Class diagram:** For “fast track” deployment with default values or deployment prototyping.
- **EJB assembler diagram:** For “full featured assembly information” with control over security and permissions.

Important If you try to deploy the same EJB component to the same application server twice, you will get an exception saying that the application server cannot identify beans with the same names. To deploy an EJB a second time from a different EJB assembler diagram, set different deployment parameters (JNDI and the bean name) for the EJB shortcuts.

For information on using the J2EE Deployment Expert, see [Chapter 40, “J2EE Deployment”](#) on page 669.

Container Transactions and EJB References

The deployment descriptor for an EJB module determines the transaction contexts for EJB methods. The deployment descriptor also determines how EJBs can communicate within the container and gain access to required resources.

This chapter discusses Together's support for both of these concepts. The chapter includes the following topics:

- “Container transaction elements” on page 601
- “EJB references” on page 604

Container transaction elements

Container transactions are the indivisible interactions between containers and databases. Deployment descriptors define transaction behaviors of the methods of an EJB. Their `container-transaction` elements specify *transaction attributes*. Together supports transaction attributes through *container transaction* diagram elements.

Container transaction attribute types

There are six different kinds of container transaction attributes for EJB methods:

- **Required:** (`TX_REQUIRED`) Executes in the context of a transaction. The container uses the current transaction context if one exists. Otherwise it starts a new one.
- **RequiresNew:** (`TX_REQUIRES_NEW`) Executes in the context of a new transaction. The container suspends any transaction already in progress until the new one is completed.

- **NotSupported:** (`TX_NOT_SUPPORTED`) Executes outside the context of any transaction. The container suspends any transaction in progress until the method completes.
- **Supports:** (`TX_SUPPORTS`) Executes inside the current transaction if one exists. If there is not a transaction in progress, the method executes without a new transaction.
- **Mandatory:** (`TX_MANDATORY`) Executes inside the current transaction. If there is no transaction, the container throws a `javax.transaction.TransactionRequiredException`.
- **Never:** (`TX_NEVER`) Executes outside the context of any transaction. If the client provides a transaction context, the container throws a `java.rmi.RemoteException`.

A transaction attribute can apply to all the methods of an EJB or it can apply to an individual method.

Creating container transaction elements

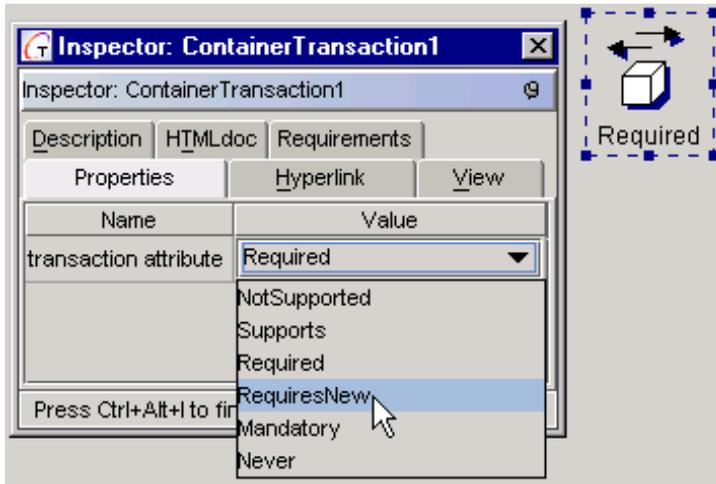
Container transaction diagram elements reside on EJB assembler diagrams.

To create a container transaction element:

1. Open the EJB assembler diagram for the EJB module that you are building.
2. Create a shortcut to the EJB that you want to use. (See [“Creating EJB shortcuts” on page 594.](#))
3. Click **Container Transaction**  on the diagram toolbar and click the diagram. This creates a container transaction element as shown in [Figure 167](#).
4. You can set the value for the transaction attribute through the container transaction element inspector. To access its inspector, right click the element and choose **Properties**.
5. The default value of *transaction attribute* is *Required*. Use the dropdown list to set the value as desired, as illustrated in [Figure 167](#).

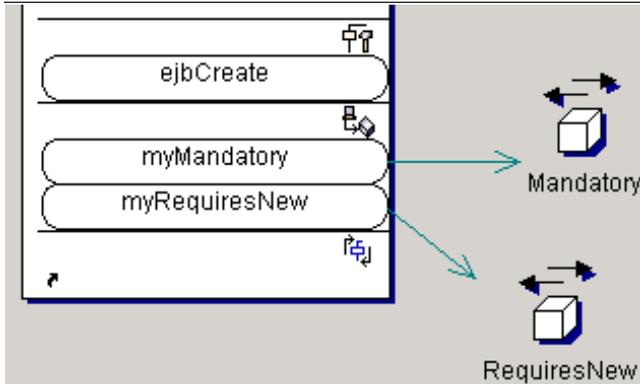
Repeat this process to create as many container transactions as needed for the different EJB methods.

Figure 167 Container transaction element inspector



To associate a container transaction element with a method or with an EJB, draw a link from method or the EJB to the container transaction element. [Figure 168](#) shows links from the methods in a session EJB to two container transactions.

Figure 168 Linking methods to container transactions



If there is a link from an EJB to a container transaction element, then the container transaction element type is the default type for all methods in the EJB. You can override the default for a particular method in the EJB by linking the method to a different type container transaction element.

During deployment, Together generates all necessary descriptions connected with container transaction diagram elements and the corresponding methods for the deployment descriptor. The container transaction elements are added to the JAR of the EJB module (the EJB assembler diagram).

EJB references

One EJB cannot directly call the methods of another. Such bean-to-bean communication requires an EJB reference. References are useful for accessing resources, security, and environment variables as well.

Together provides support for EJB references with two different constructs: *reference attributes* in EJB implementation classes, and *EJB reference elements* on EJB assembler, Web application, and application client diagrams.

Both reference attributes and EJB reference elements translate into deployment descriptor tags. When you initiate the deployment process from an EJB assembler diagram, Together adds all necessary descriptions connected with EJB references to the deployment descriptor (*.xml file) and adds EJB reference elements to the JAR of the EJB module. (This is true as well if the J2EE Deployment Expert starts from Web application, enterprise application, or application client diagrams that contain EJB reference elements. Together adds the EJB references to the WAR, EAR, and application client deployment descriptors.)

EJB reference diagram elements

EJB reference elements enable you to create applications with different combinations of components. A client EJB can get the properties of a referenced EJB through this intermediate EJB reference element. The two EJBs can be in the same JAR file or in different JAR files in the same J2EE module. You can expand the communication among EJBs by creating additional EJB reference diagram elements and redirecting links among EJBs.

An EJB reference element with its own properties provides a flexible connection unit for establishing references between EJBs. You can use the same EJB in different applications without changes. Rather than change the EJB, you merely modify the properties through the corresponding EJB reference element. Indeed, it is possible to create EJB references on a J2EE diagram without having any EJBs in the project.

When you develop a Web application, you can create EJB reference elements instead of creating shortcuts of the actual EJB components to the diagram. Specify properties (home, remote interfaces, and others) in the inspector, and draw a link from the servlet to the EJB reference. This information becomes part of the deployment descriptor.

EJB properties elements

The *EJB properties* element on an EJB assembler diagram enables a client EJB to access the deployment properties of a supplier EJB without specific knowledge of the supplier. You can define EJB deployment properties manually, and link the EJB Properties element to the corresponding shortcut to the EJB in the diagram.

To create an EJB properties element on an EJB assembler diagram:

1. Open the EJB assembler diagram and create a shortcut to the EJB whose deployment properties you want to represent.
2. Click **EJB Properties**  on the diagram toolbar, then click the diagram.
3. Draw a link  from the EJB shortcut to the new deployment properties element.
4. Open the Inspector of the new element (right click and choose **Properties**).
5. Enter the desired values for EJB deployment properties.

Note The fields in the EJB properties element inspector vary according to the EJB that links to it.

If the EJB properties element links to a CMP entity EJB, the inspector shows database connection properties. You can change the database pool name and the name of the table corresponding to the CMP bean.

The EJB properties element provides the flexibility of using the same EJB in different enterprise or Web applications without modifying the EJB code. It is not necessary to change deployment properties such as JNDI names inside the bean. Instead, you can simply create an additional deployment properties element.

You can use the deployment properties element to refer to this EJB at a later time. For example, you can place a shortcut to the deployment properties element on another EJB assembler or Web application diagram. This effectively includes the corresponding EJB in the other EJB module or Web application.

Note When Together deploys an EJB that is linked to a deployment properties element, the information in the deployment properties element overrides the corresponding information in the EJB.

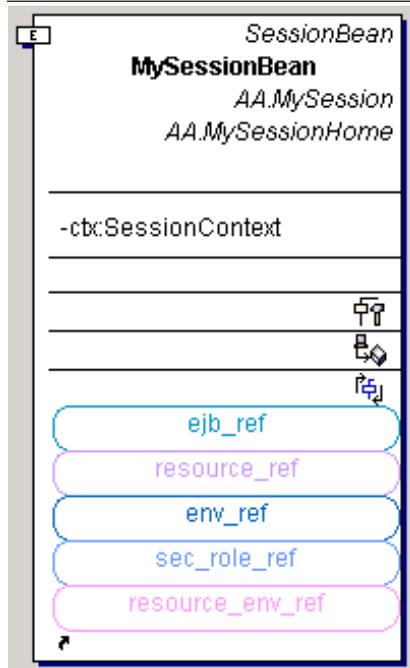
Reference attributes in the client

You can put a reference attribute in the implementation class of the EJB client.

To create a reference attribute in an EJB client class:

1. Right click the EJB on the class diagram (or on a shortcut to the EJB on an EJB assembler diagram) and choose **New** | **<reference type>**. [Figure 169](#) shows different types of reference attributes with different colors, which display in the bottom compartment of the implementation class node.
2. The new reference attribute is a `String` by default. You can change its name and type with the in-place editor.

Figure 169 EJB references in a shortcut to a session EJB.



Supported EJB reference types

Table 71 displays the types of reference attributes in EJB classes. The right column shows common deployment descriptor tags corresponding to each type.

Table 71 Supported EJB reference attribute types

Reference attribute type	Restriction	Deployment descriptor tag
EJB reference		<ejb-ref>
EJB local reference	EJB 2. 0	<ejb-local-ref>
EJB resource reference		<resource-ref>
EJB resource-environment reference	EJB 2. 0	<resource-env-ref>
EJB environment reference		<env-entry>
EJB security role reference	Entity or session EJB	<security-role-ref>

Reference attribute inspectors

You can adjust the properties of the reference attribute through its inspector (right click the attribute and choose **Properties**). The inspector for each type of reference has a special reference tab (EJB Reference tab, EJB security role tab, and so on) that shows the deployment-critical properties.

The fields on the inspector reference tabs vary according to the reference attribute type as follows:

- Common for all types of reference attributes:
 - **Name:** Name of the reference attribute in the EJB implementation class.
 - **<reference type> name:** Name of the reference in the deployment descriptor (in quotes). This is the same as the initial value of the attribute, which is a `String`.
- EJB reference and EJB local reference:
 - **Remote type:** `local` or `remote`.

Note EJB references and EJB local references differ only in the choice of the *Remote type* field in the reference inspector. Choose `EJB-REF` if you are using a remote interface; choose `EJB-LOCAL-REF` if you are using a local interface.

- **Reference type:** the type of the supplier EJB (*Entity* or *Session*).
- **Remote/Home interface:** Remote/home interface of the supplier EJB.
- **EJB link:** Name of the supplier EJB.
- **EJB JNDI name:** JNDI of the supplier EJB.
- EJB resource reference and EJB resource environment reference:
 - **Resource type:** `local` or `remote`.
 - **Resource JNDI name:** Actual JNDI name of the resource.
 - **ResourceRefType:** Connection factory type (`javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.sql.XADataSource`, `javax.mail.Session`, `java.net.URL`).
 - **ResourceRefAuth:** Reference authorization (*Container* or *Application*). *Application* is required when the resource type is local.
- EJB environment reference
 - **Environment entry type:** `Boolean`, `Integer`, `String`, `Double`, `Float`, `Byte`, `Short`, or `Long`.
 - **Environment entry value:** Value of the environment variable.
- EJB security role reference (See [Chapter 39, “J2EE Platform Security Support” on page 657.](#))
 - **Role link:** link to security role in the deployment descriptor.

Changing the properties in the inspector modifies the actual EJB code.

EJB reference diagram elements

EJB reference diagram elements on EJB assembler, Web application, and application client diagrams provide the linking between the EJB client and the EJB supplier.

Changing the EJB reference element inspector properties does not modify any code in the EJB component.

To model a client EJB and supplier EJB on in an EJB module:

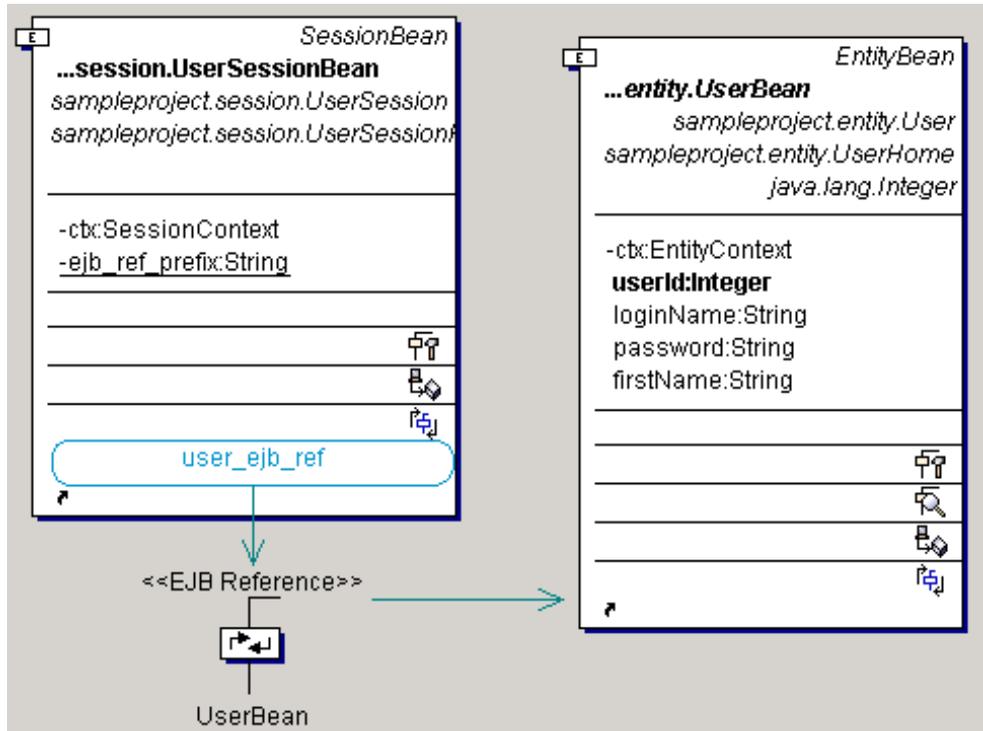
1. Open the EJB assembler diagram corresponding to the EJB module.
2. Create shortcuts to the client and supplier EJBs on the diagram. (Click  and choose the client and supplier from the resulting selection manager dialog.)
3. Right click the client EJB and choose **New | EJB reference**. A new reference named `ejb_ref` appears inside the shortcut to the client EJB (also inside the client EJB on the class diagram).
4. Click , then click the diagram. Edit the name of the EJB reference element on the diagram as desired.
5. Draw a link  from the client EJB shortcut to the EJB reference element.
6. Draw a link from the EJB reference element to the supplier EJB shortcut.

[Figure 170](#) shows an EJB assembler diagram in which `UserSessionBean` is the client EJB and `UserBean` is the supplier EJB. The client has a reference attribute named `user_ejb_ref`. There is a link from the reference attribute to an EJB reference element, which is also named `UserBean`. There is also a link from the EJB reference element to the supplier.

Note Suppose A, B, and C are two enterprise beans, with the class diagram showing an EJB reference from A to B. Suppose also that there are shortcuts to A and C on an EJB assembler diagram. An intermediate EJB reference element on the EJB assembler diagram can redirect the link from A to C.

If you start the J2EE Deployment Expert with the class diagram in focus, the deployment descriptor makes the reference from A to B. If you start the J2EE Deployment Expert with the EJB assembler diagram in focus, the reference is from A to C instead.

Figure 170 Linking clients to suppliers via references on EJB assembler diagrams



EJB resource and resource environment reference diagram elements

Resource references give EJBs and servlets access databases, mail sessions, JMS objects, and URLs. Environment references are named values that are visible to enterprise applications and Web applications.

You can create a resource reference as a resource attribute or a separate *resource* diagram element. The diagram element has all the properties of the referenced resource (for example, a database object).

To create an EJB resource reference on an EJB assembler diagram:

1. Create a shortcut to the EJB shortcut on the EJB assembler diagram. (Click  and choose the EJB from the resulting selection manager dialog.)
2. Right-click on the EJB shortcut and choose **New | EJB resource reference**. This creates a reference attribute named `resource_ref` inside the bean. Right-click the attribute and choose **Properties** to specify its properties. If you are working with a remote interface, choose `remote` as the resource reference type.
3. Click **Resource**  on the diagram toolbar, then click the diagram to create a resource element named `Resource1`.

4. Draw a link  from `resource_ref` to `Resource1`.
5. Right click `Resource1` and choose **Properties** to open the inspector. You can change the properties of a resource simply by changing the properties associated with the EJB resource design element. You need not make any changes to the properties of the actual EJB.

To create a resource-environment reference, follow the steps outlined above, choosing **New | EJB resource-environment reference** from the right-click menu of the EJB shortcut. The resource type for a resource-environment reference is local rather than remote.

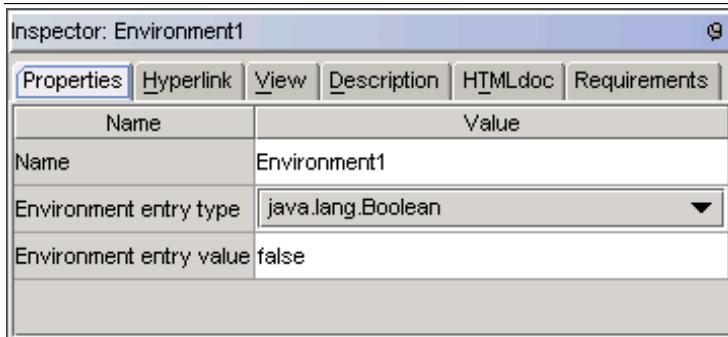
Creating EJB environment references

EJBs can use environment references to access named, static constants. Each environment reference has a type, value, and name. In *Together*, you can define an environment reference as a reference attribute or as a separate *environment* diagram element with its own properties.

To create an EJB environment reference on an EJB assembler diagram:

1. Create a shortcut to the EJB shortcut on the EJB assembler diagram. (Click  and choose the EJB from the resulting selection manager dialog.)
2. Right-click on the shortcut and choose **New | EJB environment reference**. This creates a reference attribute named `env_ref` inside the bean. Right-click the attribute and choose **Properties** to specify its name, type, and value.
3. Click **Environment**  on the diagram toolbar, then click the diagram to create an environment element named `Environment1`.
4. Right click `Environment1` and choose **Properties** to open the inspector. Change the values on the EJB environment reference tab. [Figure 171](#) illustrates how to customize a Boolean environment entry to `true`.

Figure 171 Environment reference properties



You can link an EJB or an EJB environment reference attribute to an environment element on an EJB assembler diagram. Linking an environment resource element enables you to change the environment properties without making any changes to the code of the EJB.

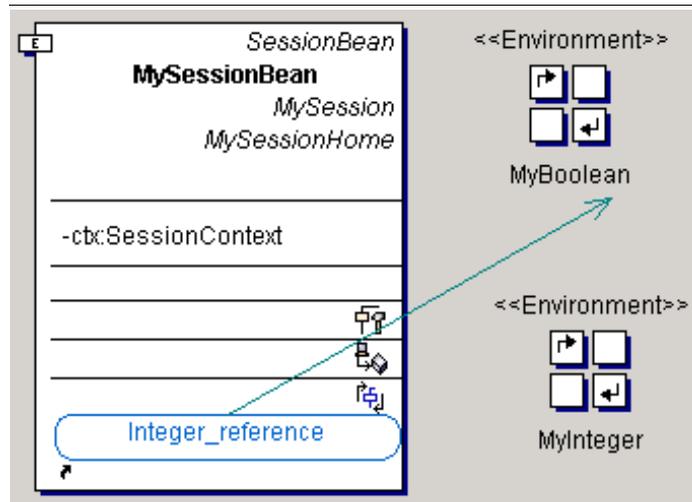
To change the type of the environment variable on an EJB assembler diagram:

1. Add a new environment element  to the current EJB assembler diagram.
2. Define its properties through its inspector (right click the environment node in the diagram and choose **Properties**).
3. Draw a link  from the environment reference attribute in the EJB to the new environment reference node on the diagram.

Figure 172 shows an environment reference attribute named `Integer_reference` which is linked to an environment reference node named `MyBoolean`. If you start the J2EE Deployment Expert from this diagram, `Integer_reference` translates to the deployment descriptor with `MyBoolean` as a “lookup” name and the type as assigned to `MyBoolean` in its inspector.

Since `MyInteger` is not linked to an EJB reference, it deploys as a separate environment variable.

Figure 172 Environment references



If the properties of the environment reference diagram element and the environment reference attribute in the EJB are different and you start the deployment from the EJB assembler diagram, the J2EE Deployment Expert uses the properties of the diagram element to determine what to write to the deployment descriptor.

Designing and Developing Web Applications

Together provides a special Web application diagram for creating and building the Web application archives (WAR files) that store all JSPs, servlets, EJBs, and taglibs in a Web application. This chapter discusses the details of using Web application diagrams to design and develop Web applications.

The chapter includes the following topics:

- “Working with Web application diagrams” on page 613
- “Developing applets” on page 620
- “Working with servlets” on page 622
- “Working with JSPs” on page 624
- “Working with tag libraries” on page 627
- “Working with filters” on page 631
- “Working with listeners” on page 632
- “Deploying a Web application” on page 634

Working with Web application diagrams

Together’s Web application diagram enables you to model the assembly of JSPs, servlets, filters, and other web files in a Web application. From a Web application diagram, you can generate a deployment descriptor for the application, assemble Web components to a *.war archive, and deploy it to an application server.

This section explains how to create Web application diagrams and the basic techniques for working with them.

Creating a Web application diagram

You can create a new Web application diagram from the Designer pane toolbar or by creating a Web application in the Object Gallery.

To create a new Web Application diagram from the Designer toolbar:

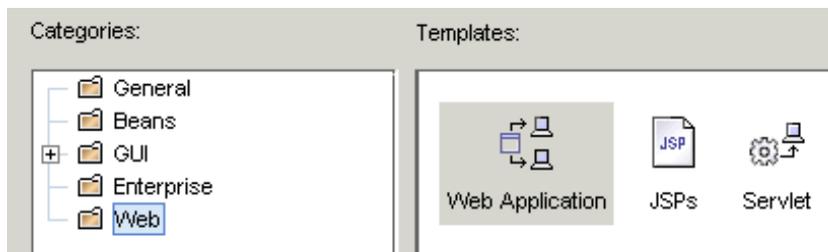
1. Click the **New Diagram** button  on the Designer toolbar.
2. In the resulting dialog box, click the Together tab.
3. Choose *Web Application diagram*. Fill in the name of the diagram and its package location.
4. Click **Finish**.

As an alternative, you can use the Object Gallery to create a Web application diagram and populate it with existing elements such as JSPs, servlets, and Web files.

To create a Web Application with the Object Gallery:

1. Choose **File | New** on the main menu. This displays the Object Gallery.
2. Choose *Web* from the categories on the left, then choose *Web Application* from the templates on the right, as shown in Figure 173.

Figure 173 Choosing a Web Application from the Object Gallery



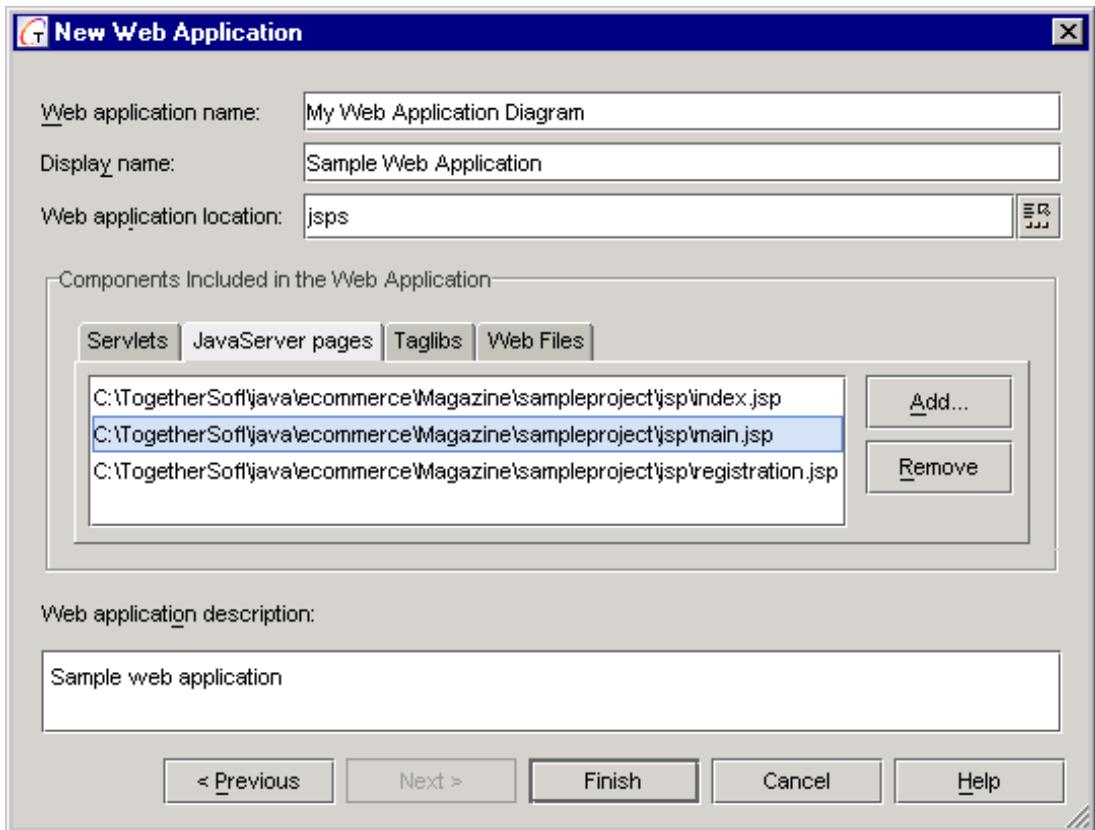
3. Click **Next**. The resulting dialog box is an expert that guides you through the process of creating the Web application, as illustrated in Figure 174.
4. Fill in the New Web Application page in the Expert:
 - **Web application name:** Name of the Web application diagram.
 - **Display name:** (Optional) Name of the application in the deployment descriptor.
 - **Web application location:** Name of the directory for the diagram.
 - **Components Included in the Web Application:** Existing components that you want to include in the application. Components are organized by four tabs: *Server*, *JavaServer pages*, *Taglibs*, and *Web Files*.

Click **Add** to open the file chooser and select JSPs or Web files. Click **Add from project** to open the selection manager and select servlets or TagLibs. To remove an item from the Web application, choose it from the list and click **Remove**.

- **Web application description:** (Optional) Diagram documentation.

5. Click **Finish** to close the dialog and complete the process.

Figure 174 Web application expert



Web application diagram elements

The special Web application design elements are buttons on the diagram toolbar.

Table 72 Design elements on a Web Application diagram

Button	Description
	Servlet Element: Creates a separate deployment properties element. (Use this to represent servlet deployment properties separate from the servlet element or for changing these properties.)

Table 72 Design elements on a Web Application diagram (continued) (continued)

	EJB Shortcuts: Opens a selection manager to add or remove Web shortcuts (to EJBs, servlets, filters, and so on).
	Security Role: A recommended security role for EJB clients; used in the J2EE Deployment Expert. (See Chapter 39, “J2EE Platform Security Support.”)
	Principal: A user or a group of users separate from their security role. Note: It is possible to change the stereotype using in-place editing.
	Security constraint: Intended protection of the Web contents. Security constraints include Web resource collections, authorization constraints, and user data constraints. (See Chapter 39, “J2EE Platform Security Support.”)
	Web resource collection: A 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: A remote type for EJB reference: EJB or EJB local (for EJB 2.0 specification). (See Chapter 34, “Container Transactions and EJB References”.)
	Environment Reference: A static constant, which cannot be changed after deployment.
	Resource Reference: Properties of a referenced resource. Define the type of reference: resource or resource-environment (for EJB 2.0 specification).
	JSP: JSP or servlet.
	JSP Collection: Opens the selection manager to choose multiple JSPs.
	Web Files: Additional Web files such as images.
	Filter mapping: For mapping servlets to filters. Link a servlet to this element and then map it to a filter. (Refer to the Servlets 2.3 specification for details.)
	Error page: A custom error page displayed by the application server when user authentication fails. You can configure an application server to display custom Web pages or other HTTP resources in response to particular HTTP errors or Java exceptions. If the diagram has an Error page element, the deployment descriptor (web.xml) contains a corresponding error-page element, which maps error codes or exception types with the resource path in the Web application.
	TagLib: JSP tag library. (This tag library must already exist.)

Servlet element

A servlet element enables to access the deployment properties of a servlet. You can define servlet deployment properties manually, and link a servlet shortcut with the servlet element.

To create a servlet element on a Web application diagram:

1. Open a Web application diagram and create a shortcut to the servlet whose deployment properties you want to represent.
2. Click **Servlet Element**  on the diagram toolbar, then click the diagram.
3. Draw a link  from the servlet shortcut to the new servlet element.
4. Open the Inspector of the new servlet element (right click, and choose **Servlet Properties**).
5. Enter the desired values for *Servlet name*, *Load on startup*, *Servlet initialization parameters*, and *Servlet mapping*.

Note The fields in the EJB properties element inspector vary according to the EJB that links to it.

If the EJB properties element links to a CMP entity EJB, the inspector shows database connection properties. You can change the database pool name and the name of the table corresponding to the CMP bean.

The EJB properties element provides the flexibility of using the same EJB in different enterprise or Web applications without modifying the EJB code. It is not necessary to change deployment properties such as JNDI names inside the bean. Instead, you can simply create an additional deployment properties element.

You can use the deployment properties element to refer to this EJB at a later time. For example, you can place a shortcut to the deployment properties element on another EJB assembler or Web application diagram. This effectively includes the corresponding EJB in the other EJB module or Web application.

Note When Together deploys an EJB that is linked to a deployment properties element, the information in the deployment properties element overrides the corresponding information in the EJB.

Web application diagram properties

You can access the properties of a Web application diagram through its Inspector - right click the background of the diagram and choose **Properties**. The Inspector has a special **Web Properties** tab, which is organized into three additional tabs at the bottom of the Inspector: **General**, **Login config**, and **App event listeners**.

General tab

The **General** tab contains the following fields:

- **Module name:** Name of the *.war file created during the deployment process.
- **Welcome File List:** (Optional) <welcome-file-list> A list of files to be used for default welcome pages. When the requested URL is a directory, the application server goes through the list, using the first file that it finds for a welcome page.

- **Session time-out:** (Optional) 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 overrides the `<session-descriptor>` element in `web.xml`.
- **Context Root:** (Optional) A string that is used in place of the actual name of the Web application when server resolves a request for the Web application. The server interprets the path name, which must begin with a slash (/), as relative to the context root.
- **Servlet Context Parameters:** (Optional) Defines context initialization parameters of the Web application servlets. The following methods access these parameters:


```
javax.servlet.ServletContext.getInitParameter()
javax.servlet.ServletContext.getInitParameterNames()
```
- **Mime type mapping:** (Required) Defines a mapping between an extension and a mime type. Enter the mime type mappings directly or add them individually using the extended editor button .
- **Display name and Small/Large icon:** (Optional) For GUI tools, the name and location of small (16x16 pixels) /large (32 X 32 pixels) *.gif or *.jpg image.

Login Config tab

The **Login Config** tab contains the Web properties relevant to security. It contains the following fields:

- **Authentication method:** (Optional) Authentication method, with choices NONE, BASIC (browser authentication), FORM (html form), and CLIENT-CERT.
- **Realm name:** (Optional) Realm name referenced for user authentication.
- **Form login page:** `<form-login-page>` (Required if authentication method is FORM) Full pathname of a JSP, HTML page, or servlet that generates an HTML form for user certification.
- **Form error page:** `<form-error-page>` (Required if authentication method is FORM) URI of a page to be used as a login error page.
- **Location form login/error page:** (Enabled only if the authentication method is FORM or CLIENT-CERT) Location of the login page (part of the `form-login-page` element of the deployment descriptor).The path begins with a leading slash (/) and is interpreted relative to this root.

For additional instructions on how to provide security, see Chapter 39, “J2EE Platform Security Support.”

App event listeners tab

The App event listeners tab contains four listener fields: *Servlet context listener*, *Servlet context attribute listener*, *Http session listener*, and *Http session attribute listener*.

For instructions on using the app event listeners tab, see “Associating listeners with Web applications” on page 633.

Creating shortcuts to applets, EJBs, EJB properties, container transactions, and references

Web application diagram cannot contain EJBs. You can, however, place an EJB shortcut on the diagram. During deployment, EJBs are included in the WAR file as ordinary classes.

Note You should use EJB modules (EJB assembler diagrams) to deploy EJBs. Information on EJB modules is in Chapter 33, “Assembling EJB Modules.”

If servlets or other elements of your Web application use EJBs, you must place shortcuts to the home and remote interfaces of the EJBs on the Web application diagram. If servlets or other elements of your Web application use EJB properties, container transaction elements, or references, you must place the corresponding design elements (or their shortcuts) on the Web application diagram.

To create a shortcut to an applet, EJB, EJB interface, EJB property, container transaction element, or reference on a Web application diagram:

1. Right-click the background of the Web application diagram and choose **New | Shortcuts**. This opens a selection manager dialog.
2. In the selection manager, choose the elements you want to add from the available content listed on the left and click **Add**. To remove an element from the existing or ready to add elements listed on the right, choose it and click **Remove**.
3. Click **Ok** to close the selection manager and place the shortcuts on the diagram.

For more information on how to work with EJB properties, container transaction elements, and references, see Chapter 34, “Container Transactions and EJB References”.

After you populate a Web application diagram with the appropriate components, you can open the J2EE Deployment Expert to generate the deployment descriptor and deploy the application a server. The J2EE Deployment Expert places all of the elements on the Web application diagram into a single WAR file. For more details on deploying and the J2EE Deployment Expert, see Chapter 40, “J2EE Deployment”.

Developing applets

Applets are not a part of J2EE technology. However, Web application diagrams may contain shortcuts to applets.

Creating an applet

The first step in creating an applet consists of creating or opening a project for a listener.

To create an applet:

1. Create a project for a listener or use an existing project.
2. To create a new applet, click **Class by Pattern** from the diagram toolbar.
3. In the resulting dialog, choose the *Applet* pattern (*UI Components - Applet*).
The applet pattern contains the standard set of methods: `init()`, `start()`, `stop()`, `destroy()`, `getAppletInfo()`, `initGUI()`.
4. Enter a name for the applet and click **Finish**.
5. Edit the source code to add the required functionality. Add the code to the skeleton in place of the green comments “// Write your code here.”
6. Correct the properties of your applet using the Inspector (choose **Properties** on the applet’s right-click menu).

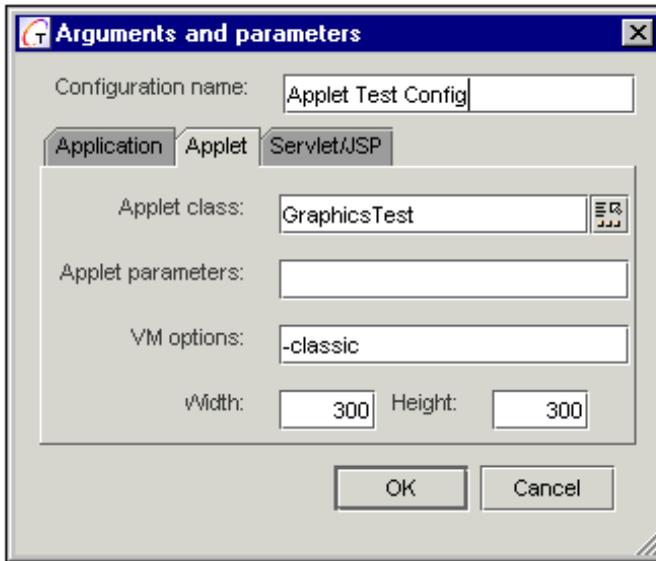
Running and debugging an applet

You can run and debug an applet directly from the main menu.

To debug an applet:

1. Make sure that `TGH/lib/openapi.jar` is in the project Search /Classpath.
2. From the main menu, choose **Run** (or **Debug**).
3. In the resulting dialog box, click the **Applet** tab.

Figure 175 Arguments and parameters dialog for running an applet.



4. Enter the name of the applet class or choose one using the browse button.
5. Enter parameters, using the format "name1"="value1" "name2"="value2" and so on.
6. Make sure the VM options include -classic.
7. Click **OK** when ready. The Debugger (or Runner) pane opens with the applet frame.

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 to open the file chooser dialog.
3. Place the shortcut of the applet on the Web application diagram.
4. Add other classes if necessary.
5. Invoke the J2EE Deployment Expert (see Chapter 40, "J2EE Deployment"). Continue the deployment process as instructed in the Expert.

Working with servlets

A servlet is a Java class on the server side that handles client-server interactions. Usually, these are http-based interactions between Internet browsers and Web servers. Any application capable of creating http-requests can act as a client.

A single servlet can respond to one or 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.

Note When you add a servlet to a project, you may need to add `TGH/bundled/tomcat/lib/servlet.jar` to the Searchpath/Classpath in the project properties.

Creating a servlet

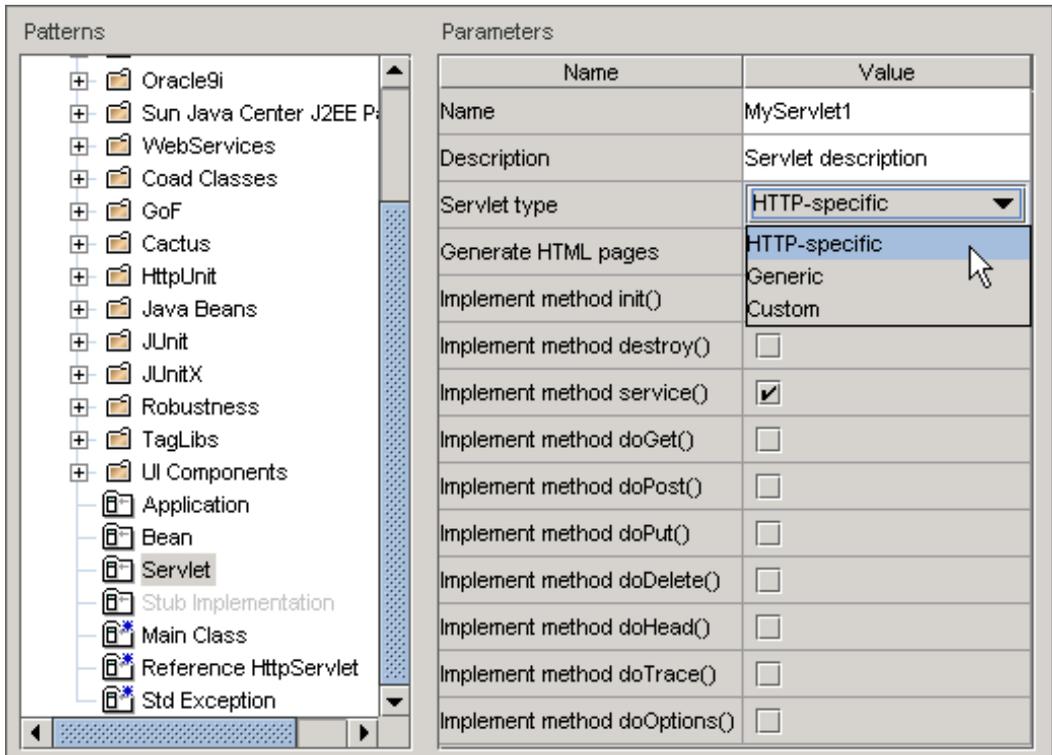
Together offers two convenient ways to create servlets: 1) apply a servlet pattern to a new class or to an existing class, and 2) use the Object Gallery.

Servlets reside on class diagrams. A servlet shortcut can belong to a Web application diagram.

To create a serolet on a class diagram by using a pattern:

1. Open the class diagram where you want to place the servlet.
2. Click the **Class by Pattern** button on the diagram toolbar. This opens the dialog shown in Figure 176.
3. From the list of *Patterns* on the left pane, choose *UI Components - Servlet*.
4. In the Properties on the right pane, enter the servlet *Name* and *Description*. Choose the *Servlet type* from the dropdown list:
 - **Http-specific servlet:** Extends `HttpServlet`.
 - **Generic servlet:** Extends `GenericServlet` and services any request-response protocol.
 - **Custom Servlet:** Implements the `Servlet` interface.
5. Check the methods that you want generated in the body of the class.
6. If desired, check *Generate HTML pages* to add the code to the request-processing methods (`service`, `doGet`, `doPost`, `doPut`, `doDelete`) to return an empty HTML page.
7. Click **Ok** to close the dialog and generate the servlet.

Figure 176 Creating a new servlet by pattern



To create a servlet using the Object Gallery

1. Open the class diagram where you want the servlet to belong.
2. On the main menu, choose **File | New**. This opens the Object Gallery.
3. In the Object Gallery, choose *Web* from the Categories on the left pane. Then choose the *Servlet* icon from the Templates on the right pane.

The resulting *New Servlet* page is an expert that guides you through the process of creating the servlet.

4. Enter the name of the new servlet and its location (the full path).
5. Optionally, to create a shortcut to the servlet on a Web application diagram, check *Create a shortcut to the Servlet class on a Web Application diagram* and enter the name of a new diagram or select an existing Web application diagram.
6. Click **Finish** to create the servlet and the optional shortcut.

The resulting servlet extends `HttpServlet`. The servlet has two methods: `getServletInfo` and `service`.

Running and debugging a servlet

Together uses Jakarta Tomcat, the Reference Implementation for the Java Servlet 2.2 (or higher), to debug servlets.

To run and debug a servlet:

1. Make sure that `TGH/bundled/tomcat/lib/servlet.jar` is in the Search/Classpath of your project.
2. Choose the servlet you want to debug on the class diagram. In the Editor, set breakpoints as desired.
3. From the main menu, choose **Run | Debug**.
4. In the resulting dialog, choose the **Servlet/JSP** tab.
5. Enter the name of the servlet, or choose one using the browse button. Make sure that the *VM options* include `-classic`.
6. Click **Ok**.

You can observe the results in the Debugger pane and in the browser window. For more information on debugging in Together, see Chapter 18, “Debugging”.

Putting a servlet into a Web application

The first step in putting a servlet in a Web application is to open the corresponding Web application diagram or create a new one if none exists.

To put a servlet on a Web application diagram:

1. Create a Web application diagram or open an existing one.
2. Click the **Add Shortcut** button on the toolbar.
3. In the left pane, navigate to find the servlet. Choose the servlet and click **Add**, then click **Ok**.

Together includes the servlet among the Web components when it deploys the diagram to an application server.

Working with JSPs

JSP pages provide a presentation layer for Web applications by combining static HTML with dynamically-generated HTML. JSP pages compile into servlets at runtime. A JSP file is not pure Java code; it contains tags and Java scriptlets that can generate an HTML page.

Placing a JSP on a Web application diagram

Web application diagrams can contain JSP elements.

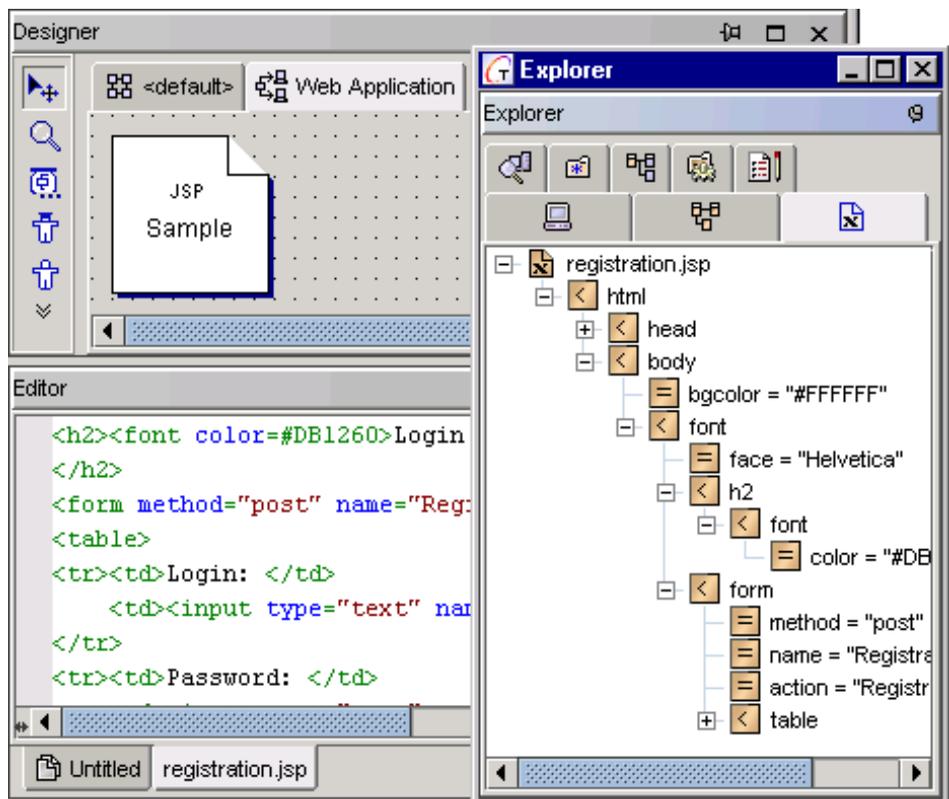
To add an existing JSP to the Web application:

1. Click the JSP button  on the diagram toolbar or right-click the diagram background and choose **New | JSP**. This creates a JSP element on the diagram.
2. Right-click the JSP in the diagram and choose **Properties** to open its Inspector.
3. In the **JSP Properties** tab of the Inspector, enter the pathname of the existing *.jsp file or click the file browser to find it.

When you click on the JSP element in the diagram, The corresponding *.jsp file opens in the Editor and in the Explorer. The Explorer reveals the underlying file structure.

Figure 177 shows the display of a JSP whose diagram node name is `Sample` and whose actual file name is `registration.jsp`. The Explorer pane is undocked.

Figure 177 JSP display in the diagram, Editor, and Explorer



To add several JSPs to the Web application diagram at the same time:

1. Click the JSP collection button  on the diagram toolbar or right-click the diagram background and choose **New | JSP Collection**.

2. The file chooser dialog immediately opens for you to choose which files you want to add. CTRL+Click to choose each *.jsp that you want to include.

The diagram displays a separate node for each JSP chosen.

When you create a JSP diagram element using the Object Gallery, you have the option of creating a new JSP file to correspond to the element on the diagram.

To create JSPs using the Object Gallery:

1. On the main menu, choose **File | New** to open the Object Gallery.
2. Choose *Web* from the categories on the left. Then choose *JSPs* from the templates on the right.
3. Click **Next**. This opens an expert that guides you through creating a JSP.
4. Fill in the fields of the expert.

Creation Scenario: You can import existing JSPs using the file chooser or you can create a new JSP. If you create a new JSP, enter its name and location.

Options: Check *Show JSPs on Web Application diagram* to include the JSPs in a Web application. In that case, you can create a new diagram or select an existing one.

5. Click **Finish**. This creates the new Web Application diagram (or opens the existing Web Application diagram) and places the JSP element on it.

Debugging JSPs

Together provides various ways to debug JSPs. A simple way to run a JSP in debug mode is to choose **Run | Debug** from the main menu. However, this kind of debugging is suitable mainly for servlets rather than JSPs. That is because most JSPs are intended for data presentation only, and they are invoked by a servlet or an HTML page.

A second way of debugging a JSP requires deploying a Web application to an application server. Together provides JSP debugging using Jakarta Tomcat 3.2, the reference implementation for the Java Servlet 2.2 and JavaServer Pages 1.1 Technologies.

To debug a JSP under Tomcat:

1. Open the Web application diagram containing the JSP. Select the JSP that you want to debug to open the JSP in the Editor.
2. Set breakpoints in the JSP as desired.
3. From the main menu, choose **Deploy | J2EE Deployment Expert**.
4. On the first page of the Expert, choose **Generic 1.1** as the application server. Check the following tasks:
 - *Compile classes referenced in the currently selected diagram*

- *Generate deployment descriptors*
 - *Run Web Application under Tomcat in debug mode*
5. On the *Run Web Application under Tomcat in debug mode* page, enter the root folder of the Web application.

Note In order for a JSP to address an EJB, the libraries of the application server where the client will search for the `INITIAL_CONTEXT_FACTORY` class must be in the project classpath. The JSP file should contain lines for the proper handling of the `InitialContext`.

It is 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.

6. Fill in the deployment properties as desired on the remaining pages of the Expert. On the final page, click **Finish** to start the debugging session.

Working with tag libraries

JSP pages have tags for interacting with Java objects residing on the server. These include custom tags, which reside in tag libraries, or taglibs.

When a JSP page with custom tags is compiled into a servlet, the custom tags become operations on server-side objects called tag handlers. Tag handlers must either implement `Tag` or `BodyTag` or extend `TagSupport` or `BodyTagSupport`. There is an additional handler that extends `TagExtraInfo` class.

Together has a special taglib diagram for modeling the handlers in a single tag library. The elements on a taglib diagram are shortcuts to the actual handler classes, which reside on class diagrams. Together also has class patterns for tag handlers.

Note When you add a `TagLib` to a project, you may need to add `$(TGH$)/bundled/tomcat/lib/servlet.jar` to the project Searchpath/Classpath.

Creating tag handlers on class diagrams

You can create tag handlers on a class diagram by applying a tag handler to an existing class or to a new class.

To create tag handler as a new class on a class diagram:

1. Open the class diagram where you want to place the tag handler.
2. Click the Class by Pattern button  on the diagram toolbar.
3. In the resulting dialog box, choose *TagLibs - <handler type>* from the patterns listed on the left.
4. Enter the name of the class in the *Name* field on the right.

- Click **Finish** to close the dialog and create the tag handler class, You can place a shortcut to the tag handler on a taglib diagram if desired.

TagLib diagrams

The Together taglib diagram enables the easy creation of handlers and declaration of the tags in a tag library descriptor.

Diagram elements

Table 73 shows taglib diagram elements. When you place an element on a taglib diagram, Together creates a new class and places the shortcut to the class on the taglib diagram.

Table 73 Main elements of taglib diagrams

Icon	Description
	Implements Tag: Handler for a simple tag without a body.
	Extends TagSupport: Handler for tag with attributes.
	Implements BodyTag: Handler for a tag with a body that contains tags, scripting elements, HTML text, and so on between start and end tags
	Extends BodyTagSupport: Handler for a body tag with attributes.
	Extends TagExtraInfo: provides information to the JSP container about the scripting variable.

Taglib diagram properties

The properties of a taglib diagram are defined on its inspector. You can open the inspector by right-clicking the background of the diagram and selecting **Properties**.

In addition to properties common to all diagrams, the taglib diagram inspector has a **TagLib properties** tab that defines the tag library. This tag includes the following properties:

- **TLD File Name:** Fully qualified name of the tag library descriptor that is generated by the **Generate TLD** command of the diagram right-click menu. You can enter the information in this field directly or select it using the file chooser button.
- **Short name:** A logical name that indirectly references the TLD (mapped to an absolute address in the deployment descriptor).
- **TagLib Version:** Version of the tag library.

- **JSP Version:** JSP specification version used by the tag library.
- **TagLib URI:** URI that uniquely identifies the tag library.
- **TagLib location:** Physical location of the tag library.
- **Use existing TLD:** Checking disables the **Generate TLD** command of the diagram right-click menu
- **Display name/Small icon/Large icon:** Equivalent to the same properties for Web application diagrams. (See “Web application diagram properties” on page 617.)

Properties of the handlers

You can open the Inspector of a tag handler class by right clicking the handler element on the class diagram or taglib diagram and choosing **Properties**. The **Tag properties** tab of the Inspector contains the tag details of the handler. These details include the following:

- **Tag name:** The name of the tag as known by the JSP.
- **Body content:** Possible values are *JSP*, *EMPTY*, and *TAGDEPENDENT*.
- **TagExtraInfo class:** A separate class containing additional information associated with the tag handler.

Creating and using tag libraries

The three basic steps of creating and using tag libraries are as follows:

Step 1: Create and populate the taglib diagram.

Step 2: Generate the tag library descriptor.

Step 3: Add the tag library to the Web application diagram.

To create and populate a taglib diagram:

1. Create the taglib diagram using either the **New Diagram** button  on the Designer toolbar or the Object Gallery. *TagLib* is a diagram type listed on the Together tab.
2. Populate the taglib diagram with the necessary tag handlers (shortcuts).
3. Open the inspector for each diagram element:
 - Fill in the inspector fields on the **Tag properties** tab: tag name, implemented `TagExtraInfo` class, and tag body content (if any).
 - Enter additional information as needed in the **Description** tab.

To generate the tag library descriptor:

1. Right-click on the diagram background to open the taglib diagram Inspector.

2. Click the **TagLib properties** tab on the top then the **General** tab on the bottom of the Inspector.
3. In the *TLD file name* field, enter the path name of the tag library descriptor, or locate it using the file chooser. Fill in the other fields as required.
4. Right-click on the diagram background and choose **Generate TLD**.

To add the tag handlers to the Web application diagram:

1. Place a **TagLib** element  on the Web application diagram.
2. In the resulting dialog, expand the nodes on the left pane and choose the tag library to be added.
3. Click **Ok** to close the dialog and place a shortcut to the tag library on the Web application diagram.

Inserting tags from TagLibs

When you work with JSPs on a Web application diagram, you can insert custom tags.

To insert the tags from your tag libraries into a JSP:

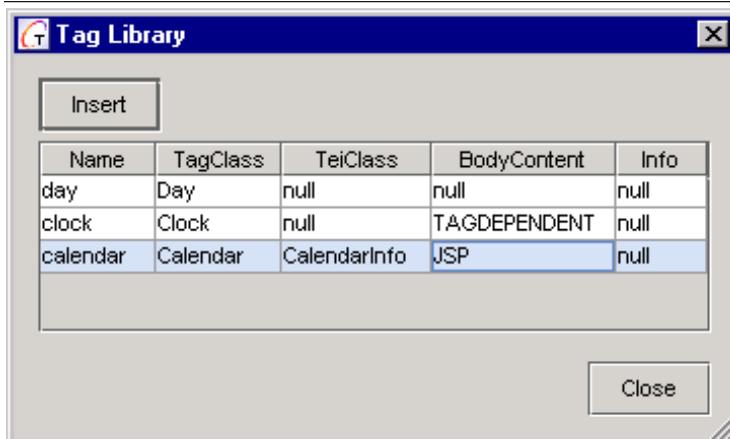
4. Open the JSP in the Editor by clicking its node in the Web application diagram.
5. Enter the `taglib` tag in the JSP between the `<head>` and `</head>` tags as follows:

```
<%@ taglib uri="tdl-file-name" prefix="taglib-prefix"%>
```

where `tdl-file-name` is the name of the taglib descriptor (maps to the location of the taglib jar in the WAR file) and `taglib-prefix` is the custom tag prefix.

6. Right click on the Editor and choose **Tools | Tag Library Helper**. The resulting dialog box lists all of the custom tags in the tag library, as shown in Figure 178.
7. Click the desired tag, then click **Insert** to add the tag to the JSP.

Figure 178 Tag Library Helper dialog



Working with filters

Filters are servlets in Web applications that can modify requests or responses. Filter mappings determine how filters apply to servlets and static Web resources.

Together supports filters and filter mappings through the filter mapping elements on Web application diagrams.

Creating filters and filter mappings

Filters are classes that reside on class diagrams. Web application diagrams can contain shortcuts to filters.

To create a filter or a collection of filters in a Web application diagram:

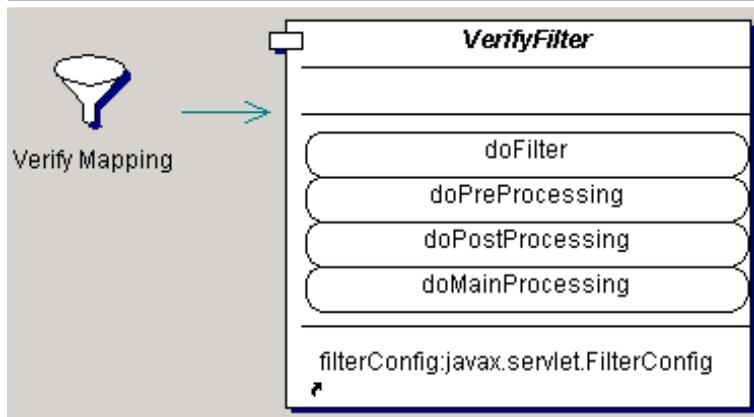
1. Place a **Filter Mapping** element  on the diagram.
2. Open the filter mapping Inspector: right-click the filter mapping element and choose **Properties**.
3. In the Inspector, choose the **Mapped Filters** tab.
4. Click **Add** to create a filter. You can edit its name in the list of filters on the tab.

This process generates a filter class on the default diagram that implements the `javax.servlet.Filter` interface. Each filter can process the current request, the current response, and the `FilterChain` of filters to be executed.

The Web application diagram displays a filter mapping, a shortcut to the filter class, and a Web link connecting the two, as illustrated in Figure 179.

Note When you add filters to a project, you may need to add `javax.servlet.jar` to the Searchpath/Classpath in the project properties. Filter mapping is defined only for application servers supporting EJB 2.0 specification.

Figure 179 Filter mapping on a Web application diagram



Setting filter and filter mapping properties

You can access the inspector of a filter class through its node on a class diagram or through its shortcut on the Web application diagram. Right-click the filter on the diagram and choose **Properties**.

The filter class inspector has a **Filter Properties** tab, where you can specify the following properties:

- **Filter name:** Maps 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:** Filter initialization parameters. Click the chooser button at the right of the field to open a dialog box for adding or removing parameters.

The filter mapping inspector has two special tabs: **Mapped filters** and **URL Pattern Mapping**.

- **Mapped filters:** For adding filters (as described in the previous section) or removing links. The **Remove** button deletes the link between a filter mapping and a filter. It does not delete the filter class and it does not delete the shortcut to the filter class on the Web application diagram.
- **URL Pattern Mapping:** Defines 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*.

Deploying filters

During deployment, Together archives filters to a *.war file and deploys to the application server. For each filter element Together generates the corresponding lines in the deployment descriptor.

Working with listeners

You can monitor a servlet by creating a listener to respond to servlet events. The listener is a class that implements a servlet event listener interface. Servlet event listeners can notify for state changes in `ServletContext` and `HttpSession` objects.

Note Listeners are defined only for application servers supporting the EJB 2.0 specification.

Supported event types

A listener can notify the rest of a system when an event takes place (for example, the Web server shuts down or the application is removed from the Web server).

Together supports the following event types:

- **Servlet Context Events**

- **Lifecycle:** The servlet context has just been created and is available to service its first request or the servlet context is ready to be shut down.

`javax.servlet.ServletContextListener` has these methods:

- `contextDestroyed(ServletContextEvent)`
- `contextInitialized(ServletContextEvent)`

- **Changes to attributes:** Attributes on the servlet context have been added, removed, or replaced.

`javax.servlet.ServletContextAttributesListener` has these methods:

- `attributeAdded(ServletContextAttributeEvent)`
- `attributeRemoved(ServletContextAttributeEvent)`
- `attributeReplaced(ServletContextAttributeEvent)`

- **HttpSession Events**

- **Lifecycle:** An `HttpSession` has been created, invalidated, or timed out

`javax.servlet.http.HttpSessionListener` has these methods:

- `sessionCreated(HttpSessionEvent)`
- `sessionDestroyed(HttpSessionEvent)`

- **Changes to attributes:** Attributes have been added, removed, or replaced on an `HttpSession`.

`javax.servlet.http.HttpSessionAttributesListener` has these methods:

- `attributeAdded(HttpSessionBindingEvent)`
- `attributeRemoved(HttpSessionBindingEvent)`
- `attributeReplaced(HttpSessionBindingEvent)`

Creating a listener

The first step in creating a listener is to determine the class diagram (or the package) for the listener.

To create a listener:

1. Open the class diagram for the listener. Then click **Class by Pattern**  on the toolbar and click the diagram. This opens the Choose Pattern dialog.
2. In the patterns displayed on the left, expand *J2EE - App Event Listeners*.
3. Choose one of the four listener patterns and enter a name on the right.
4. Click **Finish**.

Associating listeners with Web applications

You can include listeners as properties of Web application diagrams.

To specify a listener as a Web application property:

1. Create a Web application diagram or open an existing one. Then open its inspector by right-clicking the diagram and choosing **Properties**.
2. Choose the **Web Properties** tab on the top of the inspector and then choose the **App event listeners** tab on the bottom. The **App event listeners** tab has four fields, one for each type of listener.
3. At the right of the field for the appropriate listener type, click the button  to open a multi-string editor. The resulting dialog displays the listeners of that type.
 - Click **Add** to add a listener. This opens a new line in the listener display. Clicking the button at the right side of a line opens a *Select Element* dialog for choosing the listener. You can add multiple listeners by repeating this process.
 - To remove a listener from the list, choose the listener and click **Remove**.
 - The order that the listeners are listed is the order in which they are processed. To change the processing order, choose a listener and click **Up** or **Down** as needed.
4. Click **Ok** to close the dialog.

The **App event listeners** tab displays the listeners for each type, separating listeners of the same type by commas.

Deploying a Web application

After you populate a Web application diagram with the desired components, you can set its deployment properties in the *Web Properties* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for a Web application displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see <http://info.borland.com/techpubs/books/bes/pdfs52/>.

open the J2EE Deployment Expert by choosing **Deploy | J2EE Deployment Expert** from the main menu. The J2EE Deployment Expert generates a deployment descriptor, creates a WAR file from all of the elements on the Web application diagram, and deploys the application to a server.

For more details on how to use the J2EE Deployment Expert, see Chapter 40, “J2EE Deployment”.

Note You can archive several Web applications and EJB modules in one EAR and deploy it to the current application server. For information, see Chapter 36, “Designing and Developing Enterprise Applications”.

For “fast track” deployment with default values or deployment prototyping, you can use the J2EE Deployment Expert from an appropriate class diagram. For “full-featured assembly information” with control over security and permissions, you should use the J2EE Deployment Expert from the Web application diagram.

Designing and Developing Enterprise Applications

Together provides a special enterprise application diagram to visually assemble enterprise applications for deployment. This chapter discusses designing and developing enterprise applications by using diagrams. The chapter includes the following topics:

- “Visual assembly of enterprise applications for deployment” on page 637
- “Deploying an enterprise application” on page 642

Visual assembly of enterprise applications for deployment

Together provides a special *Enterprise application* diagram for collecting elements for an EAR file into a single diagram. The enterprise application diagram combines Web applications, EJB modules, classes, and connectors. You can use it to visually assemble an enterprise application for deployment.

An enterprise application diagram can contain shortcuts to EJB assembler, Web application, resource adapter, and class diagrams. You can include existing WAR, JAR, and RAR archive files on the diagram as well. An enterprise application 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 (such as BEA WebLogic 5.1 and IBM WebSphere 3.5), multiple *.jar files are generated instead. Otherwise, a single *.ear file is generated when deployment is performed against the enterprise application diagram.

Enterprise application diagrams also support security roles. (See [Chapter 39, “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. The diagram provides application assembly information to the application deployer, which the J2EE Deployment Expert in Together.

Creating 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 for a single enterprise application, you can use this to create an enterprise application diagram that is ready for deployment.

To create a new enterprise application diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *General* in the categories on the left, and choose the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Open the Together tab, choose the *Enterprise Application* icon, and click **Finish**.

Alternatively, you can click **New Diagram**  on the diagram toolbar, go to the Together tab, and select **Enterprise Application**.

At this point, you can develop the content of the diagram using visual design elements from the toolbar.

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 with the Object Gallery:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *Enterprise* in the categories on the left.
3. Choose the *Enterprise Application* icon in the templates on the right.
4. Click **Next**.

The resulting dialog box is an expert that guides you through the process of creating the enterprise application. Follow its recommendations, filling in the most important parameters, inserting names, and choosing paths as needed. 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.

Enterprise application diagram elements

All enterprise application diagram elements are listed in the toolbar on the left side of the diagram. Table 74 lists the special elements for enterprise application diagrams:

Table 74 Enterprise application diagram design elements

Icon	Description
	Module: A shortcut to an existing EJB assembler, Web application, resource adapter, application client, or class diagram.
	Archived Module: A shortcut to an existing archive file.
	Security Role: The <i>security role</i> that stands for one of the recommended security roles for the application clients. The <i>Security Role</i> element presents a simplified view of enterprise applications security to the application deployer (such as the J2EE Deployment Expert).

Properties of an Enterprise Application diagram

You can set or change properties of an enterprise application diagram using the diagram's Inspector.

To invoke the Inspector and to set the diagram properties:

1. Right-click on the diagram background and choose **Properties**.
2. Click the **Properties** tab to set the diagram name.
3. Click the **Enterprise Properties** tab to set specific enterprise properties used during the deployment process:
 - **Module name:** Name of an archive *.ear file that is created during the deployment process. By default, this name is the same as the *Name* on the Properties page.
 - **Display name, Small icon, Large icon:** Optional parameters for the deployment descriptor.

Creating diagram shortcuts

You can assemble one or more of your project's existing Web application, EJB assembler, and resource adapter diagrams into an enterprise application diagram. These other diagrams are added to the enterprise application diagram as shortcuts.

To create a shortcut to a diagram:

1. Create a new enterprise application diagram or open an existing one.

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. Click **Module**  on the diagram toolbar and place the element on the designer pane. (Alternatively, right click the diagram background and choose **New | Module**.)

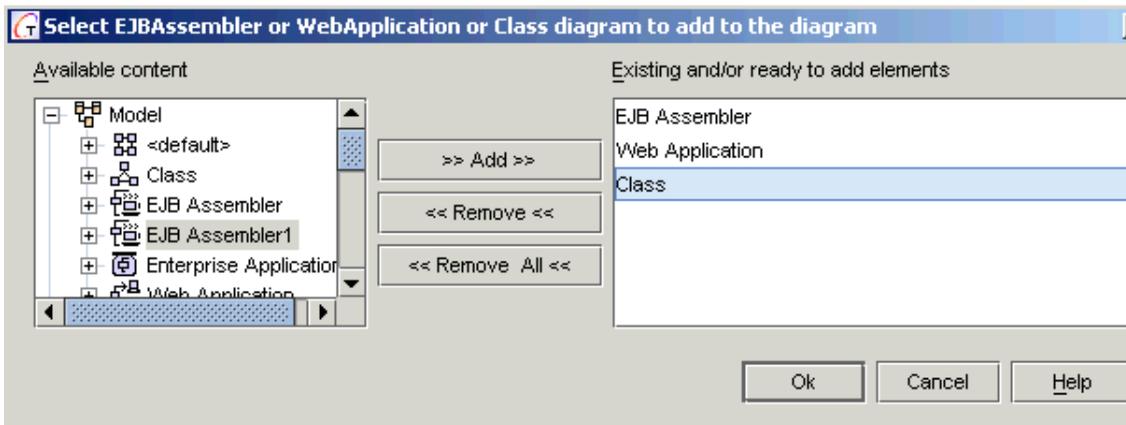
This opens a section manager dialog where you can choose available EJB assembler, Web application, resource adapter, or class diagrams to be added to the enterprise application diagram.

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. Also 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 180](#). Select them in the tree view and click the **Add** button.

Note Even though any shortcut may be physically present on an enterprise application diagram, you should place only “allowed” shortcuts on an enterprise application diagram. Otherwise, the deployment will fail.

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**).

Figure 180 Adding elements to an enterprise application diagram



Importing existing archive files

You can import existing JAR, WAR, and RAR archive files to an enterprise application diagram.

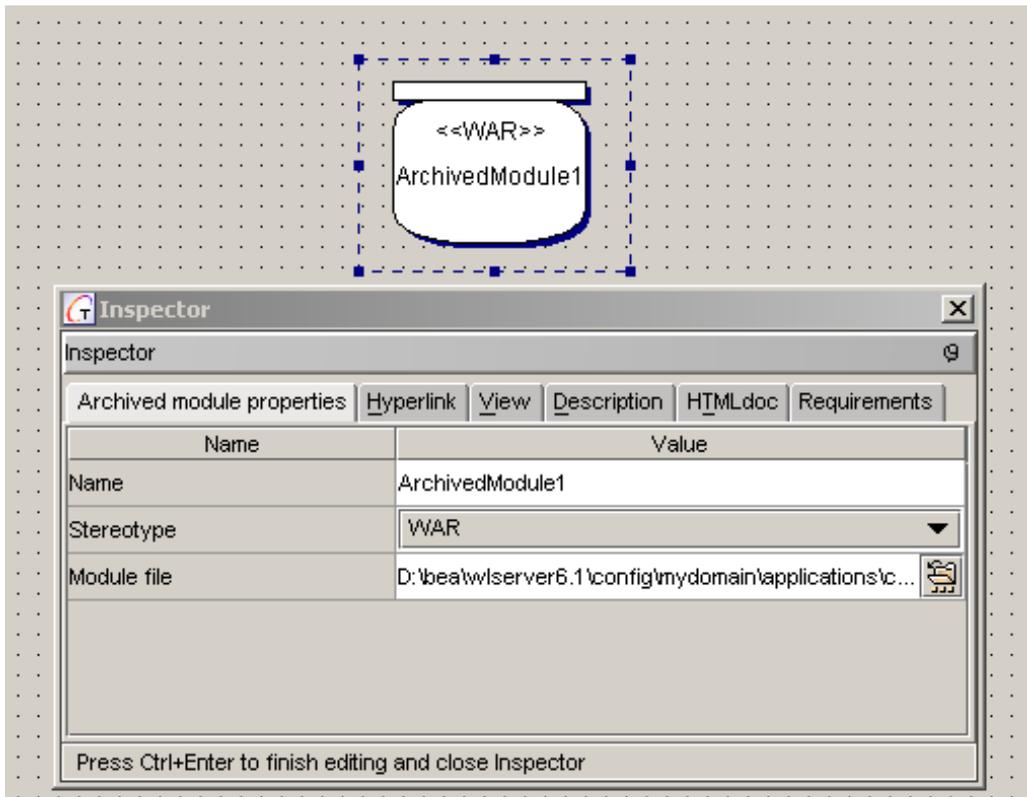
To add existing JAR, WAR or RAR archive modules to an enterprise application diagram:

1. Click **Archived Module**  and place this element on the diagram.
2. Open the Inspector to define the name of this element and its other properties. Choose *Stereotype* (JAR,WAR or RAR), and use the file chooser to find the full path to the corresponding WAR, JAR, or RAR file.

Figure 181 shows a WAR file Inspector along with the WAR element on the diagram.

Note Import of the archive files is possible only for the standard deployment descriptors. Vendor-specific descriptors (such as `weblogic-ear-jar.xml`) are ignored; the information contained within them is not used. You can import only Generic 1.1 and Generic 2.0 archive files without restrictions.

Figure 181 Inspector of a WAR file.



Deploying an enterprise application

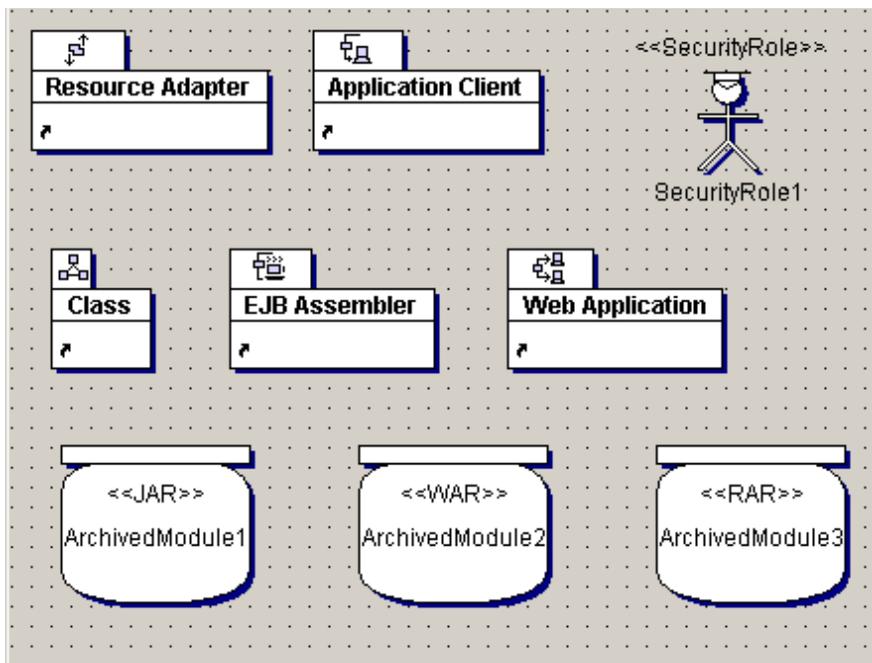
Together's deployment expert simplifies the process of deploying EJBs to various application servers.

To deploy an enterprise application:

1. Place all necessary shortcuts to EJB assembler, Web application, resource adapter, application client, and class diagrams and other design elements on the enterprise application diagram.
2. Define a security role for this diagram if desired.
3. From the main menu, choose **Deploy | J2EE Deployment Expert**. Then follow the steps outlined by the J2EE Deployment Expert.

Figure 182 shows an enterprise application diagram that has been populated with all possible elements.

Figure 182 Enterprise application diagram



4. After you populate an Enterprise application diagram with the desired shortcuts, you can set its deployment properties in the *Enterprise Properties* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for an Enterprise application displays the Borland Enterprise Server 5.2 tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see <http://info.borland.com/techpubs/books/bes/pdfs52/>.

5. From the main menu, choose **Deploy | J2EE Deployment Expert**. Then follow the steps outlined by the J2EE Deployment Expert.

Designing and Developing Application Clients

Together provides special application client diagrams for modeling clients of enterprise applications. This chapter discusses how to use application client diagrams in preparation for deploying enterprise applications.

This chapter includes the following topics:

- “Modeling enterprise application clients” on page 645
- “Using an application client diagram” on page 649

Modeling enterprise application clients

Together provides a special *application client* diagram for modeling elements of an enterprise application client. Elements on an application client diagram include shortcuts to the classes and the references to EJBs and external resources required to deploy the client.

Application clients are packaged in JAR format files with the `*.jar` extension. They include a deployment descriptor describing the enterprise beans and external resources that the application references. Access to resources is configured at deployment time.

An application client diagram contains all the necessary elements needed for deployment. The Together J2EE Deployment Expert can generate the deployment descriptor, adding it and the client to the resulting JAR archive.

Creating application client diagrams

If your project has application client classes, you can create an application client diagram and place shortcuts to classes on this diagram. You can also create references to additional files, references to EJBs, and so on.

There are two ways to create an application client diagram:

- Create a new application client diagram using the Object Gallery or the diagram toolbar button and then add components to it.
- Create an application client 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.

To create a new application client 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. Click the Together tab, click the **Application Client** icon, and click **Finish**.

Alternatively, you can click the **New Diagram** icon on the diagram toolbar, go to the Together tab, and select **Application Client**.

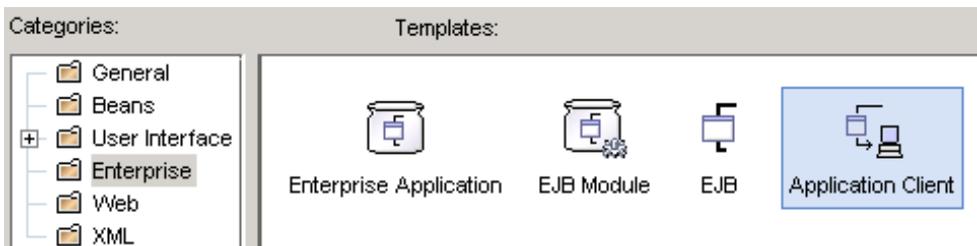
The new application client diagram with the default name *Application Client* opens in the Designer pane. To change the default name, use the inspector (right-click menu | **Properties**). At this point, you can develop the content of the diagram using elements from the diagram toolbar.

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 application client with the Object Gallery:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *Enterprise* in the categories on the left. Then choose *Application Client* among the icons on the right, as shown in [Figure 183](#).

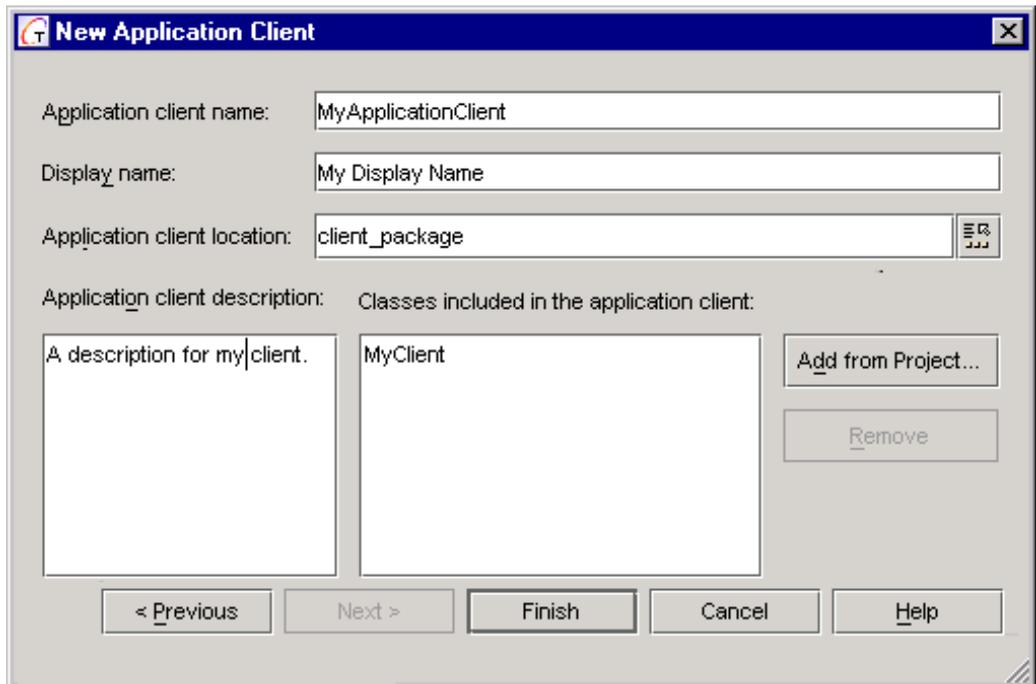
Figure 183 Choosing Application Client in the Object Gallery



3. Click Next.

The resulting dialog box is an expert that guides you through the process of creating the application client. [Figure 184](#) shows the application client expert.

Figure 184 Application client expert from the Object Gallery



Follow the recommendations in the Object Gallery, filling in the parameters as required:

- **Application client name:** Name of the application and the new application diagram
- **Display name:** Display name for the deployment descriptor
- **Application client location:** Location of the diagram. Enter the full path name or click the file chooser button on the right to select the folder.
- **Application client description:** Diagram description (on the Description tab of the diagram inspector).
- **Classes included in the application client:** Click **Add from Project** to open a selection manager dialog for choosing the client classes. To remove class from the list, click the class to select it then click **Remove**.

4. Click Finish. This creates the application client diagram that represents your application client.

Visual design elements

Buttons for all application client diagram elements are in the diagram toolbar. The design elements are listed in [Table 75](#). Notice that the elements are either shortcuts or references. An application client diagram cannot contain classes.

Table 75 Application client diagram design elements

Icon	Description
	Client Classes shortcuts: Opens a selection manager dialog box, where you add or remove shortcuts of client classes.
	EJB Reference: Creates a reference to an EJB. Define a remote type for EJB reference: EJB, or EJB local (for the EJB 2.0 specification).
	Environment Reference: Creates an element with the properties of some static constant, which cannot be changed after EJB deployment.
	Resource Reference: Creates an element representing the properties of the referenced resource. Define the type of reference: resource or resource-environment (for the EJB 2.0 specification).

Properties of an application client diagram

You can specify the properties of an application client diagram using the diagram inspector.

To open the inspector and set the diagram properties:

1. Right-click on the diagram background and choose **Properties** on the right-click menu.
2. In the inspector, open the **Properties** tab and set the diagram name
3. Open the **Application Client Properties** tab to set specific EJB properties used during the deployment process.
 - **Module name:** Name of an archive *.jar file that is created during the deployment process and then included in an archive *.ear file. By default, this name is the same as the *Name* on the Properties page.
 - **Application client main class:** Enter the main class, or browse for it using the file chooser button for selecting the full path to the existing main class.
 - **Display name, Small icon, Large icon:** Optional parameters for the deployment descriptor.

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 as if they constitute a separate module with the same name as the diagram. You can:

- 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 an enterprise application diagram or an EJB assembler diagram.

To start the deployment process:

1. Create or open an enterprise application or EJB assembler diagram.
2. Click **Shortcuts** then the diagram. In the resulting dialog, choose the application client diagram.
3. Place other necessary design elements on the enterprise application or EJB assembler diagram.
4. Choose **Deploy | J2EE Deployment Expert** on the main menu to invoke the J2EE Deployment Expert and start the deployment process (see [“J2EE Deployment” on page 669](#)).

Chapter 38

Designing and Developing Resource Adapters

Resource adapters enable Java applications to connect to enterprise information systems. This chapter discusses the Together resource adapter diagrams that support resource adapters and the J2EE connector architecture.

This chapter includes the following topics:

- “J2EE Connector Architecture” on page 651
- “Creating a resource adapter diagram” on page 652
- “Using a resource adapter diagram” on page 655

J2EE Connector Architecture

J2EE 1.3 includes a connector architecture (J2EE CA) for integration of enterprise information systems (EIS) with application servers and enterprise applications. The focus of J2EE CA is integration of J2EE applications with EISs that are not relational databases. JDBC provides the database connectivity.

A *resource adapter* is software that enables a Java application to connect to an EIS. Both JDBC and J2EE CA are examples of resource adapters. Resource adapters collaborate with application servers to provide the connection pooling, transactions, and security between EIS and enterprise applications.

EJBs get resource references of a specific provider through JNDI. They find the necessary resources using `InitialContext.lookup()` and `<res-ref-name>`. This enables the container to control the resources of EJBs.

Together supports resource adapters and the J2EE container architecture through special *resource adapter diagrams*. During deployment Together creates RAR files from resource adapter diagrams.

Important You cannot include a RAR file into an enterprise application that is targeted to the Sun EE 1.3.1 reference implementation.

Creating a resource adapter diagram

If your project already has appropriate `ConnectionFactory` and `ManagedConnectionFactory` classes, you can create a resource adapter diagram.

To create a new Resource Adapter diagram:

1. Choose **File** | **New** on the main menu.
2. In the Object Gallery, choose *General* in the categories on the left, and click the **Diagram** icon in the templates on the right. Click **Next** to select the type of diagram.
3. Open the Together tab, then click the **Resource Adapter** icon.
4. Click **Finish** to complete the process and create the diagram.

The default name of the new diagram is Resource Adapter. To change the default name use the diagram inspector (right-click menu | **Properties**).

Resource adapter diagram elements

The special resource adapter diagram elements are on the toolbar on the left side of the diagram. [Table 76](#) lists these elements.

Table 76 Resource adapter diagram elements

Icon	Description
	Config Property: Declaration of a single configuration property for a <code>ManagedConnectionFactory</code> instance (an option description, a name, a type and an optional value for a configuration property). This element corresponds to the <code><config-property></code> attribute in the deployment descriptor.
	Authentication Mechanism: A type of an authentication mechanism. Any additional security mechanisms are outside of the scope of the connector architecture. This element corresponds to the <code><authentication-mechanism></code> attribute in the deployment descriptor.
	Security Permission: Security permission based on the Security policy file syntax. This 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 element corresponds to the <code><security-permission></code> attribute in the deployment descriptor.

Table 76 Resource adapter diagram elements

Icon	Description
	Connection: Links to a connection interface and to a connection implementation class.
	Connection Factory: Links to a connection factory interface and to a connection factory implementation class.
	Resource Adapter Link: Links Connection/Connection Factory elements with the corresponding interfaces and implementation classes.

A resource adapter diagram contains only shortcuts to connection implementation classes and their interfaces. It does not contain the actual connection classes.

To add a shortcut of an existing connection class:

1. Right click the diagram background and choose **New | Shortcuts**.
2. In the resulting selection manager dialog, choose the connection class.

Creating a resource adapter diagram using a pattern

Together provides a special pattern for resource adapters and their associated connection classes.

To create a resource adapter diagram using a pattern:

1. Open the class (or package) diagram where you want to place the connection classes corresponding to the new resource adapter diagram.
2. Click **Class by Pattern**  and click the diagram.
3. In the resulting dialog box, choose *J2EE - Resource Adapter* from the Patterns listed on the left.
4. In the Parameters on the right, enter the name for the new resource adapter diagram. For the remaining values, enter the names of new connection classes. If you want to use an existing connection class instead of creating a new one, click the file browser button at the right of the name field.

[Figure 185](#) shows the parameters for the resource adapter pattern. The new resource adapter diagram will have the default name, Resource Adapter.

Figure 185 Resource adapter pattern parameters

Parameters	
Name	Value
Resource Adapter Diagram name	Resource Adapter
Managed Connection Factory class name	myManagedConnectionFactory
Connection Factory interface name	myConectionFactoryInterface
Connection Factory class name	myConnectionFactory
Connection interface name	myConnectionInterface
Connection class name	myConnectionClass

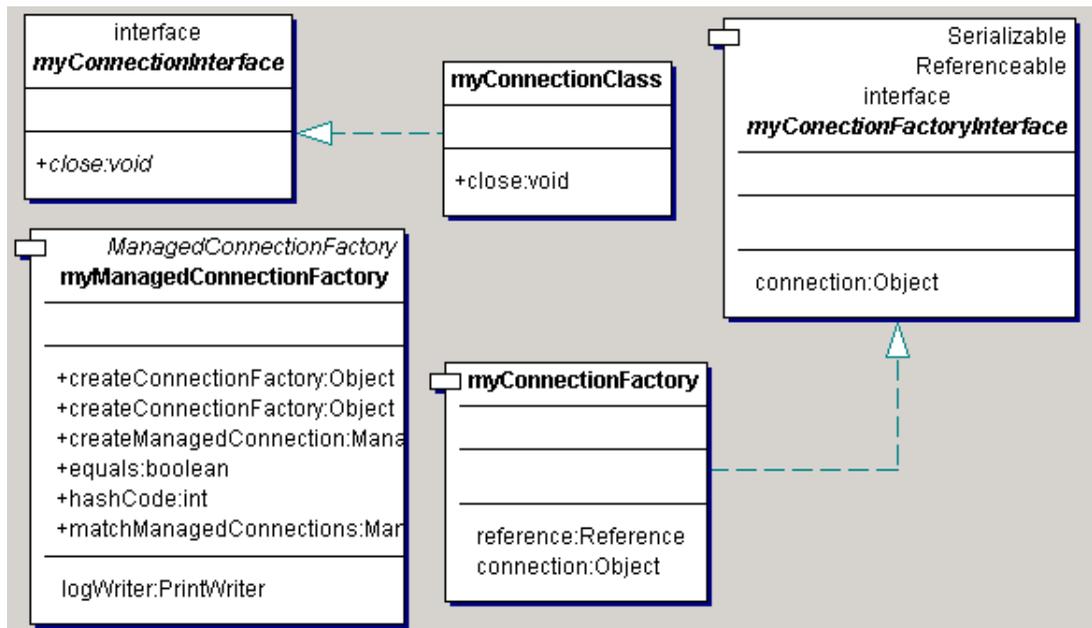
5. Click **Finish**.

The class diagram displays the new connection classes and interfaces. [Figure 186](#) shows the classes created as a result of filling in the dialog box of [Figure 185](#).

The pattern also creates the new resource adapter diagram, as shown in [Figure 187](#). The diagram is part of the class diagram package. Notice that the classes on the resource adapter diagram display with the shortcut symbol in the lower left corner.

You can open the resource adapter diagram from the Explorer. Look for the diagram name and the Resource Adapter icon, .

Figure 186 Class diagram with connection classes



Using a resource adapter diagram

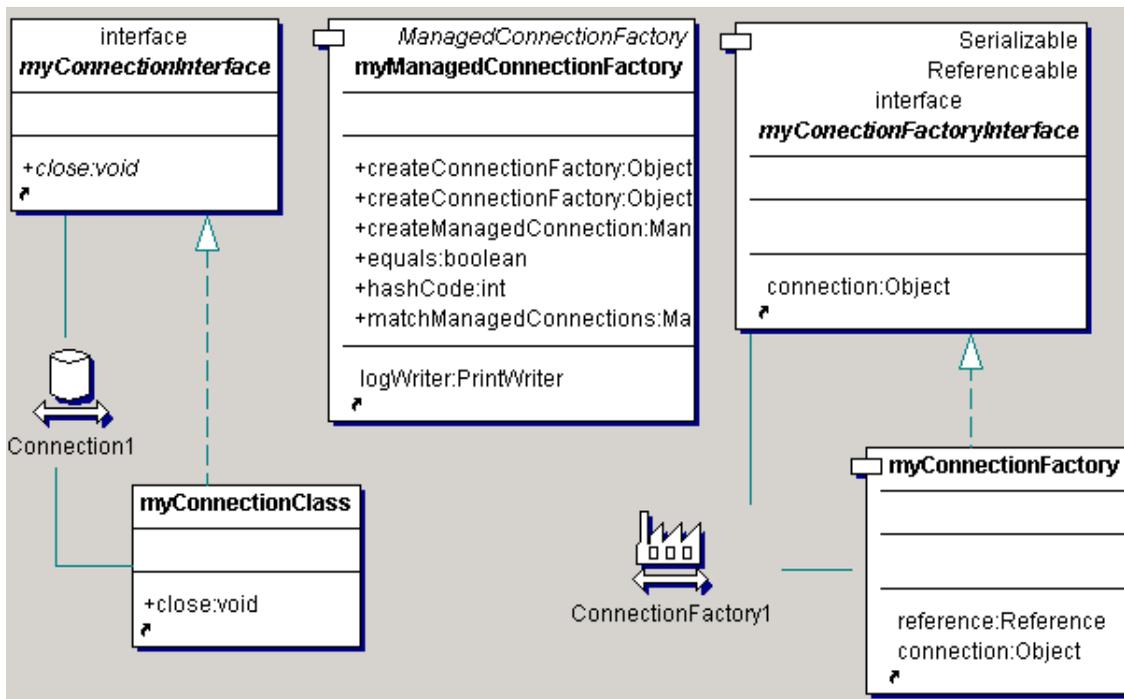
You can consider the elements that are on a resource adapter diagram as a separate module with the diagram's name.

Together provides:

- Generating an XML deployment descriptor for each resource adapter diagram.
- Packaging the compiled classes and a deployment descriptor into an RAR for each resource adapter diagram.
- Deploying to the current application servers, which support J2EE specification, version 1.3.

The deployment process in Together completes all of these tasks.

Figure 187 Resource adapter diagram.



After you populate a Resource adapter diagram with the desired components, you can set its deployment properties in the *Resource Adapter* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the Inspector for a Resource adapter displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see <http://info.borland.com/techpubs/books/bes/pdfs52/>.

To deploy from a resource adapter diagram:

1. Open the resource adapter diagram.
2. On the main menu, choose **Deploy | J2EE Deployment Expert**.
3. Continue the deployment process, filling in the process options and fields as required by the J2EE Deployment Expert.

For detailed instructions on deployment, see [Chapter 40, "J2EE Deployment"](#).

Chapter 39

J2EE Platform Security Support

The security features of an enterprise or Web application identify which users or other entities can access the application or components of the application. Security features are implemented in deployment descriptors and in code.

This chapter discusses the support in Together for J2EE security features. It includes the following topics:

- [“Security support in J2EE diagrams” on page 657](#)
- [“Security in EJB modules” on page 659](#)
- [“Security in Web applications” on page 663](#)
- [“Security in enterprise applications” on page 666](#)
- [“Security in resource adapters” on page 666](#)

Security support in J2EE diagrams

J2EE security features are intended to provide security for each tier of components of enterprise and Web applications. Primary J2EE security features include security roles, security role references, principals, security constraints, security and method permissions, Web resource collections, and authentication mechanisms.

Together enables you to visually model security features for your application through special security elements on J2EE diagrams, EJB security role references, and links defining the relationships among security elements and application components.

Together translates some of the security elements on diagrams directly into code. The J2EE Deployment Expert captures information for security design elements in deployment descriptors when you deploy from Together.

Security elements in J2EE diagrams

J2EE security elements on J2EE diagram vary with the type of diagram. [Table 77](#) lists all of the security elements along with their corresponding diagram types.

Table 77 J2EE security elements on diagrams

Button	Meaning	Diagram Types
	Security Role: A category of application users. Corresponds to <security-role> in a deployment descriptor.	EJB assembler, Web application, enterprise application
	Principal: A user or a group of users who can be authenticated.	EJB assembler, Web application
	Method Permission: Indicates which roles are allowed to access which methods. Corresponds to <method-permission> in a deployment descriptor.	EJB assembler
	Web resource collection: A set of URL patterns or HTTP methods that describe resources to be protected. Corresponds to <web-resource-collection> in a deployment descriptor.	Web application
	Security constraint: Determines who can access a Web resource collection. This includes authorization constraints and user data constraints. Corresponds to <security-constraint> in a deployment descriptor.	Web application
	Link: Provides mapping between security elements.	EJB assembler, Web application
	Authentication Mechanism: Describes how users of resources are to be authenticated. Corresponds to <authentication-mechanism> in a deployment descriptor.	resource adapter
	Security Permission: Indicates which permissions are allowed from specified code sources. Corresponds to <security-permission> in a deployment descriptor.	resource adapter

Security roles

A *security role* is a category of users which is defined by the role they play with respect to an enterprise or Web application. Security roles are the central J2EE feature that supports authorization. Together provides a security role elements on EJB assembler, enterprise application, and Web application diagrams.

Mappings between security roles and other J2EE security elements indicate who or what entity can access all or specific parts of an application. These elements include:

- **Principals:** users (in a security policy domain) or groups (in an operational environment) that can be authenticated. There are special diagram elements for principals.
- **Method permissions:** indicate which security roles can access a collection of methods. There are special diagram elements for method permissions.
- **Security role references:** EJB attributes that define security roles.

Entity and session EJBs on Together diagrams have right-click menu commands for inserting security role references into the bean code. Security role references display in a bottom compartment of an EJB in a light blue color.

Linking security elements

Together provides a link diagram element on EJB assembler and Web application diagrams to establish mappings between different security elements. You can link the following types of elements and features:

- Security roles to principals
- Method permissions to security roles
- Methods to method permissions
- Security role resources to security roles
- Security constraints to Web resource collections
- Security constraints to security roles

When you link EJB methods to a method permission, you establish the access permission for those methods. When you link that method permission to a security role, you establish the roles that users must be playing to access the methods. When you link that security role to principals, you establish who can access the methods.

Some of the linking is optional. However, there are two restrictions on linking security roles and security role elements:

- If an EJB defines a security role reference (shown in the last compartment of the EJB node in the diagram), the reference must be linked to exactly one security role element.
- One security role element can be linked to any number of EJB security role references or method permissions, including none at all.

Security in EJB modules

There are two variants of security methodologies in EJB modules:

- **Declarative:** The security features of the application are defined external to the Java code, usually in a deployment descriptor.
- **Programmatic:** The security features of the application are defined within the application code.

Together supports both variants in EJB assembler diagrams. Recall that an EJB assembler diagram represents a single EJB module.

Declarative security and method permissions

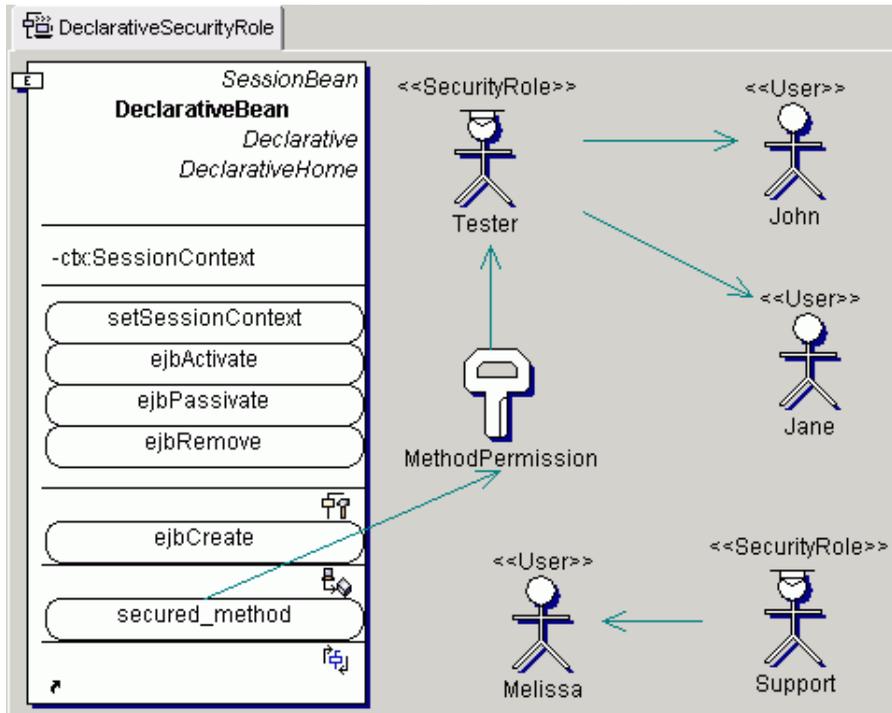
In declarative security, all of the security is defined external to the application. No special security code is required in the EJB. The `<method-permission>` element of the deployment descriptor determines the list of methods that can be accessed by users with a certain security role. On an EJB assembler diagram, the method permission diagram element is critical for declarative security.

To implement declarative security in an EJB module:

1. Open the EJB assembler diagram representing your EJB module.
2. Determine which methods you want to be secured in the EJBs on the diagram. In [Figure 188](#), `secured_method` is such a method.
3. Place a method permission element on the diagram () and link the secured method to the method permission.
4. Place security role elements on the diagram (), editing their names as desired. These names are used in the deployment descriptor. [Figure 188](#) shows an EJB assembler diagram with two security roles, `Tester` and `Support`.
5. Place principal elements on the diagram () and set their properties.
 1. To set the properties, right click the principal and choose **Properties**. This opens the inspector.
 2. Open the **Properties** tab. of the inspector. You can change the name; you can set the element stereotype to `User` or `Group`.

[Figure 188](#) has three principals, `John`, `Jane`, and `Melissa`. All of them are users.

Figure 188 EJB assembler diagram showing a declarative security role



6. Link each security role to the principals making up that role.
7. Link the method permission to a security role.

Figure 188, shows that the only users who can access `secured_method` are John and Jane.

It is not necessary to link each security role to a method permission. For example, Figure 188 shows `Support`, which is not linked to any method permission. In that case, the J2EE Deployment Expert adds the `Support` security role and its user `Melissa` to the deployment descriptor.

Programmatic security and security role references

In programmatic security, the security features of an application are built into the Java code. Two methods implement programmatic security for EJB modules in conjunction with security roles:

- `isCallerInRole`: tests if the caller has a given security role with regard to the security context of the client
- `getCallerPrincipal`: obtains a `java.security.Principal` interface for the current caller.

`isCallerInRole` requires a `<security-role-ref>` element in the deployment descriptor for all the security role names used in the EJB code. The J2EE Deployment Expert creates such deployment descriptor elements by making use of security role references, which are attributes of the EJB code.

To create a security role reference in an EJB module:

1. Open the EJB assembler diagram representing your EJB module.
2. Create security role references in the EJBs as required.
 1. Right click the EJB and choose **New | EJB Security Role Reference**.
 2. Edit the name when you create the reference.

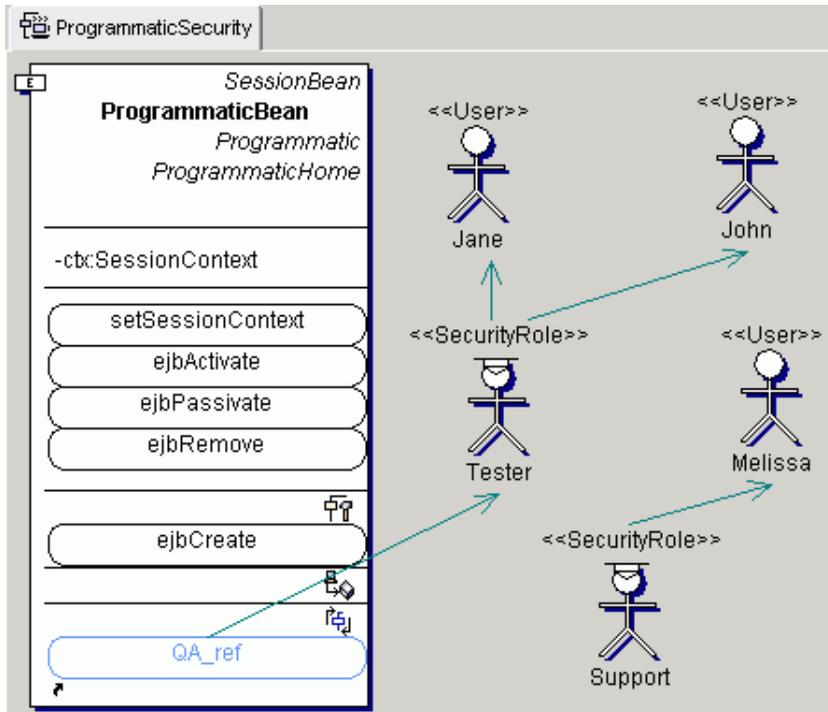
This defines an attribute in the bean representing the security role reference. [Figure 189](#) shows an EJB assembler diagram with an EJB named `ProgrammaticBean` and its security role reference named `QA_ref`.

3. Place security role and principal elements on the diagram as required. See [“Declarative security and method permissions” on page 660](#) for details.
4. Link the EJB security role reference to the corresponding security role elements. [Figure 189](#) shows the link from `QA_ref` with the security role `Tester`. In this example, the access of `Jane` and `John` to the `ProgrammaticBean` is defined by the `Tester` security role.

You can also place a security role element on the diagram and link it to some principal element without any connection to a concrete method. [Figure 189](#) shows this variant where the `Support` security role links to the `Melissa` principal.

If an EJB has a security role reference linked to a security role, then the J2EE Deployment Expert will generate information about properties of the reference and its associated security role in the deployment descriptor.

Figure 189 Programmatic Security Role



Security in Web applications

Three elements on Web application diagrams support the security of Web applications: security roles, security constraints, and web resource collection.

Note You can specify the user authentication options on the Login Config tab of the Web Properties tab in the diagram inspector. For details, see [“Login Config tab” on page 618](#).

Security roles and principals

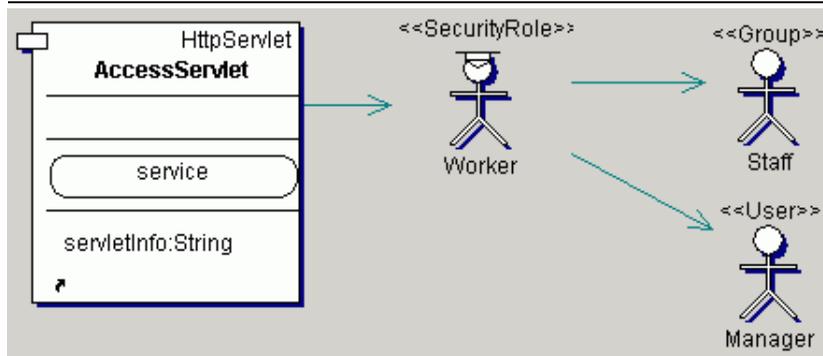
The Web container determines which users have access to which parts of a Web application. The deployment descriptor created by the J2EE Deployment Expert can provide that kind of information from security roles on Web application diagrams.

To use a security role on a Web application diagram:

1. Open the Web application diagram corresponding to your application. The diagram should contain the elements necessary for your project (shortcuts of servlets, JSPs, Web files, and so on). [Figure 190](#) shows a Web application diagram with a single code element, which is a shortcut to a servlet named `AccessServlet`.
2. Place a security role element on the diagram (). Set the name for this security role. This name will be used in the deployment descriptor. [Figure 190](#) shows a single security role named `Worker`.
3. Place principal elements on the diagram () and set their properties. [Figure 190](#) shows two principals, `Staff` and `Manager`. The stereotype for `Staff` is `Group`, while the stereotype for `Manager` is `User`.
4. Link the shortcut to the security role.
5. Link the security role to its principal users.

In [Figure 190](#), the access of `Staff` and `Manager` to `AccessServlet` is defined by `Worker`.

Figure 190 Using a security role design element for supporting security



Constraints and Web resource collections

A Web resource collection is a set of URL patterns and HTTP methods that describe Web resources to be protected, such as images and JSPs. You can specify the protection in Web application diagrams by using a security constraint and security roles. The security constraint determines who is authorized to access the Web resource collection.

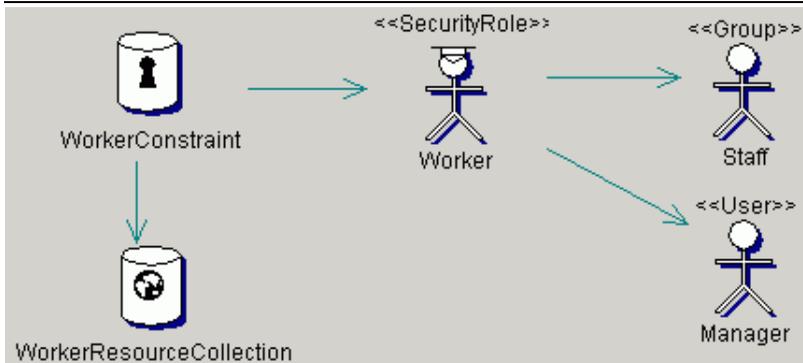
To specify security constraints for a Web resource collection:

1. Open the Web application diagram corresponding to your Web application.
2. Place a Web resources collection element on the diagram () and set its properties.
 1. Right click the Web resources collection element and choose **Properties**. This opens the inspector.

2. Open the **Web resource collection properties** tab.
 3. Fill the **URL pattern** field with the path to the protected resource files. For multiple patterns, click the button on the right of the field.
 4. Fill the **HTTP method** field with the appropriate HTTP methods for the resources (such as `GET` and `POST`). For multiple methods, click the button on the right of the field.
3. Place a security constraint element on the diagram (🔒) and set its properties.
 1. Right click the security constraint element and choose **Properties**. This opens the inspector.
 2. Open the **Security constraint properties** tab.
 3. In the **User data constraints** field, choose one of the following transport guarantees:
 - **NONE**
 - **INTEGRAL**: data cannot change while transmitting between client and server
 - **CONFIDENTIAL**: no other entities can observe the contents of the transmission
 4. Draw a link from the security constraint to the Web resources collection.
 5. Create security roles and principals as described in [“Declarative security and method permissions”](#) on page 660.
 6. Draw a link from the security constraint to the appropriate security role.

In [Figure 191](#), the Web container grants users access to the `WorkerResourceCollection` of Web resources only after it determines that they are in a `Worker` role, which means that they are `Manager` or members of the `Staff` group.

Figure 191 Links between security supported elements



Security in enterprise applications

A security role element on an enterprise application diagram defines the security properties for all elements on the enterprise application diagram. When you deploy from the enterprise application diagram, the J2EE Deployment Expert generates the lines corresponding to a security role in the deployment descriptor

Security in resource adapters

Two diagram elements that support security in resource adapter diagrams: authentication mechanisms and security permissions.

The security permission element corresponds to the deployment descriptor tag `<security-permission>`, which in turn contains a `<security-permission-spec>` tag. These tags describe the permission to be granted to each resource adapter.

To describe a permission for a resource adapter:

1. Open the resource adapter diagram corresponding to the RAR file.
2. Put a security permission element on the diagram ().
3. To set the properties of the security permission element, right click the element and choose **Properties**. This opens the inspector.
4. Open the **Properties** tab of the inspector. Fill in the **Specification** field with the appropriate grant statement, such as the following one shown in [Figure 192](#):

```
grant {permission java.security.SecurityPermission "Security.insertProvider.*";};
```

Figure 192 Setting security permissions



The authentication mechanism element corresponds to the deployment descriptor tag `<authentication-mechanism>`. It describes how the resource adapter user is to be authenticated.

To specify authentication mechanisms for RAR files:

1. Open the resource adapter diagram corresponding to the RAR file.
2. Put an authentication mechanism element on the diagram ().
3. To set the properties of the authentication mechanism element, right click the element and choose **Properties**. This opens the inspector.
4. Open the **Properties** tab of the inspector. Choose settings for the following:
 - **Authentication mechanism:** Choose **BasicPassword** (Basic user-password- based authentication specific to an EIS) or **Kerbv5** (Kerberos version 5-based). This field corresponds to the deployment descriptor tag `<authentication-mechanism-type>`.
 - **Credential interface:** Choose **javax.resource.spi.security.PasswordCredential** or **javax.resource.spi.security.GenericCredential**. This field corresponds to the deployment descriptor tag `<credential-interface>`.

J2EE Deployment

J2EE deployment is a process of generating deployment descriptors and packing archive files (*.jar, *.war, or *.ear) on an application server. This chapter discusses using the J2EE Deployment Expert to assist in the deployment process.

This chapter discusses the following topics:

- [“Deployment process” on page 669](#)
- [“Supported application servers” on page 670](#)
- [“Installations and resources required for deployment” on page 671](#)
- [“Together provides the following plug-ins for starting application servers:” on page 672](#)
- [“J2EE Deployment Expert features” on page 674](#)
- [“J2EE Deployment Expert options common to all servers” on page 679](#)
- [“Server-specific J2EE Deployment Expert pages” on page 688](#)
- [“Making transitions among application servers” on page 696](#)
- [“Encoding in deployment descriptors” on page 700](#)

Deployment process

Without the aid of an “expert,” the process of deploying to application servers can be tedious and difficult. For example, an e-commerce or other enterprise application based on J2EE technology may contain multiple EJBs, JSP files, servlets, and other components. Before the application is ready to use, you must select an application server, write the deployment descriptors (and possibly other necessary files depending on the server), pack all files in the archive, and deploy the result to a specific location.

Together automates this process. By using the J2EE Deployment Expert, you can verify and correct your code according to the EJB specification of the application server, generate deployment descriptors, compile all files, pack them in *.jar, *.war, *.rar, or *.ear files, and deploy to the application server. You can optionally generate a simple JSP client for testing the deployed EJB running on the application server.

Together can deploy to most of the popular application servers. On server platforms that support it, Together can hot deploy directly to the application server. You can also start some application servers from within Together.

Supported application servers

When you open a new Java project and then invoke the J2EE Deployment Expert, the dialog opens with default Generic 2.0 settings.

To see a list of the currently supported servers and to select a supported server:

1. Open a project.
2. From the main menu, choose **Project | Project Properties**.
3. In the resulting project properties dialog box, click the **EJB** tab.
4. The drop-down list of application servers shows the server platforms that are currently integrated into Together. Choose your target application server from the list.

Integrating other application servers

It is possible to 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 site **http://www.togethersoft.com/developers/integrations/** to see the current list of J2EE Application servers supported in Together.

This list provides the following information:

- The level of integration (*High, Medium, Low*).

If the level of integration for your application server is high, you can deploy directly from within Together using the J2EE Deployment Expert. If the level is not high, download this application server and read the help on the TogetherSoft Web site about the possibilities of integration.

- Where you can find this application server (*In Build, Download*).

If the location of this application server is *In Build*, you can find it in `TGH/bundled` (for example, SunEE Reference Implementation J2EE or Tomcat).

If the location is *Download*, you can download this application server using the corresponding hyperlink.

Generic server options

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.

When the target application server is generic, Together creates the deployment descriptor for `ejb-jar.xml` according to the respective EJB specification. Together adds information about references or relationships satisfying the corresponding EJB specification to the deployment descriptor.

If you choose Generic 1.0, the J2EE Deployment Expert can generate:

- `ejb-jar.xml` for EJB modules
- `web.xml` for Web applications

If you choose Generic 1.1, 2.0, the J2EE Deployment Expert can also generate:

- `ejb-jar.xml` for EJB modules
- `web.xml` for Web applications
- `application.xml` for enterprise applications

For Generic 2.0 you can import existing JAR files prepared according to EJB 2.0 specification.

For Generic *x.x*, there are plug-ins for the following operating systems: Windows 98, Windows 2000, Windows Me (Windows Millennium Edition), Sun OS (Solaris), HP Unix, OSF/1 (Open Software Foundation), Linux, Macintosh OS (Mac OS X).

Installations and resources required for deployment

The following installations and permissions are prerequisites to deploying:

- 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).

Important 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. You can avoid this situation by using an external compiler (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.

Together provides the following plug-ins for starting application servers:

- Start SAP J2EE AS 6.20
- Start Oracle 9i Application Server 1.0.2.2.1
- Start Sun ONE Application Server 6.5
- Sun EE Starter (built-in SunEE Reference Implementation J2EE 1.3.0 Application Server)
- WebLogic Server 5.1.0 Starter
- WebLogic Server 6.0 Starter
- WebLogic Server 6.1 Starter

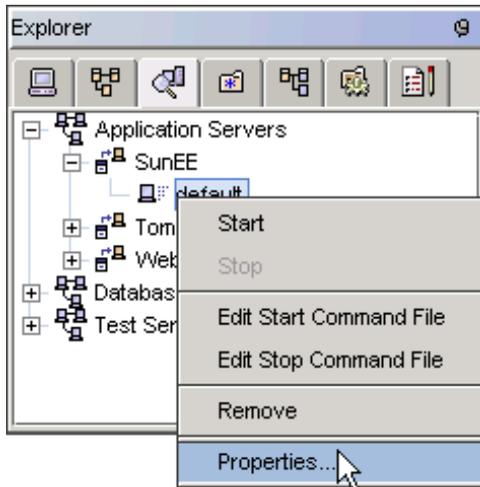
You can start these application server from the Explorer pane or from the J2EE Deployment Expert.

Starting application servers from Together

To start an application server from the Explorer pane:

1. In the Explorer pane, open the Server Explorer tab . Make sure that all properties of your server are defined correctly. For example, to view or edit the SunEE reference implementation server properties, navigate to *Application Servers - Sun EE - default*. Right click and choose **Properties**, as shown in [Figure 193](#).

Figure 193 Accessing the SunEE reference implementation properties.



2. If necessary, edit the *Start command file* for your server. Right-click the server in the Explorer pane and choose **Edit Start Command File**.

Edit example: For BEA WebLogic 6.1, you should add two lines to the command file `startWebLogic.cmd`:

1. In the beginning of this file after the `SETLOCAL` program line, and before the line `cd ..\..` program, add this:

```
cd D:\bea\wlserver6.1\config\mydomain\
```

where `D:\bea\wlserver6.1` is the full path to the directory where BEA WebLogic 6.1 is installed. The result should look like this:

```
SETLOCAL
D:

cd D:\bea\wlserver6.1\config\mydomain\
cd ..\..
```

2. Set your BEA WebLogic password, so that you will not be prompted for the password during server startup. Add a new line before `set STARTMODE=true`. The result should look like this:

```
set WLS_PW=together
set STARTMODE=true
```

Note On UNIX systems, `JAVA_HOME` might not be set in `StartWeblogic*.sh`. In this case, set `JAVA_HOME` in the script manually.

3. If necessary, edit the *Stop command file* for your server. Right-click the server in the Explorer pane and choose **Edit Stop Command File**.
4. To launch your server from Together, right click the server in the Explorer pane and choose **Start**.

An alternative method to start an application server:

1. Open a project.
2. Open a diagram appropriate for deployment (see [“Diagrams for deployment” on page 674](#)).
3. From the main menu, choose **Deploy | J2EE Deployment Expert**.
4. The first page of the Expert has a drop-down list of application servers with the currently integrated server platforms. Choose an application server to start from the available plug-ins listed at the beginning of this section.

J2EE Deployment Expert features

The J2EE Deployment Expert guides you through the deployment process. This section discusses the general features of the Expert.

Diagrams for deployment

You can invoke the J2EE Deployment Expert from diagrams of the following types:

- Class diagrams (creates a JAR archive file).
- EJB assembler diagrams (creates a JAR archive file). See [Chapter 33, “Assembling EJB Modules” on page 591](#) for more information.
- Web application diagrams (creates a WAR archive file). See [Chapter 35, “Designing and Developing Web Applications” on page 613](#) for more information.
- Resource adapter diagrams (creates an RAR archive file). See [“Designing and Developing Resource Adapters” on page 651](#) “Designing and Developing Resource Adapters” for more information.
- Enterprise application diagrams (creates an EAR archive file). See [“Visual assembly of enterprise applications for deployment” on page 637](#) for more information.

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 enables you to specify the following:

- The target application server platform.
- Which deployment-related actions you want to take place (such as compile).

- 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 EJBs.
- Optional verification and correction.

Running the J2EE Deployment Expert

You can start the J2EE Deployment Expert from a diagram that supports deployment (class, enterprise application, EJB assembler, resource adapter, and Web application).

To run the J2EE Deployment Expert:

1. Open a diagram that supports deployment.
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. Generic 2.0 application server is set by default. Select the application server that you want to target.

[Figure 194](#) shows the first page of the J2EE deployment Expert for the Generic 2.0 specification for an application server.

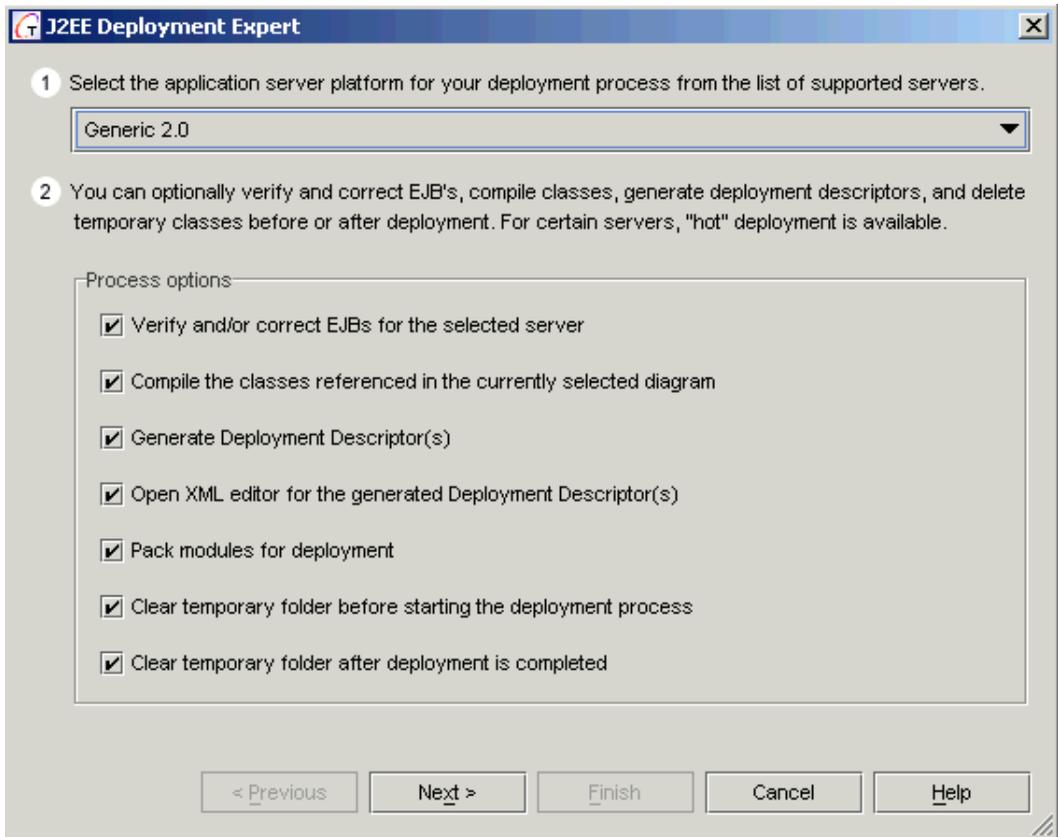
4. Go through the pages of the Expert, filling in the options as desired. Use **Previous** and **Next** buttons to navigate through the Expert's pages.
5. Click **Finish** to start the deployment process.

As its initial step, Together tests the project for EJBs or other elements that should be deployed to an application server.

According to your options, Together interacts with the compiler to compile EJB or other classes, verifies the EJB classes, generates the XML deployment descriptors, updates the EJB specification conformity, generates 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.

Important If the deployment fails unexpectedly, a possible reason is that the target application server does not support the current encoding. See ["Encoding in deployment descriptors" on page 700](#) for more information.

Figure 194 Default *Process options* page



Main scenarios for using the J2EE Deployment Expert

There are two main scenarios for using the J2EE Deployment Expert:

- For “fast track” deployment with default values for security and permissions or for deployment prototyping, start the J2EE Deployment Expert from a class diagram. Choose the target server platform and set the other options as desired.
- For “full featured” assembly information with control over security and permissions, start the J2EE Deployment Expert from an EJB assembler diagram, Web application diagram, resource adapter diagram, or enterprise application diagram.

Web application and enterprise application diagrams can be used with these choices of target application servers: Generic 1.1, Generic 2.0, IBM WebSphere 4.0, BEA WebLogic 6.0, 6.1 and 7.0, Sun ONE 6.5, Borland Enterprise Server 5.2, and SunEE 1.2, 1.3.

Configuration files

Deployment is possible for different operating system platforms using special plug-in config files. If you want to change something in the one of plug-in files, consider these naming rules:

- If no OS-specific config file is defined, the common `<plug-in-name>.config` file is used.
- OS-specific config files have the suffix `java.lang.System.getProperty("os.name")` with all non alpha-numeric symbols replaced by underscores. For example, if you have `MYASPlugin`, then configs would be:

`MYASPlugin.config`: Common config for all “unknown” operating systems

`MYASPlugin.config_Windows_NT`: Main config for Windows NT

`MYASPlugin.config_SunOS`: Main config for Solaris

`MYASPlugin.config_Linux`: Main config for Linux

`MYASPlugin.settings.config_Windows_NT`: User settings specified in last session of the deployment.

Server-specific deployment information

This section provides server-specific information for starting deployment on the Sun EE reference implementation application server, BEA WebLogic application servers, and IBM WebSphere application servers.

Note This User Guide assumes that you are familiar with the requirements of your application server.

You should make sure that all the necessary libraries are added to your project. Together automatically adds the maximum number of available libraries in the plug-ins, but some servers require that you add paths. For example, according to SunEE requirements, you should add the `client.jar` to the project class path before running to avoid an exception. If the EJBs and other elements are deployed in different projects, you should add the paths to these archive files to the project.

Sun EE reference implementation application server

Together provides the Sun EE reference implementation J2EE 1.3.0 application server within the Together build. This application server is a default implementation that is helpful for conceptual testing.

Together supports the Sun EE Reference Implementation for J2EE 1.2.1 application server as well the built-in version. If you want to use Sun EE Reference Implementation J2EE 1.2.1, you should first install it.

Note All projects deployed to Sun EE Reference Implementation J2EE 1.2.1 deploy successfully to Sun EE Reference Implementation J2EE 1.3.0.

Borland Enterprise Server 5.2

Together supports the deployment to Borland Enterprise Server 5.2. To hot deploy from Together, first start this application server.

Note Together currently supports deployment to BES 5.2 for Windows and Solaris only.

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.

Important When you are developing the project for IBM WebSphere application servers, make sure that you create a package. IBM WebSphere does not work without packages.

With IBM WebSphere 3.5 and 4.0 application servers, you can initiate the deployment process from class, EJB assembler, Web application and enterprise application diagrams. You can also place shortcuts to application client diagrams on enterprise application diagrams for subsequent deployment to IBM WebSphere 4.0.

BEA WebLogic Application Server

Together supports BEA WebLogic 4.5.1, 5.1, 6.0, 6.1, and 7.0 Application Servers.

Note BEA WebLogic 6.1 and 7.0 application servers support EJB 2.0 specification. You can use applications previously developed under EJB 1.1 specification without any code modification (this is very important for CMP Entity Beans).

Important Be careful when deploying a project satisfying the EJB 2.0 specification to BEA WebLogic Application Server 6.0, which supports only public draft 2 of EJB specification 2.0. (In that draft, EJB relationships were not completely implemented.) However, the Together plug-in for BEA WebLogic 6.1 completely supports EJB relationships in accordance with the final release of EJB specification 2.0.

You can deploy to BEA WebLogic 6.1 and 7.0 from any of the diagrams listed in [“Diagrams for deployment” on page 674](#). You can deploy to BEA WebLogic 6.0 for any of those diagrams except resource adapter diagrams.

You can deploy to BEA WebLogic 5.1 from class diagrams or EJB assembler diagrams only (JAR archive file is created). You can also generate a deployment descriptor using a Web application diagram.

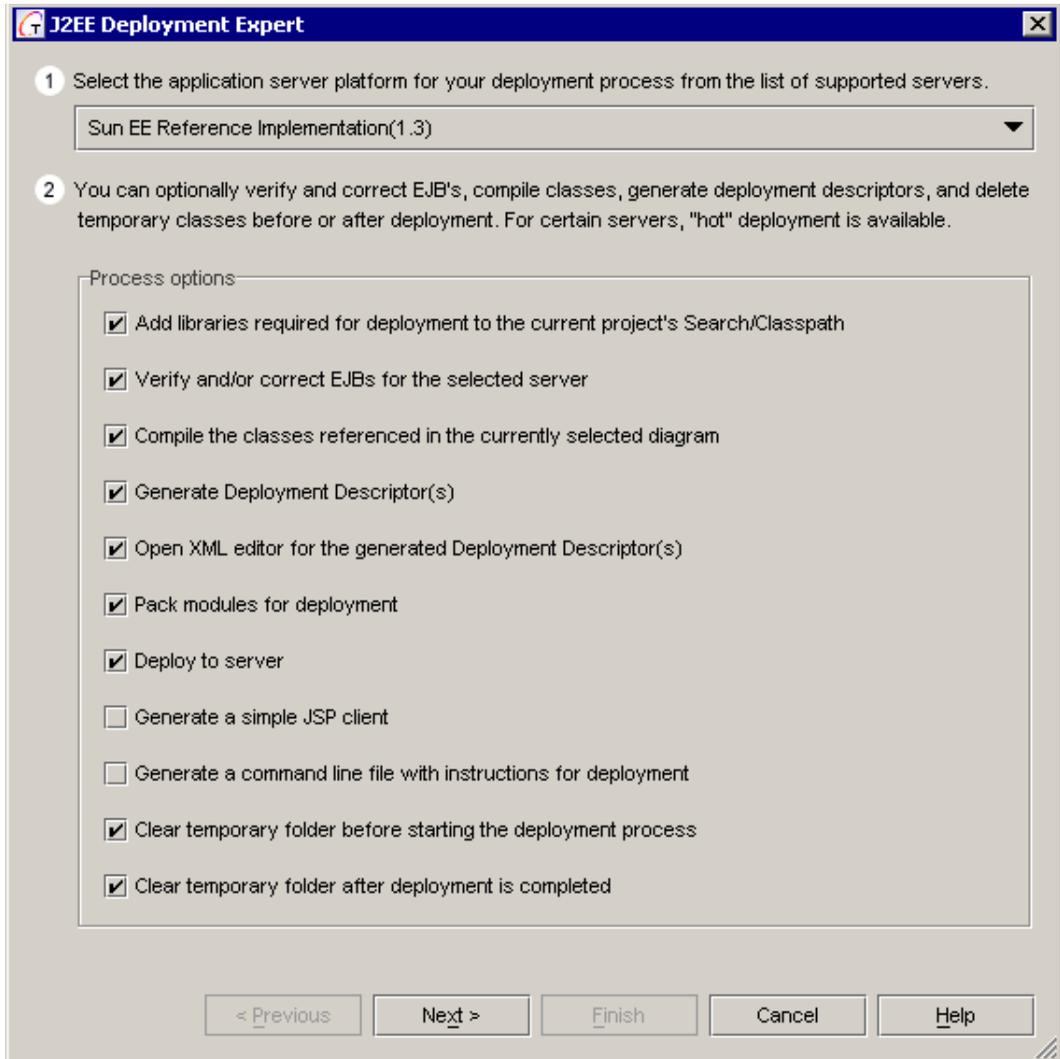
J2EE Deployment Expert options common to all servers

The J2EE Deployment Expert is a guide that takes you through several pages of options that describe deployment tasks and settings. The pages depend on the current application server. This section discusses the options on pages that are common to a few application servers: Sun EE Reference Implementation 1.2.1, 1.3.0, IBM WebSphere 3.5 and 4.0, and BEA WebLogic 5.1.0, 6.0, 6.1, and 7.0.

First page: process options

The first page of the Expert displays the process options. [Figure 195](#) shows the process options for the Sun EE reference implementation application server.

Figure 195 Process options for Sun EE reference implementation



The options in the Expert that common to Sun EE, Borland Enterprise Server 5.2, BEA WebLogic, and IBM WebSphere are as follows:

- **Add libraries required for deployment to the current project's Search/Classpath:** EJB deployment requires several server-specific libraries. Because diverse servers use different versions of the J2EE specification, server vendors generally suggest adding their own libraries that cover the J2EE specification. (For Sun EE deployment, Sun EE is added to the classpath. For Borland Enterprise Server 5.2, this option is absent).

Note If all the necessary libraries have already been added to the classpath, you should uncheck this option to avoid problems with the compiler locking JAR files.

- **Verify and/or correct EJBs for the selected server:** Verifies that the classes on the current diagram correspond to the EJB specification supported by the target server (plus individual specifications for some servers). 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 the EJBs.

Note If you have both EJBs supporting EJB 1.1 specification (including CMP EJB1.1) and EJBs supporting EJB 2.0 specification in the same project, uncheck **Verify and/or correct EJBs for the selected server**.

- **Compile the classes referenced in the currently selected diagram:** Compiles the bean, packs it in a JAR file, and stores the JAR in a temporary folder. **Together** uses the standard Sun tools (`javac.exe` and `jar.exe`) for compilation and packing.
- **Generate Deployment Descriptor(s):** Generates deployment descriptors. As soon as the descriptors are ready, they are added to the JAR, WAR, RAR, or EAR archive together with some additional classes.

Note **BEA WebLogic.** **Together** uses the BEA WebLogic tool `ejbc.exe`.

- **Open XML Editor for the generated Deployment Descriptor(s):** Opens the XML Editor so that you can edit the deployment descriptor as it is generated.
- **Pack Modules for Deployment:** Creates deployable JAR, WAR, RAR, or EAR archives.
- **Generate a simple JSP client:** Generates a JSP client for the deployed EJBs. The client is a set of interrelated JSP and HTML files, which you can view in a browser. The intention of this client is to demonstrate access to remote EJB objects through an open interfaces. The semantic information required for JSP client generation is taken from the active diagram of the current project. **Together** provides this feature as a universal testing facility .

It is possible to generate a JSP client from Web application and enterprise application diagrams.

Note Since EJBs reside inside the container and JSPs typically reside outside, the generated JSP client does not support EJBs that do not have remote or home interfaces (it ignores local interfaces). The JSP client does not support message-driven beans either.

- **Generate a command line file with instructions for deployment:** Generates a command file consisting of the sequence of calls to various console tools for the application server. You can subsequently launch this file 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.

Sun EE and Borland Enterprise Server 5.2 process options

There is one process option for Sun EE and Borland Enterprise Server 5.2 in addition to the common process options listed above:

- **Hot Deploy to server** or **Deploy to server:** Does the hot deploy. This step makes use of a bean already compiled and jarred, so you must use this step in conjunction with the others. The server must already be started.

You should compile classes, generate descriptors, and deploy the beans all together, because some tasks use the results of the previous tasks. For example, hot deploy requires a *.jar file to be prepared in a temporary folder.

IBM WebSphere process options

Process options for IBM WebSphere 4.0 are similar to those for IBM WebSphere 3.5. Several options are in addition to the common ones listed earlier.

- **Enable Advanced Single Edition support:** (IBM WebSphere 4.0 only) Check to use the Advanced Single Edition variant of IBM WebSphere 4.0. Do not check this option if you are going to use the Advanced Edition version.
- **Deploy to server:** Deployment executes 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 do support hot deployment. For IBM WebSphere 3.5, the name of this option is **Hot Deploy to server**.

- **Generate executable files for the client:** (IBM WebSphere 3.5 only) Together and WebSphere use different versions of the JDK. WebSphere requires using the IBM JDK for client programs. Thus you cannot make use of Together internal tools. You must generate two batch files, one to compile and the other to run the client program. Subsequently, you can start them separately from Together.
- **Process Servlet(s):** (IBM WebSphere 3.5 only) Check this box to deploy servlets.
- **Launch J2EE Application Client:** (IBM WebSphere 4.0 only) Launches the already deployed application client.

If you use an application client diagram to create your application client, place the shortcut for the application client diagram on the current enterprise application diagram and then deploy.

BEA WebLogic process options

The process options for BEA WebLogic 5.1, 6.0, 6.1, and 7.0 include all of the common options listed earlier as well as an additional one:

- **Hot Deploy to server:** Does the hot deploy. This step makes use of a bean already compiled and jarred, so you have to use this step in conjunction with the others. The server must already be started.

You should compile classes, generate descriptors, and deploy the beans at the same time, because some tasks use the results of the previous tasks. For example, hot deploy requires a *.jar file to be prepared in a temporary folder.

Note All hot deployed beans and deployment descriptors are stored in the Together temporary folder and in the WebLogic home folder. You can also save these files in a special folder. If you choose BEA WebLogic 7.0 as a target application server, Together generates a correct web.xml file according to the Java Servlet specification, Version 2.3.

The following options are available for the BEA WebLogic 4.5.1 application server:

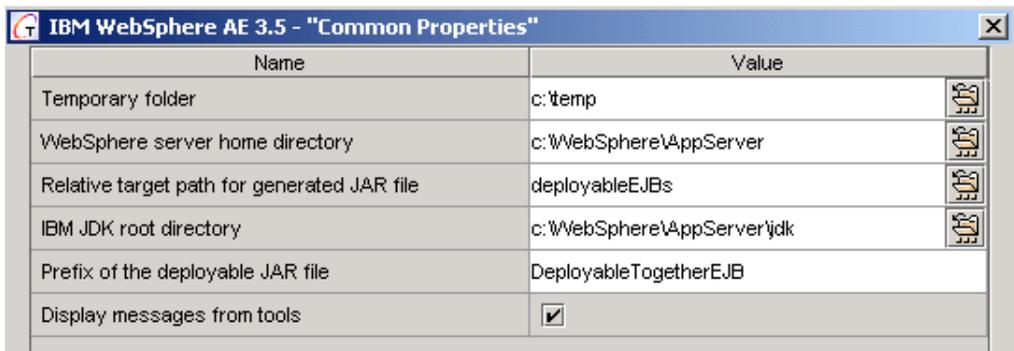
- Verify and/or correct EJBs for the selected server
- Compile the classes referenced in the currently selected server
- Generate Deployment Descriptor(s)
- Hot Deploy to server
- Clear temporary folders before starting the deployment process
- Clear temporary folders before/after deployment is completed

The Common Properties page has fields for locations of folders, the JDK, and servers. For every application server, this page has at least three fields:

- **Temporary folder:** Location of temporary files generated during deployment.
- **Folder for the generated JAR and EAR files or Relative target path for generated JAR file (WebSphere 3.5):** Location of the resulting JAR file and where it will be loaded by the server.
- **<Server> root or home directory:** Location of the server.

Figure 196 shows the common properties page for IBM WebSphere 3.5.

Figure 196 Common properties for IBM WebSphere 3.5



Sun EE common properties

The special Sun EE common properties include the following:

- **JDK 1.3 root directory:** Location of the JDK.
- **J2EE root directory:** Location of the directory `TGH/bundled/j2ee`.

Borland Enterprise Server 5.2 common properties

The special Borland Enterprise Server 5.2 common properties include the following:

- **JDK 1.4 root directory:** Location of the JDK.
- **Server root directory:** Location of the Borland Enterprise Server 5.2 directory.

IBM WebSphere common properties

Common properties for IBM WebSphere include the following:

- **IBM JDK root directory:** (WebSphere 4.0) Location of JDK 1.3.
- **Prefix of deployable JAR file:** (WebSphere 3.5) The name assigned to the JAR file (or files) generated in the *Compile Classes* task from the selected diagram. While preparing for deployment, Together creates an additional JAR file whose name has this prefix.
- **Display messages from tools:** (WebSphere 3.5 only) Displays tool messages in the Message pane.

BEA WebLogic common properties

The BEA WebLogic common properties has one field in addition to the common ones listed above.

- **JDK 1.3 root directory:** The JDK location.

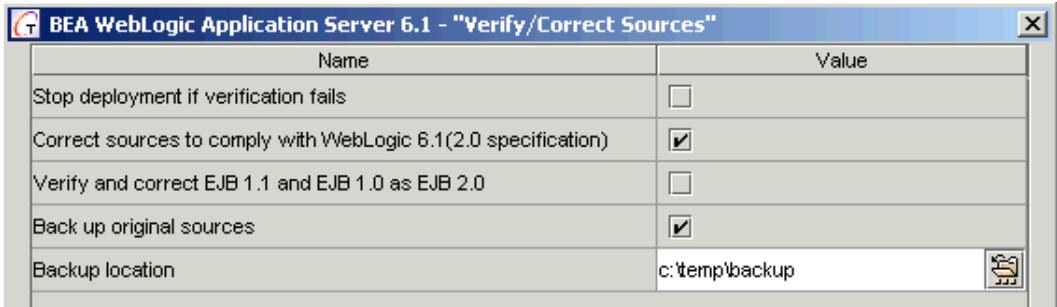
Important Set paths to `jdk1.2` or later for BEA WebLogic 5.1, and to `jdk1.3` for BEA WebLogic 6.0 and 6.1. You cannot use `jdk1.2` for BEA WebLogic 6.0, 6.1, or 7.0.

Verify / Correct Sources page

The Verify/Correct Sources page has fields for terminating deployment if verification fails, correcting sources to make them compliant with the application server, and making backup copies. [Figure 197](#) shows the Verify/Correct Source page for BEA WebLogic 6.1.

You should use the fields on this page to provide verification or correction of the source Java code.

Figure 197 Verify/Correct page for WebLogic 6.1 application server



Options for verification and correction include the following:

- **Stop deployment if verification fails**
- **Correct source code to comply with server specification:** Together can change code as needed.

Note IBM WebSphere 3.5 supports the EJB 1.0 specification. Later specifications require that any create methods of a home interface should return the primary key type, while the 1.0 specification requires that create methods return void. If you deploy a project originally targeted to a later specification to WebSphere 3.5, Together replaces all changed types with void.

- **Verify and correct EJB 1.1 and EJB 1.0 as EJB 2.0:** (BEA WebLogic and Sun EE only) If there are 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.

Note **Sun EE.** If your project has EJBs satisfying different EJB specifications (EJB 1.0, 1.1, 2.0), you can deploy it to Sun EE Reference Implementation without verification and correction. There is only one exception: you cannot deploy CMP bean 1.0 together with EJBs satisfying EJB 1.1, 2.0 specification.

- **Back up original sources:** Backs up the original sources when Together changes the source code to comply with the server specification.
- **Backup location:** Directory for backup copies of the source code.
- **Uninstall already deployed application:** Check if you are going to deploy an application with the same name as an existing application deployed to the same application server.

Simple JSP Client Generation page

The Simple JSP Client Generation page displays only if you check *Generate a simple JSP client* on the first page of the Expert. [Figure 198](#) shows the Simple JSP Client Generation page for BEA WebLogic 5.1.

These are the common entries for all servers:

- **Show JSP client in default Internet browser:** Launches your default Internet browser and displays the start page.
- **Root path for JSP storage:** The root of the file hierarchy, which can be accessed externally. You must place any HTML or JSP files that you want to access under the document root.

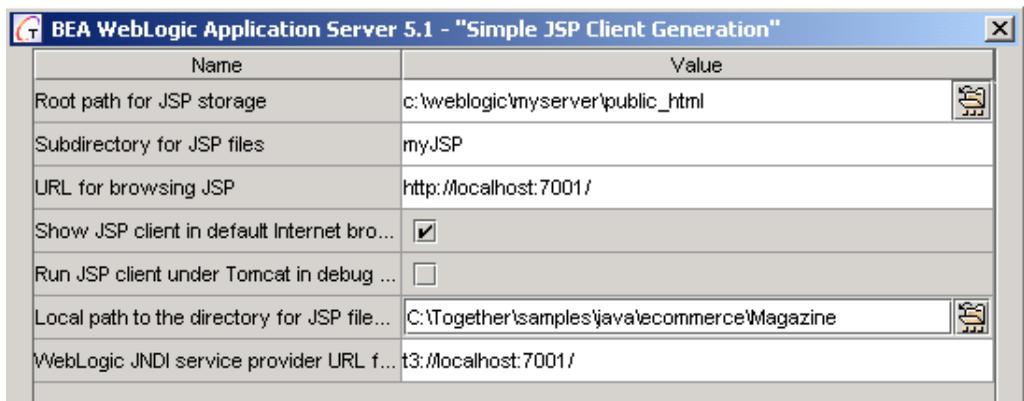
Note By default, the WebLogic public directory resides at `$WebLogic_home$/myserver/public_html`

Options for WebSphere 3.5

The additional JSP generation fields for WebSphere 3.5 are as follows:

- **Subdirectory for JSP files:** Subdirectory of the public directory for the JSP files. 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:** 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 public directory causes an access error. By default, the base URL is `http://localhost`.
- **WebSphere JNDI service provider URL:** (WebSphere 3.5 only) Establishes a connection between a JSP client and an EJB server. The value 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`.

Figure 198 Simple JSP Client Generation page for BEA WebLogic 5.1



Options for WebLogic 5.1

The JSP client options for WebLogic 5.1 include these two, which are equivalent to their WebSphere counterparts listed above.

- **Subdirectory for JSP files**

- **URL for browsing JSP:** By default, the base URL is `http://localhost:7001`.

WebLogic 5.1 has additional options for running under debug mode.

- **Run JSP client under Tomcat in debug mode:** The generated JSP client runs under Tomcat rather than the 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.

If you check *debug mode*, the Navigation Page of the JSP client runs automatically. In this case, the *Show JSP client in default Internet browser* option is ignored.

- **Local path to the directory for JSP files for Tomcat:** Folder of the generated JSP files for running under Tomcat. The folder is an alternative to the one in the WebLogic public directory. Its default value is the root directory of the current project.
- **WebLogic JNDI service provider URL for Tomcat:** 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.

Options for Sun EE, Borland Enterprise Server 5.2, WebSphere 4.0, WebLogic 6.0, 6.1, 7.0

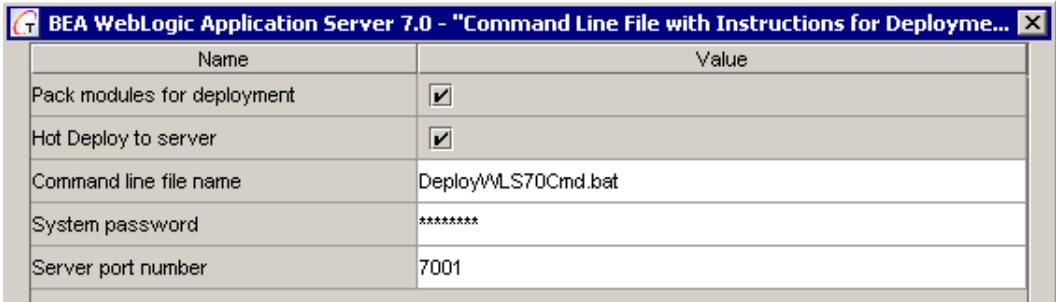
The following options are in addition to the ones common for all servers.

- **Name of the target Web Application Diagram:** The Web application diagram where Together will store the generated JSPs.
- **Open target Web Application Diagram:** Opens the diagram named above.
- **Clear target directory before deploying**
- **Web server host name:** Default value for WebLogic is `localhost`.
- **Web server port number:** Default value for WebLogic is `7001`.

Command Line File with Instructions for Deployment page

This page displays if you check the box *Generate a command line file with instructions for deployment* on the first page of the Expert. [Figure 199](#) shows this page for BEA WebLogic 7.0 application server.

Figure 199 Command Line File page for WebLogic 7.0



Name	Value
Pack modules for deployment	<input checked="" type="checkbox"/>
Hot Deploy to server	<input checked="" type="checkbox"/>
Command line file name	DeployWLS70Cmd.bat
System password	*****
Server port number	7001

The fields are self-explanatory. They vary according to the application server. All of them however, have this field:

Command file name: Name of file with deployment instructions.

When you check an option in this page, Together adds a command for the associated task in the command line file.

Server-specific J2EE Deployment Expert pages

Pages of the J2EE Deployment Expert vary according to the application server. The previous section discussed pages and options common to several servers. This section discusses the pages that vary among servers according to title and intent or according to fields.

Sun EE Deployment Properties page

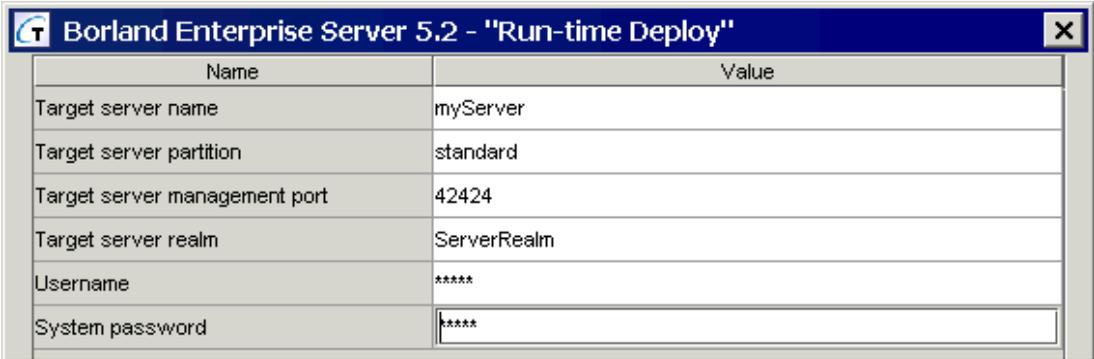
The Deployment Properties page for Sun EE has information necessary for Together to communicate with the application server. The fields are as follows.

- **Server host name:** Default is `localhost`.
- **Web server port number:** Default for Sun EE is `8000`.
- **Uninstall already deployed application:** Check if you are going to deploy an application with the same name as an existing application already deployed.

Borland Enterprise Server 5.2 Run-time Deploy properties page

The Run-time Deploy properties page has information required for hot deployment. [Figure 201](#) shows this page for Borland Enterprise Server 5.2.

Figure 200 Run-time Deploy page for Borland Enterprise Server 5.2



Name	Value
Target server name	myServer
Target server partition	standard
Target server management port	42424
Target server realm	ServerRealm
Username	*****
System password	*****

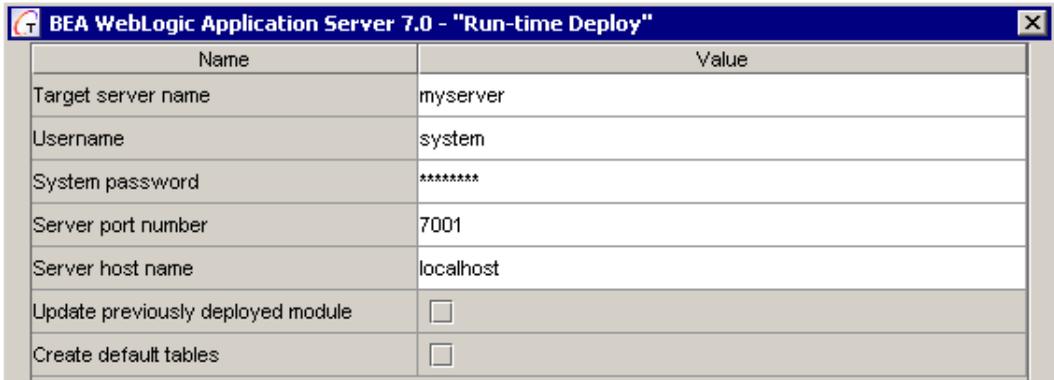
Fields and options on this page include the following:

- **Target server name:** the name of Borland Enterprise Server 5.2.
- **Target server partition:** Default is `standard`.
- **Target server management port:** The default server port is 42424. If you do not know the server port number, you can find it in the Borland Management Console.
- **Target server realm:** Default is `ServerRealm`.
- **Username:** Username for your Borland Enterprise Server 5.2 installation. Default is `admin`.
- **System password:** Defined during Borland Enterprise Server 5.2 installation. Default is `admin`.

BEA WebLogic Run-time Deploy properties page

The Run-time Deploy properties page has information required for hot deployment. [Figure 201](#) shows this page for BEA WebLogic 7.0 applications server.

Figure 201 Run-time Deploy page for BEA WebLogic 6.1



Name	Value
Target server name	myserver
Username	system
System password	*****
Server port number	7001
Server host name	localhost
Update previously deployed module	<input type="checkbox"/>
Create default tables	<input type="checkbox"/>

Fields and options on this page include the following:

- **Target server name:** WebLogic server name.
- **Username:** Username for your WebLogic installation. Default is `system`.
- **System password:** Defined during BEA WebLogic installation.
- **Server port number:** The default server port is 7001. 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:** By default, the server host is `localhost`. Because the 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, it is possible to update any modules that have already been deployed. Check this box to update modules during deployment.
- **Create default tables:** (WebLogic 7.0 only) Corresponds to the deployment descriptor tag `<create-dbms-default-tables>`.

Before proceeding to the next page of the expert, make sure that the server is available at the selected port and host and that the current application server has been started.

IBM WebSphere 3.5 pages

The IBM WebSphere 3.5 deployment options differ significantly from those for other servers.

EJB Deployment Properties page

[Figure 202](#) shows the EJB Deployment Properties page for IBM WebSphere 3.5. The deployment options include the following:

- **Admin Node Name:** Local computer name (by default, assigned automatically).
- **Dependent Classpath:** 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 and its name is Default Server. If you use a different application server, enter its name here. If the server with that name does not exist, it is created.
- **Application Server Command Line Arguments / Application Server Classpath:** Corresponds to the properties on the 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 and its name is Default Container. If you use a different container, enter its name in this field. If the container with this name does not exist, it is created.
- **User ID / Password:** (CMP beans only) You can use the user name and password assigned during WebSphere installation.
- **Create Table for CMP EJBs:** (CMP beans only) Creates new tables in the server persistence repository designed to support Container Managed Persistence. The structure of the tables corresponds to the beans being deployed.
- **Port number of the WebSphere node:** Port 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:** Path to the WebSphere application server installation root.
- **Launch server in debug mode:** Runs the server in debug mode and attaches to it from the debugger process in Together. This option does not restart the server in debug mode. Instead, it adds special command-line arguments to the application. You must restart the application server from the WebSphere Administrative Console.

Note Several additional libraries are required to launch WebSphere in debug mode. See WebSphere InfoCenter documentation for details.

- **Debug port number:** Port number for communication between WebSphere and the external debugger. The server and debugger must use the same port number. You can use the default port number 8787.
- **Generate WLM Jar for Beans:** Creates a work load management file.

Figure 202 Deployment Properties page for IBM WebSphere 3.5

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	<input type="checkbox"/>
Port number of the WebSphere node	900
Target WebSphere server directory	d:\WebSphere\AppServer
Launch server in debug mode	<input type="checkbox"/>
Debug port number	8787
Generate WLM JAR for Beans	<input type="checkbox"/>
Stop server and remove existing Bean...	<input checked="" type="checkbox"/>
Start Beans/Servlets after deployment	<input checked="" type="checkbox"/>

Servlet Deployment Properties page

This page opens only if *Process Servlet(s)* is checked. You can use this page for creating servlets and WebSphere service servlets. [Figure 203](#) shows the Servlet Deployment Properties page.

The properties on this page include the following:

- **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, and its name is `default_host`. It is associated with the `localhost`. To use a different virtual host, specify its 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. If you use a different servlet engine, enter its name here. If the engine with the specified name does not exist, it is created.

Figure 203 Servlet Deployment Properties for IBM WebSphere 3.5

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	<input type="checkbox"/>
Create "JSP 1.0" Support Servlet	<input type="checkbox"/>
Create the serving Servlet "File"	<input type="checkbox"/>
Load Servlets at startup	<input checked="" type="checkbox"/>
Enable Servlets	<input checked="" type="checkbox"/>
Enable Debug	<input type="checkbox"/>

- **Web Application name:** Each WebSphere servlet engine can hold several Web applications. The J2EE Deployment Expert uses the `default_app` Web application. If you are going to use a different Web application, enter its name here. 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, which 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 except that it does not refer to any actual directory; instead it is a Web resource alias for the Web application. The 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 is `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:** The part of Web path to access a servlet. This is appended to the relative path to servlet, described above. For example, if the virtual host is `localhost:900`, the 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, resulting in the loss of all deployed objects. 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) EJBs, Servlets*, and *Start EJB's, Servlets* on the page *EJB Deployment Properties*.
- **Create "JSP 1.0" Support Servlet:** If the project uses JSPs, this 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`.
- **Create the serving Servlet "File":** If your project uses HTML resources, 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:** Loads a servlet when the application server starts rather than when the servlet is requested. (Default is unchecked.)
- **Enable Servlets:** Indicates whether the servlet is available to handle requests. (Default is checked.)
- **Enable Debug:** Indicates whether to start the servlet in Java debug mode. (Default is unchecked.)

Client Properties page

The Client Properties page contains properties required to generate start and compile files for the client task (if the corresponding box is checked on the first page of the Expert). [Figure 204](#) shows the Client Properties page.

Properties on this page include:

- **Main class of the client, including package names:** Name of the class that contains the main method to run the client application. Automatically generated batch files refer to this class.
- **Path to the client sources:** Fully qualified path to the directory where client source files are stored.

- **Path to the target directory:** Fully qualified path to the directory where compiled files will be placed.

Figure 204 Client Properties page for WebSphere 3.5

Name	Value
Main class of the client, including package names	Main
Path to the client sources	.
Path to the target directory	C:\Together\out\magazine
Deployed JAR file name	deployableEJBs\DeploySession1 Bean.jar

IBM WebSphere 4.0 pages

Advanced Single Edition Properties page

The Advanced Single Edition Properties page opens only if the *Advanced Single Edition support* option on the first page of the Expert is checked. [Figure 205](#) shows the two options on the page:

- **Restart WebSphere server:** Restarts IBM WebSphere 4.0 Advanced Single Edition.

Note IBM WebSphere 4.0 Advanced Edition supports hot deployment, but Advanced Single Edition does not support hot deployment. You should restart the current application server for deployment.

- **Server configuration file:** Saves information about EJBs in the corresponding server configuration file `server-cfg.xml`. Set the path to this configuration file.

Important 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.

Figure 205 *Advanced Single Edition Properties* page for IBM WebSphere 4.0

Name	Value
Restart WebSphere server	<input type="checkbox"/>
Server configuration file	D:\WS_4_0\config\server-cfg.xml

Deployment Properties page

The Deployment Properties page provides fields for hot deployment and other tasks. The options include:

- **Uninstall previous application:** Check this if there is an older deployed application with the same name on the current application server.
- **Install new application:** Check this if you are going to deploy a new application.
- **Node name:** Local computer name (by default, assigned automatically).
- **Server name:** Name of the current application server in the IBM WebSphere Administrative Console.
- **Virtual host name:** Name of the host in the IBM WebSphere Administrative Console.
- **JNDI name of the data source / Schema name:** 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):** Name and password defined during IBM WebSphere installation.

Making transitions among application servers

It is a common task to change application servers after you have implemented your project and deployed your application to an application server. This presents two major problems:

- Different servers support different EJB specifications (EJB 1.0, 1.1, 2.0).
- Each application server has its own specific features.

This section discusses transitioning among different EJB specifications and among different application servers.

Transitions among specifications

If you want to redeploy your application to a new application server, you should first determine 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 need to consider only the specific features of the application server.

If the new application server does not support the EJB specification for the project's target server (for example, the new server supports a later EJB specification), you must 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. Entity CMP beans require the most extensive code corrections to make a transition from one specification to another.

Together determines which EJB specification this application server supports, and verifies and corrects the EJBs according to the specification (EJB 1.0, 1.1, or 2.0).

Automatic verification and correction of EJBs

Together can automatically correct most of the EJB code that is specification sensitive.

To automatically verify and correct EJBs:

1. Open the diagram for the original deployment.
2. From the main menu, choose **Deploy | J2EE Deployment Expert**.
3. Choose the new application server from the list at the top of the Expert.
4. Check *Verify and/or correct EJBs*. You can clear all other options if you merely want to change the code without attempting to deploy.
5. Continue through the remaining pages as usual.

Together verifies and corrects only the methods described in the EJB x.x Specification. For an additional discussion of using verification and correction, see [“EJB verification and correction” on page 597](#).

Manual correction of EJBs

Together cannot make all necessary corrections to your code, especially if you move from an earlier EJB specification to a later one. For example, if your EJB contains business methods (which are not included in the EJB specifications), Together verifies and “corrects” them if they do not satisfy the EJB specification. When Together corrects a business method, however, it replaces the body of the method with the default (empty) body.

Together saves a backup of all Java classes of the project in the backup folder. You can completely rewrite the code for a “corrected” business method, or simply use the appropriate 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, manually making any necessary adjustments to meet the selected EJB specification.

Moving from EJB 2.0 to EJB 1.0

To understand how Together corrects code when you transition from the later EJB 2.0 specification to the earlier EJB 1.0 specification, consider a simple sample from BEA WebLogic 6.1 (EJB 2.0). Attempt the verification and correction by using IBM WebSphere 3.5 (EJB 1.0) as the target application server.

When verifying and correcting code, Together removes features of EJB 2.0 that not defined in EJB 1.0 specification and adds features of EJB 1.0 specification. Together automatically makes the following changes in the EJB implementation:

- Removes `EJB Exceptions`.

- Adds `RemoteException`.
- Changes the returned value of the `ejbCreate` method to `void` for CMP EJBs and removes the `return` statement in the method body.
- Changes `Collection` class to `Enumeration` in `findNullAccounts` and `findBigAccounts` methods.
- Changes transaction attributes.
- Removes the `abstract` modifier 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 change any EJB components that are defined in the later specification but not defined in the earlier one (such as message-driven beans, which are defined only in the EJB 2.0 specification).

After automatic correction, you should analyze your code and correct it manually as needed.

Moving from EJB 1.0 to EJB 1.1

You can use the same technique discussed in the previous section to understand how Together corrects code when moving from EJB 1.0 to EJB 1.1 specifications. Consider a simple sample from IBM WebSphere 3.5 (EJB 1.0 specification). Attempt the verification and correction by using BEA WebLogic 5.1.0 (EJB 1.1 specification) as the target application server. The most dramatic changes occur in entity EJBs.

Transitions among application servers

When you transition a project from one application server to another, expect to make changes in the following places:

- **Server-dependent deployment descriptors:** The J2EE Deployment Expert generates all deployment descriptors, including server-dependent ones. You can edit these descriptors only during the deployment process.
- **Code of the client connected with the new application server:** You must change the `getInitialContext()` method (`Context.INITIAL_CONTEXT_FACTORY`) in the code of the client, according to the target application server. You must also change the `localhost` in the `main()` method (for examples, see [“Developing and deploying clients for various servers” on page 699](#)).
- **EJB Java code, if the two application servers support different EJB specifications:** Changes in the code connected with the difference between EJB specifications are described in [“Transitions among specifications” on page 696](#).

- **Properties of the targeted 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 may 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 depends on the selected application server. For example, the simple client for the `HelloWorld` project for BEA WebLogic application server (`$TGHS/samples/java/ejb/J2ee/HelloWorld`) is implemented with the following 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.HelloHome"), 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 code described below:

- *findByPrimaryKey* method: If your project contains CMP Entity EJBs, it uses the `findByPrimaryKey` method. The signature of the `findByPrimaryKey` method varies among different EJB specifications.

With the EJB 2.0 specification, you should use

```
findByPrimaryKey(id).
```

In the EJB 1.0 specification you should use

```
findByPrimaryKey(new <Name>PK(id))
```

where <Name>PK is the returned type of Primary Key.

- **localhost:** You should correct the `main()` method according to the application server. For example, for BEA WebLogic 6.0, use:

```
String url ="t3://localhost:7001"
```

For IBM WebSphere 3.5, use:

```
String url ="iiop://localhost:900"
```

- ***getInitialContext* method:** The `getInitialContext()` method varies among application servers. 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")
```

- **lookup:** In CMP EJBs, the lookup method varies among application servers. For example, for BEA WebLogic 6.0, you should write this in lookup home:

```
Object home = (<Name>Home) ctx.lookup("containerManaged.AccountHome")
```

For IBM WebSphere 3.5, you should write:

```
Object home = (<Name>Home) ctx.lookup("<Name>Home")
```

Encoding in deployment descriptors

Deployment descriptors may contain special symbols or non-Latin characters. XML files use UTF-8 or UTF-16 encoding in general, but you can specify another encoding if necessary.

To provide special encoding in deployment descriptors:

1. Open the Options dialog from the main menu by choosing **Tools | Options | <level>**.
2. Click *General* on the left pane. Find **Encoding** in the right pane.
3. Choose your preferred encoding from the drop-down list of the Encoding option.
4. Click *EJB* on the left pane.
5. Uncheck **Use the standard encoding for deployment descriptors** in the right pane.

The selected encoding will be the default encoding for all Together files that contain text, including deployment descriptors.

Because XML processors read only UTF-8 and UTF-16 encodings by default, a deployment descriptor must specify any other encoding in the text declaration. Both the J2EE Deployment Expert and the Web services Deployment Expert in Together add this corresponding prologue to deployment descriptors. For example, if you choose Cp1251 as the default encoding, the Web services Deployment Expert adds the following lines to the beginning of the `web.xml` file when it generates deployment descriptors:

```
<?xml version="1.0" encoding="Cp1251"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

Important

The XML parser for your target application server may not support the current encoding. For example, the BEA WebLogic server parser does not support Cp1252, which is the default encoding in Together. Please note that this depends on your target application server and the particular encodings it supports. If the target application server does not support the current encoding, you could encounter problems such as server crashes.

If you want to retain your encoding in the rest of your project, you can “turn off” encoding for deployment by using a special option that does not add the encoding information to the deployment descriptor. In this case the target application server will consider the text of the deployment descriptor as written with standard UTF-8 or UTF-16 encoding. One result is that you will not be able to use any special symbols or characters in the deployment descriptor.

To use the standard encoding for deployment descriptors:

1. Open the Options dialog from the main menu by choosing **Tools | Options | <level>**.
2. Click *EJB* on the left pane.
3. Check **Do not write encoding declaration for deployment descriptors** on the right pane.

WEB SERVICES

- Chapter 41, “Creating Web Services”
- Chapter 42, “Deploying a Web Service”
- Chapter 43, “UDDI Browser”

Creating Web Services

This chapter discusses transforming classes and EJBs into Web services. It includes the following topics:

- “Supported platforms” on page 705
- “Creating Web services from existing code” on page 706
- “Exposing methods to clients” on page 713
- “Generating WSDL files from Web services” on page 714
- “Generating Web service clients” on page 715

Supported platforms

Together currently targets the following platforms for Web services:

- Apache SOAP (<http://xml.apache.org/soap>) To use Apache SOAP, you must install an application server such as Apache Tomcat.
- BEA WebLogic 6.1, 7.0 (<http://e-docs.bea.com/wls/docs61>, <http://e-docs.bea.com/wls/docs70>)
- IBM WebSphere AES 4.0 (<http://www-4.ibm.com/>)
- Borland Enterprise Server 5.2 (<http://info.borland.com/techpubs/books/bes/pdfs52/>)

Different application servers offer different approaches to Web services. You should read the product documentation on your application server to understand the extent and nature of its Web service support. For example, Apache SOAP supports Web services from ordinary classes while WebLogic supports web services from EJBs only.

- The code that Together generates when you create a Web service or a Web service client depends on the target server of the project for Web services.

To choose the target server of the project for Web services:

1. From the main menu, choose **Tools | Options | Project** or **Tools | Options | Default**.
2. In the resulting dialog box, choose *Web Services* on the left.
3. Choose the target server from the *Application Server* list on the right.
4. Click **Ok** to finish the process.

Creating Web services from existing code

This section discusses creating Web services from an ordinary Java class and from the implementation class of a session EJB. The steps for transforming either into a Web service require first preparing it as a Web service.

To prepare a class or an EJB as a Web service:

1. Open the inspector for the class or the bean. Right click the class or bean on the diagram and choose **Properties** from the right-click menu.
2. Click the **Properties** tab at the top of the inspector to bring it to the front.
3. At the bottom of this tab, check *Web Service*.

The result is that a new **Web Service** tab displays at the top of the inspector.

Creating a Web service from a class

This section is applicable to Apache SOAP and IBM WebSphere AES 4.0, and Borland Enterprise Server 5.2 only. (BEA WebLogic 6.1 does not support Web services created from ordinary classes.)

The **Web Service** tab at the top of the class inspector for a Web service has three subtabs at the bottom: **Properties**, **Server-specific Properties**, and **Type Mapping Properties**. These tabs provide a convenient mechanism for specifying the Web service properties necessary for the corresponding WSDL and deployment descriptor.

Properties tab of the Web service inspector

The **Properties** tab lists the context and the namespaces for Web services. [Figure 206](#) shows the **Properties** tab for a class named `MyService`.

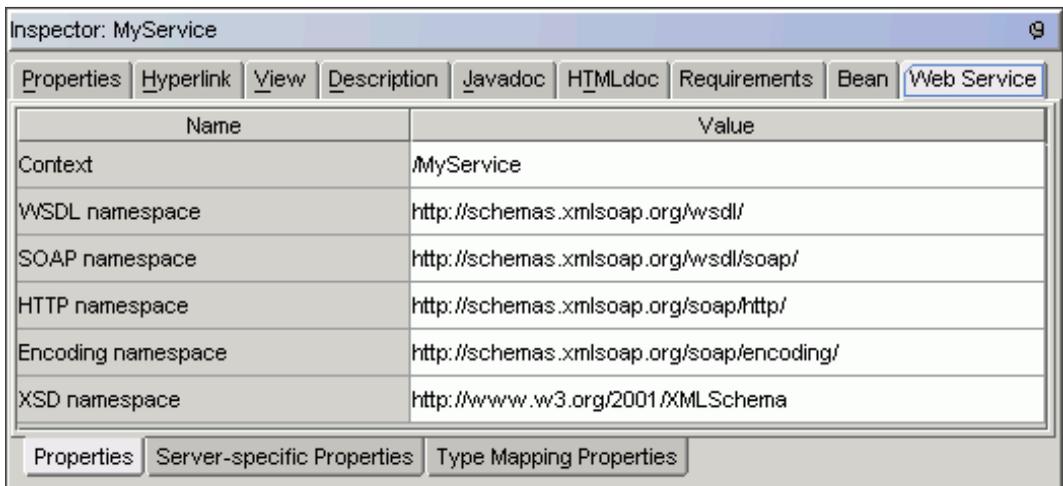
There are six fields on the tab:

- **Context.** The relative path to the Web service (relative to the base path where all Web services are stored on the application server).

Note If you are deploying to Tomcat you can leave this field empty. In Tomcat, all Web services have the same path, such as `http://localhost:8080/SOAP/servlet/rpcrouter/`.

- **Namespaces.** For generating the Web Services Description Language (WSDL) file for the Web service.
 - **WSDL namespace.** WSDL namespace for WSDL framework
 - **SOAP namespace.** WSDL namespace for WSDL SOAP binding
 - **HTTP namespace.** WSDL namespace for WSDL HTTP GET and POST binding.
 - **Encoding namespace.** Encoding namespace as defined by SOAP 1.1.
 - **XSD namespace.** Schema namespace as defined by XSD.

Figure 206 Properties on the Web Service tab of the class inspector



The default namespace settings are determined by the project options.

To change the default namespace settings:

1. From the main menu, choose **Tools | Options | Project** or **Tools | Options | Default**.
2. In the resulting dialog box, expand *Web Services* on the left.
3. Choose *Namespaces*.
4. Change the default values as desired on the right.
5. Click **Ok** to save the settings and close the dialog.

Note For WSDL specifications, refer to <http://www.w3.org/TR/wsdl>.

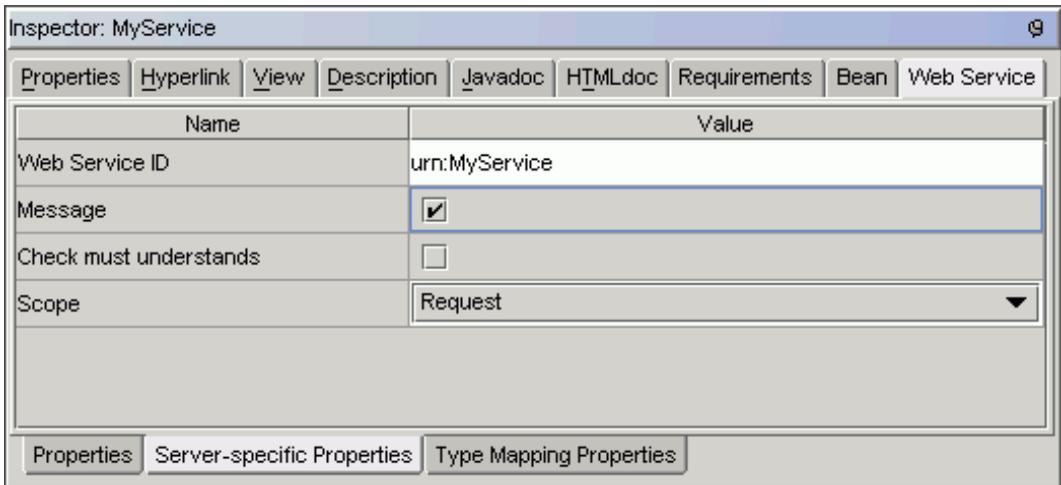
Server-specific Properties tab of the Web service inspector

A deployment descriptor for a Web service that uses only standard Java classes has the following form:

```
<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:service>
```

Together generates the mandatory properties of the deployment descriptor automatically when you deploy a Web service. Some of these properties are on the **Server-specific Properties** tab, which is shown in [Figure 207](#).

Figure 207 Server-Specific Properties tab of the inspector of a class Web service



The fields on the **Server-specific Properties** tab are as follows:

- **Web Service ID.** Corresponds to *service-urn* in the deployment descriptor, which is the unique identifier for Web service on the server. (This is required for Tomcat.)
- **Message.** Specifies that the class is a message-type Web service. The parameter *message* in a deployment descriptor indicates this service is a document-oriented service (rather than a procedure-call invoked service).
- **Check must understands.** Indicates whether the processing of a SOAP header block is mandatory or optional at the target SOAP node. *CheckMustUnderstands* is an optional Boolean attribute of a deployment descriptor. If its value is true,

the server should throw an exception when there are incorrect SOAP headers that are marked as *mustUnderstand*. If its value is false, the *mustUnderstand* parameter is ignored.

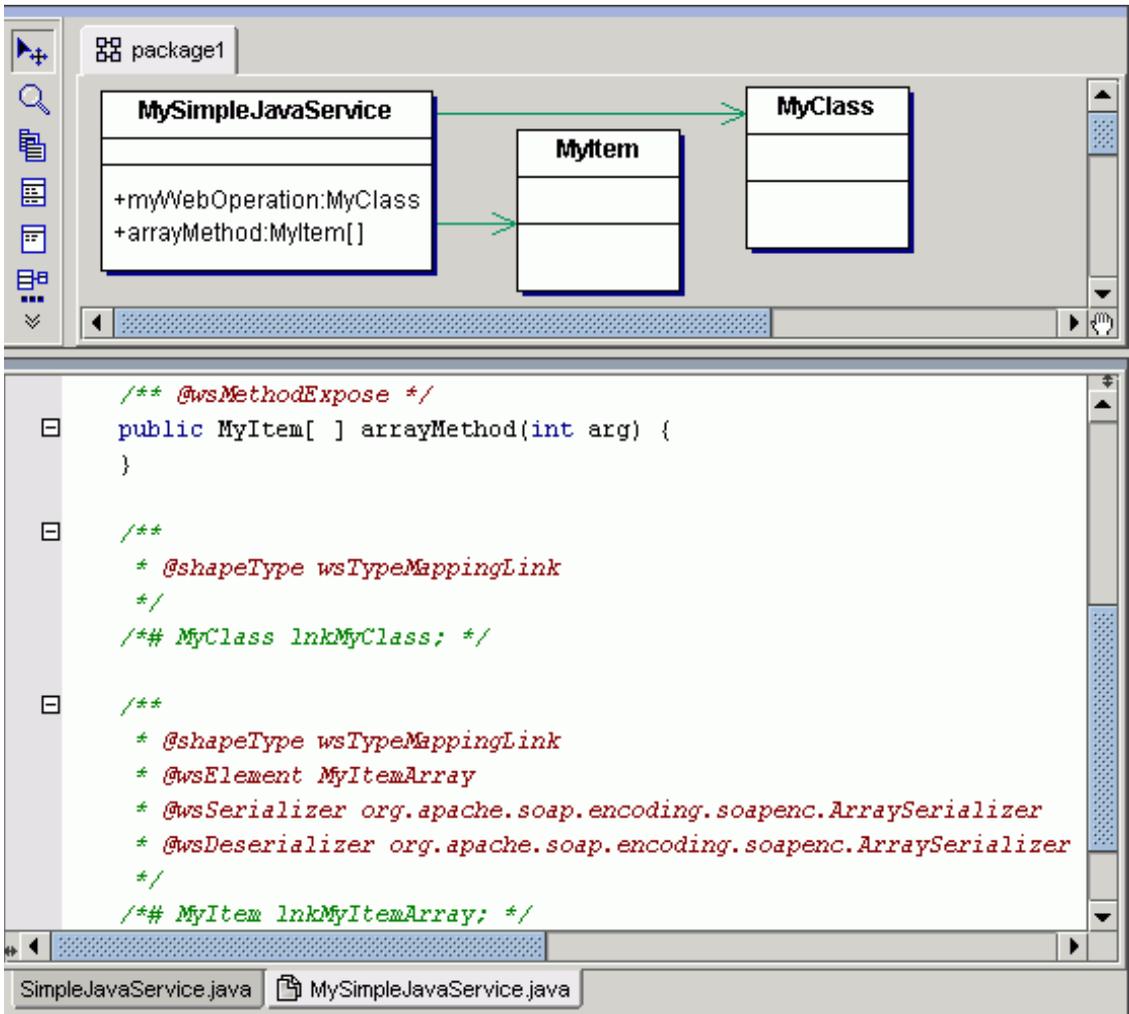
- **Scope.** Defines the lifetime of a Web service. It can have one of the following values:
 - **Request.** The Web service is accessible only during a request. The object is removed after this request is completed.
 - **Session.** The Web service is accessible during an HTTP session.
 - **Application.** The lifetime of the Web service depends on the lifetime of the corresponding application. The Web service is accessible until the servlet which is servicing the requests is terminated.

Type Mapping Properties tab of the Web service inspector

The *types* element of a WSDL file encloses data type definitions that are relevant for the exchanged messages. Apache SOAP uses type mappings to determine how Java class types are marshalled to and from XML.

[Figure 208](#) shows mappings from `MySimpleJavaClass` to `MyItem` and to `MyClass`. The mapping to `MyClass` is a complex type. The mapping to `MyItem` is an array of complex types. The code has a stub for `arrayMethod` plus Javadoc comments for the mappings.

Figure 208 Web service mappings



To map a Web service to a Java class:

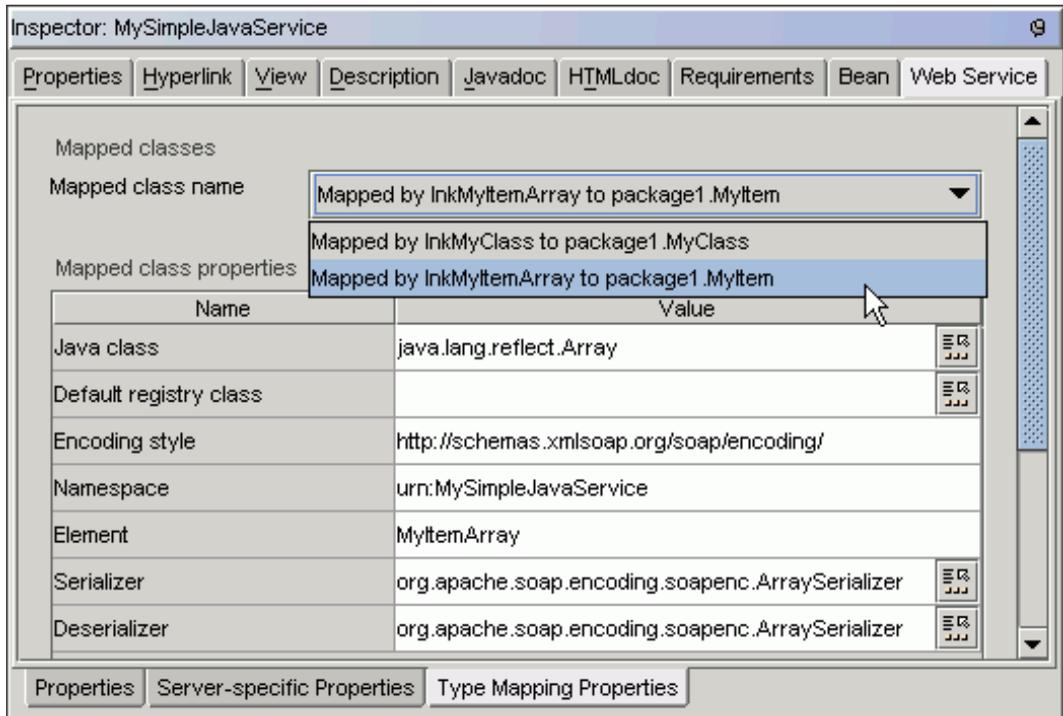
1. Create a Web service and a Java class on a class diagram.
2. Create a Web-exposed method in the Web service. Right click the Web service and choose **New | Web Exposed Method**.
3. Use the in-place editor to change the return type of the method to the name of the Java class (with brackets, [], for an array mapping).

The diagram displays a green symbolic link from the Web service to the class. There are new Javadoc comments in the Web service code as well. (See [Figure 208](#).)

The Type Mapping Properties tab describes types that have no standard representation in XSD. This tab contains information only if the Web service has an exposed method that returns a Java class or has a Java class parameter.

Figure 209 shows the The **Type Mapping Properties** tab on the Web service class inspector for `MySimpleJavaClass`.

Figure 209 Type Mapping Properties tab



The **Type Mapping Properties** tab contains the following fields:

- **Mapped class name.** Name of the symbolic link and the supplier Java class.
- **Java class.** The name of the Java class that is reserved in Serializer and Deserializer. Instances of this class are mapped to XML data and vice versa.
- **Default registry class.** Default UDDI registry.
- **Encoding style.** URI for encoding style of serialization (such as SOAP Encoding, as shown in Figure 209.)
- **Namespace.** Urn of the Web service.
- **Element.** ComplexType name.

- **Serializer.** The Apache SOAP serializer to describe the fields of the Java class to XML. (Complex classes will be referred to by this name in SOAP.) The default value is `BeanSerializer` for complex types and `ArraySerializer` for arrays of complex types.
- **Deserializer.** The Apache SOAP deserializer for decoding from XML to the fields of a Java class. The default values are the same as for serializers.

Creating a Web service from an EJB

Together enables you to create Web services from stateless session EJBs, including RPC-style (remote procedure call) Web services and message-style Web services. BEA WebLogic 6.1, 7.0, and Borland Enterprise Server 5.2 support EJB-based Web services.

RPC-style Web service

An RPC-style Web service uses a stateless session EJB.

To create an RPC-style Web service from a stateless session EJB:

1. Create a stateless session EJB.
2. Starting with the bean class, follow the instructions in [“Creating a Web service from a class” on page 706](#).

Alternatively, you can apply a pattern to a stateless session bean, using the instructions on [“Creating Web services from existing code” on page 706](#) below.

Message-style Web service

You can create a message-style Web service on an EJB assembler diagram by using its Message Web Service button ().

The Message Web Service inspector enables you to specify the properties of the Web service. Access to the inspector is as usual: right click the *message Web service* node and choose **Properties**.

The **Properties** tab of the message Web service inspector contains the following fields:

- **Web Service name.** Name of the current Web service.
- **Action.** Whether the client that uses this message-style Web service is a receiver or sender of XML data to the JMS destination. Choose *send* or *receive* from the list.
- **Destination JNDI name.** The JNDI name of a JMS topic or queue.
- **Destination type.** JMS destination type. Choose **topic** or **queue** from the list.
- **Connection factory JNDI name.** The JNDI name used to create a connection to the JMS destination.

- **URI.** URI used by clients to invoke the Web service. The full URL to access the Web service is [protocol]://[host]:[port][context][uri]

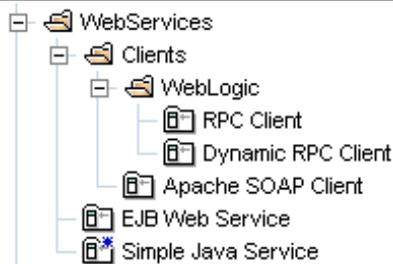
Creating Web services by using patterns

Together provides two patterns for creating Web services.

To use a pattern for creating a Web service:

1. Open the class diagram. Click the **Class by Pattern** button on the toolbar and click the diagram.
2. In the Choose Pattern dialog, expand *WebServices* on the left. Choose *EJB Web Service* or *Simple Java Service* as desired. [Figure 210](#) lists the built-in Web service patterns from the left section of the dialog.
3. Enter the name of the Web services on the right. For an EJB Web service, choose *Stateless Session EJB* for the *EJB Type*.
4. Click **Finish** to create the Web service and display it on the diagram.

Figure 210 Patterns for Web services



Exposing methods to clients

A client has access to a method of a Web service only if the method is public and exposed.

To expose a method in a Web service:

1. Open the method's inspector by right clicking the method and choosing **Properties**.
2. In the inspector, click the **Web Service** tab to bring it to the front. (The **Web Service** tab is present only when the class is prepared as a Web service, as described in "[Creating Web services from existing code](#)" on page 706.)
3. Check *Expose* for the method of a simple class.
4. Make sure that *Expose in remote interface* or *Expose in local interface* is checked for a stateful session EJB.

When you deploy the Web service, Together lists this exposed method among the *exposed-methods* in the deployment descriptor.

Generating WSDL files from Web services

WSDL files provide the interface between Web services and their clients. Together can generate WSDL files in two different ways, which are described in this section.

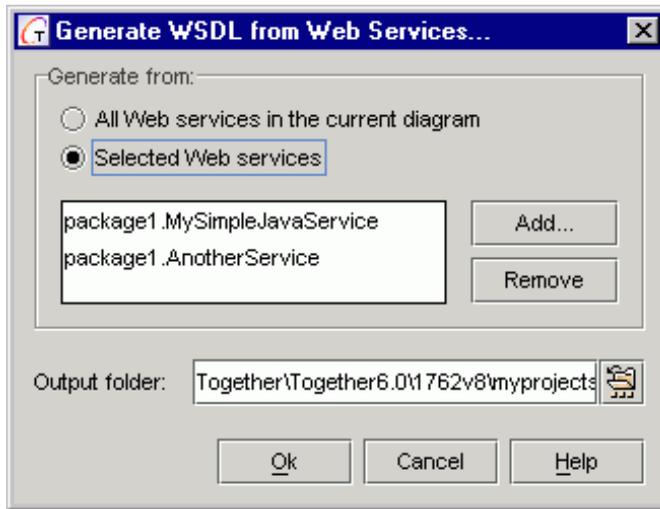
To generate a WSDL file from a collection of Web services on a class diagram:

1. Open the class diagram containing the Web services.
2. From the main menu, choose **Tools | Web Services | Generate WSDL from Web Services**.
3. In the resulting dialog, specify which Web services to use for generating WSDL files. Choose *All Web services in the current diagram* or *Selected Web services*. If you choose *Selected Web services*, as shown in [Figure 211](#), click the **Add** button to specify the Web services that you want to include. Click the **Remove** button if you want to exclude a Web service from the list.
4. In the *Output folder* field, enter the full pathname of the output folder for the generated WSDL files, or use the file chooser.
5. Click **Ok**.

For each Web service that you choose, Together generates a *.wsdl file in the output folder.

[Figure 211](#) shows the Generate WSDL from Web Services dialog with two selected Web services. Together generates two WSDL files, `MySimpleJavaService.wsdl` and `AnotherService.WSDL`.

Figure 211 Generating WSDL from Web services



To generate a WSDL file for a single Web service directly from the diagram:

6. Right-click on the Web service and choose **Generate WSDL** from the right-click menu.
7. In the resulting dialog box, enter the output folder for the WSDL file.
8. Click **Ok** to complete the process.

Generating Web service clients

You can generate a proxy Web service client from a WSDL file. Together also has Web service client patterns for creating clients for Apache SOAP or BEA WebLogic.

Note You can generate a Web service client from an external WSDL using Together's UDDI Browser. For detailed information, refer to [“Generating a WSDL client” on page 730](#).

Generating a proxy Web service client from a WSDL file

When you generate a proxy Web service client from an existing WSDL file, you can opt to generate the type serializers and deserializers as well.

To generate a proxy Web service client:

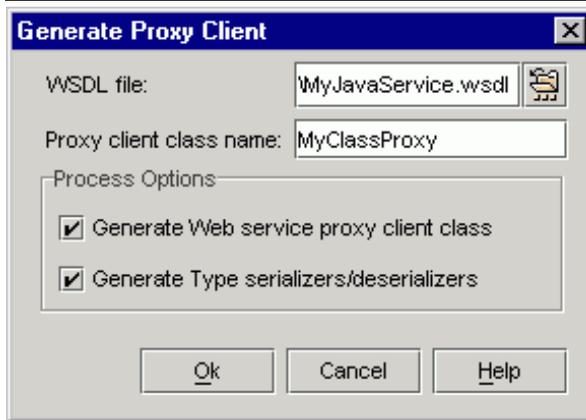
1. On the main menu, choose **Tools | Web Services | Web Service Client Generation**. A dialog opens, such as the one shown in [Figure 212](#).
2. Enter the name of the WSDL file or pick one using the file chooser button.

3. Enter the *Proxy client class name*.
4. In the Process options, check *Generate Web service proxy class*. If the WSDL has a complex type, check *Generate Type serializers/deserializers* if you want Together to generate classes for serializers and deserializers as well.
5. Click **Ok**.

The result is a proxy client class with a method to invoke the accessible Web services methods.

Note If the message “Would you like to add Apache-SOAP libraries to the project?” displays when you complete the process, click **Ok**. Otherwise, you may need to add `soap.jar` to the project Search/Classpath. `soap.jar` resides in the `TGH/lib` directory.

Figure 212 Generating a proxy Web service client



Using patterns to generate proxy Web service clients

Together has two patterns for proxy Web service clients. One creates an Apache SOAP client; the other creates a WebLogic client.

Apache SOAP

The Apache SOAP client is appropriate for both Apache SOAP and IBM WebSphere.

To generate a proxy Apache SOAP Web service client using a pattern:

1. Open the class diagram where you want to place the client.
2. Click the **Class by Pattern** button on the diagram toolbar, and click the diagram. This opens the pattern selection dialog.
3. On the left, expand *WebServices - Clients*. Choose *Apache SOAP Client*.
4. Set the properties for the client on the right:

- **Proxy class name.** Name of the proxy class.
- **Web service class.** Name of the Web service for which you are going to create a proxy client.
- **By Web service class from diagram.** Indicates if you are using an existing Web service on the current class diagram.
 Leaving this box unchecked generates an empty proxy client without any generative Web service class. The code contains commented lines with a usage example.
- **Web service URL.** Location of this Web service.
- **Web service ID.** A unique number that identifies this Web service among all Web services that have the same URL.

5. Click **Finish** to close the dialog and create the client.

BEA WebLogic

With WebLogic, you can generate an RPC client (a client that accesses a Web service through its remote interface) or a Dynamic RPC client (a client that accesses a Web service directly).

To generate a Web service client for BEA WebLogic using patterns:

1. Open the class diagram where you want to place the client.
2. Click the **Class by Pattern** button on the diagram toolbar then click the diagram to open the pattern dialog.
3. In the left section of the dialog, expand *WebServices - Clients - WebLogic*. Choose *RPC Client* or *Dynamic RPC Client*.
4. Set the pattern properties on the right:
 - **Name.** Name of the client class to generate.
 - **Initial Context Factory.** Initial Context Factory for invoking the Web service.
 - **Bean JNDI name.** JNDI name of the bean that implements the Web service.
 - **Remote interface of Bean.** Name of the remote interface of the bean implementing the Web service.
 - **Context.** Context of the Web service that can be defined during deployment using the Web Services Deployment Expert.
 - **Host name.** Name of the server where the Web service is deployed.
 - **Port.** The binding HTTP port of the server where the Web service is deployed.
 - **Invoke procedures.** Check if you want code in the client to invoke each Web service method that is in the bean.

5. Click `Finish`.

The Web service client displays on the current diagram. You can view or modify its code in the Editor pane.

Deploying a Web Service

This chapter contains the instructions for deploying Web services to three different application servers. The chapter includes the following topics:

- “Setting deployment properties” on page 719
- “Running the Web Services Deployment Expert” on page 720
- “Deploying to Apache-SOAP” on page 720
- “Deploying to BEA WebLogic 6.1, 7.0” on page 722
- “Web Services Deployment Expert for IBM WebSphere Application Server 4.0” on page 723

Setting deployment properties

The project options determine the application server and default parameters for the namespaces. The application server must know the namespace parameters when you deploy the Web service.

To change the default application server and the default namespaces:

1. From the main menu, choose **Tools | Options | <level>**.
2. In the resulting dialog box, click **Web Services** in the left panel.
3. Choose the target server from the **Application Server** list on the right panel.
4. On the left panel, expand **WebServices** and click your application server.
5. Enter the namespace properties as desired in the right panel.
6. Click **Ok** to close the dialog and complete the process.

Running the Web Services Deployment Expert

The Web Services Deployment Expert in Together is similar to the J2EE Deployment Expert (see [Chapter 40, “J2EE Deployment”](#)). It has fields for the target server platform, deployment-related actions such as compiling, paths to the server, server tools, and deployment output, and server connection parameters.

The Web Services Deployment Expert varies according to the target application server. Before you use the Web Services Deployment Expert, the server for the deployment of Web services should be installed on your computing system.

To run the Web Services Deployment Expert:

1. Open the project and the class diagram that contains the Web services to be deployed.
2. From the main menu, choose **Deploy | Web Services Deployment Expert**.
3. Choose the target server platform and set the other options as desired. Click **Next**.

Note Together does not allow you to continue to a successive page if it detects errors on the current page.

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. Click **Next** to continue.
5. View default settings for registering Web services during runtime for the current application server. Change them if necessary.
6. Click **Finish**.

Depending on the deployment options, Together interacts with the compiler to compile classes, generates the XML deployment descriptors, and registers the Web services on the application server.

Note For information on project encoding how it impacts deployment descriptors for Web services, see [“Encoding in deployment descriptors” on page 700](#).

Deploying to Apache-SOAP

Tomcat, which comes bundled with Together, is pre-configured to support Apache SOAP. You should start Tomcat before attempting to deploy to Apache-SOAP.

Note If you are going to create a client or a service for Tomcat, you should add the library `$TGHS$/lib` to the project path to include `soap.jar` and `xerces.jar`. If you want to use another version of Tomcat instead of the bundled version, you need to add Apache SOAP to Tomcat manually.

The first step in using the Web Services Deployment Expert is to specify the application server. Choose **Apache-SOAP** from the dropdown list. The expert consists of three pages, which are described below.

Process options for Apache-SOAP

Process options are checkboxes are on the first page of the deployment expert for compiling options, deployment descriptor generation options, and deployment options:

- **Add libraries required for deployment to the current project's Search/Classpath** adds libraries containing `soap.jar` and `xerces.jar` if necessary.
- **Compile classes from the currently selected diagram** forces a new compilation before attempting to deploy. Only compiled classes can deploy.
- **Generate Deployment Descriptor(s)** generates the deployment descriptors.
- **Open XML editor for the generated Deployment Descriptor(s)** opens the XML editor and loads the deployment descriptors as part of the deployment process.
- **Register Web Services on the server** is the actual deployment.
- **Clear temporary folder before/after deployment** options are for file system maintenance. Together creates a temporary folder named `TogetherDeploy` for files it generates during the deployment process. You can opt to remove all the files from `TogetherDeploy`.

Common Properties for Apache-SOAP

Common Properties indicate where to find the necessary files on your system.

- **Temporary folder** is the existing folder containing `TogetherDeploy`.
- **JDK 1.3 root directory** and **Apache-SOAP home directory** have built-in default values. You cannot proceed to the next page of the expert unless these values are correct.

Note If you are using Tomcat bundled with Together, the Apache-SOAP home directory is `$(TGHS)/lib`.

Run-time Registering Web Services for Apache-SOAP

Run-time Registering Web Services define server properties.

- **Server host name** is `localhost` by default. The Web Services Deployment Expert uses the file system access to the application server with Apache SOAP, so the server runs locally.
- **Server port number** has a default value of 8080.

Deploying to BEA WebLogic 6.1, 7.0

Before actually attempting to deploy, you must start the BEA WebLogic server. You can optionally start the server within Together. Instructions are in [“Together provides the following plug-ins for starting application servers:”](#) on page 672.

The first step in using the Web Services Deployment Expert is to specify the application server. Choose **BEA WebLogic 6.1/7.0** from the dropdown list. The expert consists of three pages, which are described in this section.

Process options for BEA WebLogic 6.1, 7.0

Process options are checkboxes for compiling options, deployment descriptor generation options, and deployment options.

- **Compile classes from the currently selected diagram** compiles the classes from the selected diagram (using `javac.exe`), packs them in a JAR file using (`jar.exe`), and stores them in a temporary folder.
- **Generate EJB Deployment Descriptor** creates a deployment descriptor for all RPC Web Services and the `build.xml` file.
- **Open XML editor for the generated Deployment Descriptor** opens the XML editor and loads in the deployment descriptor and `build.xml`.
- **Pack modules for deployment** creates a deployable `*.jar` archive (standard JAR for deployment of an RPC Web service session EJB)
- **Assemble Web Services** generates a `build.xml` file and generate a deployable `*.ear` archive using Java Ant.
- **Register Web Services on the server** deploys the generated `*.ear` by registering the RPC-style or message-style Web Services on the target application server.
- **Clear temporary folder before/after deployment** options are for file system maintenance. Together creates a temporary folder named `TogetherDeploy` for files it generates during the deployment process. You can opt to remove all the files from `TogetherDeploy`.

Common Properties for BEA WebLogic 6.1, 7.0

Common Properties tell where to find the necessary files on your system.

Temporary folder and **JDK 1.3 directory** are identical to their counterparts in the earlier section, [“Common Properties for Apache-SOAP.”](#)

- **WebLogic Server 6.1/7.0 root directory** is the path to the directory that includes the BEA WebLogic startup file and config files for your applications.
- **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.

Run-time Registering Web Services for BEA WebLogic 6.1, 7.0

Run-time Registering Web Services options define the server access properties.

- **Target server name** is the name of the server that you want to use for your applications.
- **Server host name** is 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 6.1/7.0 application server. Therefore the server runs on the same computer.
- **System password** is your password for accessing the server.
- **Server port number** is the port number of BEA WebLogic Server 6.1/7.0, with the default value 7001.
- **Protocol** is the protocol that clients use to access the Web service. There are two possible values: http or https. The default value is http.
- **Update previously deployed module** determines whether this module replaces one that is already deployed.

Web Services Deployment Expert for IBM WebSphere Application Server 4.0

Before actually attempting to deploy, you must start the IBM WebSphere AES 4.0.

The first step in using the Web Services Deployment Expert is to specify the **Application Server**. Choose **IBM WebSphere 4.0** from the dropdown list.

The Web Services Deployment Expert consists of three pages of options, which are described in the rest of this section.

Process options for IBM WebSphere 4.0

This page is almost identical to the one for Apache SOAP, which are described in [“Process options for Apache-SOAP” on page 721](#). The only differences are the selected application server at the top of the page and an option to **Enable Single Edition Support**.

Common Properties for IBM WebSphere 4.0

Common Properties tell where to find the necessary files on your system. They include:

- **Temporary folder** is the folder containing the TogetherDeploy folder.

- **WebSphere server home directory** is the location of IBM WebSphere application server.
- **Folder for generated packed files** by default is the same as the Temporary folder.
- **JDK 1.2 root directory** is in the WebSphere directory by default.

Deployment Properties for IBM WebSphere 4.0

Deployment Properties include server activities and parameters.

- **Uninstall previous application, Install new application, and Restart IBM WebSphere Application Server** are checkboxes for specifying server activities.
- **Node name** is the node name in the WebSphere console.
- **Server name** is the name of your server in the WebSphere console.
- **Server configuration file** specifies the path to the configuration file `server-cfg.xml`.

Chapter 43

UDDI Browser

Universal Description, Discovery, and Integration (UDDI) registries are databases of business services that are maintained by groups of companies called operators. Together has a UDDI browser for working with UDDI registries located on operator sites on the Internet.

This chapter includes the following topics:

- “UDDI Browser overview” on page 43
- “Searching for Web objects” on page 43
- “Publishing Web services” on page 43
- “Using templates” on page 43

UDDI Browser overview

You can use the UDDI browser in Together for multiple purposes: to access the UDDI registries, to discover available Web services, to execute UDDI requests, to call Web services, and to register new Web services. You can also download the WSDL (Web Services Description Language) file for a Web service if the file is available.

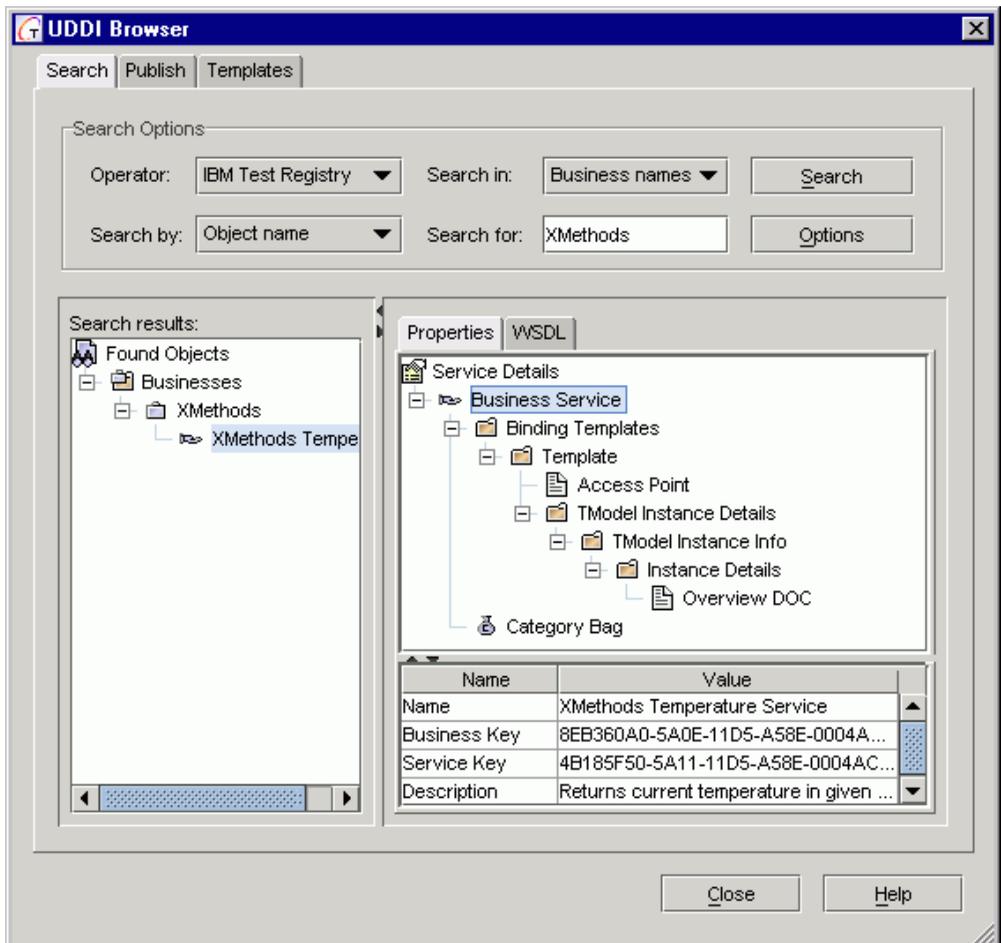
The UDDI browser is a feature that does not require an open Together project. To open the browser, choose **Tools | Web Services | UDDI Browser** on the main menu. The UDDI browser is divided into three major tabs: Search, Publish, and Templates.

Searching for Web objects

The Search tab displays a UDDI registry and the names of registered Web objects that it finds through some search criteria for Web businesses or Web services.

The Search tab is divided into three sections, as shown in [Figure 213](#). Search options are across the top. Search results are on the left, in a tree of Web services and businesses. Properties of the selected object are on the right.

Figure 213 Search tab of the UDDI browser



Search options

Search Option settings determine how the UDDI browser searches for Web objects. There are four settings:

- **Operator.** An operator site with an installed UDDI registry. The default site is *IBM Test Registry*. The dropdown list has additional operators. You can add a different Internet address to the list of operators. Click the **Options** button to see the properties of the current operator or to add a new one. (See “[UDDI browser options](#)” on page 43.)
- **Search by.** Switches between different search methods. There are five choices:
 - **Object name.** Search by a simple object name.
 - **Category bag.** Search by categories of objects.
 - **Identifier bag.** Search the unique identifier in the registry.
 - **TModel bag.** Search by service type.
 - **Discovery URLs.** Search by discovery URL, which is a link to additional information about the Web service provider.
- **Search in.** The type of UDDI object to search for. There are three choices:
 - **Business names.** Businesses, which contain lists of Web services. Businesses support search by object name, category bag, identifier bag, tModel bag, and discovery URLs.
 - **Service names.** Names of Web services. Services support search by object name, category bag, or tModel bags.
 - **TModel names.** TModels, which are service types. TModels support search by object name, category bag, or identifier bags.
- **Search for.** The name of the object or context information to search for. You need to enter only the first part of a name to restrict the search.

After you set the search options, click **Search** to start the search operation. The results display in a tree on the left side of the browser.

UDDI browser options

On the **Search** tab of the UDDI browser, click the **Options** button to view and change the settings for the operator and the search. The resulting dialog box displays operator configuration options on the top and search options on the bottom.

Operator configuration options

The operator configuration option shows the list of available operators on the left. You can change that list by using the **Add** and **Remove** buttons. After you add a new operator, use the fields on the right to set its values. There are three *Inquiry Options* for setting the properties of the selected operator:

- **Operator Site Title.** Title for the operator that is displayed in the Operator drop-down list on the *Search* tab.

- **Inquiry Operator Site URL.** Inquiry registry entry point URL. You can enter a URL for an Internet site with an installed UDDI registry from the default list of site addresses, or you can add a new Web address.
- **Enable Publish Options.** Indicates whether this operator has a registry that enables publish operations. If this is unchecked, the *Publish Options* group shown below is disabled and the operator is not displayed in the list of operators on the *Publish* tab. However, the operator is still on the *Search* tab.

There are three *Publish Options* for operators with registries enabled for publishing:

- **Publish Operator Site URL.** Sets the publish registry entry point URL.
- **User Name.** User name for authorization on the site.
- **Password.** Password for the specified user.

Note You must register on one of the publish operator sites in order to get a user name and password. To register on the IBM Test Registry, go to <https://www-3.ibm.com/services/uddi/testregistry/protect/registry.html>.

Search Options

Search options are filters for searching. There are five options:

- **Exact Match.** Requires that the string entered in the *Search for* field is the exact name of the object to search for. If this box is unchecked, searching retrieves all objects whose names start with this string.
- **Case Sensitive.** Searches according to characters and case in the *Search for* string.
- **Sort By Name Descending.** Sorts the results by name in descending order. Names are sorted in ascending order by default.
- **Sort By Date Ascending.** Sorts the results by date from least recent to most recent.
- **Maximum Occurrences.** Maximum number of objects to be displayed in the search results. Searching stops after this limit is reached.

Search results

When you initiate a search, the UDDI browser gives the search results as an expandable tree on the left side of the Search tab window. Clicking on the icon or the name of a node in the tree selects it. [Table 78](#) lists the icons for the nodes in the tree.

Table 78 Types of results from a UDDI search

Icon	Description
	Web businesses folder. Expand to reveal the business entities inside.

Table 78 Types of results from a UDDI search

	Business entity. Expand to reveal the business services inside.
	Business services folder. Expand to reveal the business services inside.
	Business service
	TModels folder. Expand to reveal the tModels inside.
	TModel

Properties display

Choosing an object from the Search results sends a request to the UDDI registry to return information about the object. The Properties tab on the right displays the results.

You can expand the containing items such as templates and category bags in the Properties tab to reveal their contents. The bottom part of the Properties tab shows the detailed information on the selected item in the top. The information varies with the type of object, and it includes such things as business key, service key, and description.

There are three kinds of objects that a search can reveal (according to the *Search in* option):

- **Business names.** Web businesses with sublists of business services for each business. When you select a specific business from the search results tree, the Properties tab displays the detailed properties for this business, such as the owner name, telephone numbers, and email addresses.

Web businesses do not have WSDL files.

- **Service names.** Web services. When you select a service in the search results tree, the Properties tab displays the associated templates, category bags, tModel instances, and documents. The information on a Web service is the same as when you search in business names and then select a specific business service in the results.

A Web service can have a corresponding WSDL file. You can view its contents by going to the WSDL tab and choosing its address from the drop-down list.

- **TMethod names.** TModel bags. When you select a tModel bag in the search results tree, the Properties tab displays the associated category bag, overview, and identifier bag.

WSDL file display

If the selected object in the search results tree has a corresponding WSDL file, the WSDL tab on the right displays the WSDL file organization and data. [Figure 214](#) shows how the UDDI browser displays a WSDL file. Each method is in its own folder, as are the data for the other tags such as binding and port.

If there is no corresponding WSDL file, the tab displays the message No WSDL.

Generating a WSDL client

You can generate proxy clients for WSDL files in the UDDI browser.

To generate a WSDL client:

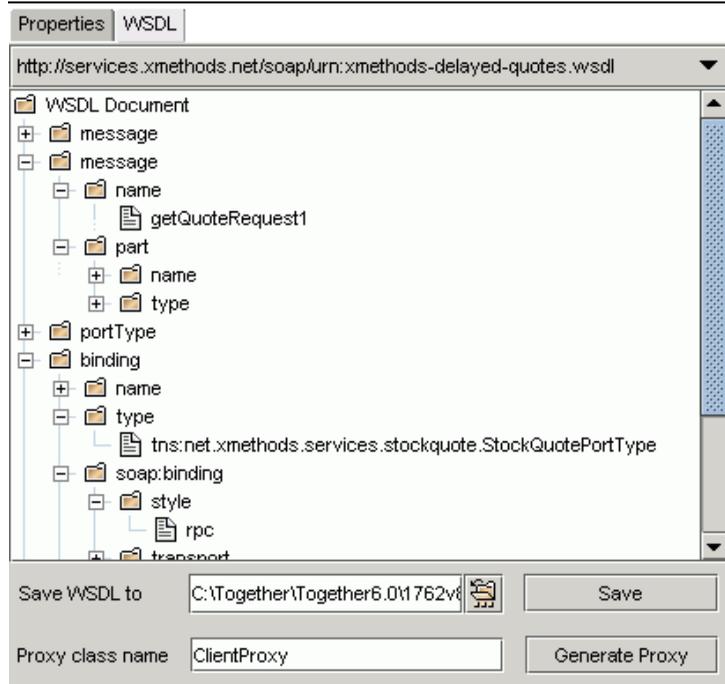
1. Open a project and open a class diagram in the project.
2. Search for the desired Web object in the UDDI browser.
3. Choose the Web object among the *Search results* and click the *WSDL* tab.
4. Enter a name for the client in the *Proxy class name* text field.
5. Click **Generate Proxy**.

This generates a client file in the project and places a shortcut to the client on the active class diagram.

Saving WSDL files

Fields and buttons for saving the WSDL file are at the bottom of the Search tab. To save the WSDL file to the hard disk, use the file chooser to select the necessary path and click Save.

Figure 214 Contents of WSDL tab from a UDDI browser search



Publishing Web services

The Publish tab of the UDDI browser is for working with publish registries to create and manage UDDI businesses, service types, and services. There are four buttons at the top of the tab:

- **Operator.** Displays the publish-enabled operators.
- **Options.** Opens UDDI browser options. (See [“UDDI browser options” on page 43.](#))
- **Log on.** Launches the logon process for the publish operator registry.
- **Status.** Displays the current connection status and the number of objects registered on the publish operator.
- **Refresh.** Updates the list of registered objects. Refresh is enabled only when you are logged on.

The remainder of the Publish tab consists of two sections. The Registered Objects section displays a tree of all objects that you have registered on the operator site. When you choose an object from the tree, the Properties section on the right displays the object as a tree, with detailed information about the selected properties tree node at the bottom.

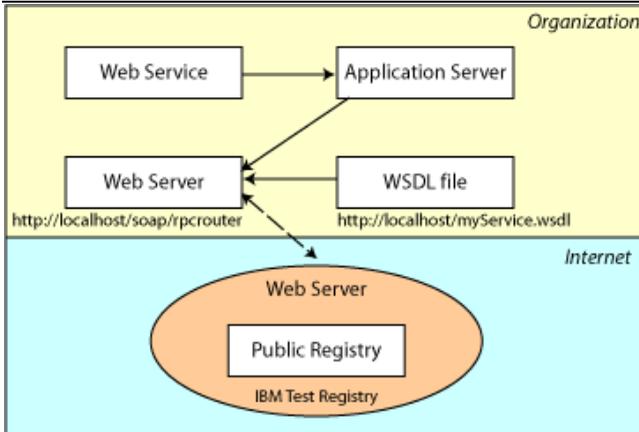
Publishing prerequisites

In order to publish on a site, you must be a registered user with a user name and password. You can define your user name and password when you register at the operator site.

A Web service must be visible from the Internet to be accessible to other users. This means the Web service must be on the list of published Web services in the public registry on an operator’s site, which is located on another Web server.

[Figure 215](#) illustrates the relationship of a Web service to the Internet. Notice that an application server is connected to a local Web server. You can call a Web service using the Web server and HTTP protocol. The exact address is set during deployment.

Figure 215 The relationship between Web services and the Internet



Logging on to the operator site

You must log on to the publish operator site before attempting to publish.

To log on to the publish operator site:

1. In the UDDI Browser, choose the *Publish* tab.
2. If you have not yet added your operator site to the list of operators in the UDDI browser:
 1. Click the **Options** button.
 2. Click **Add** to add a new operator.
 3. Enter the information for all of the publish operator fields, including a name, site URL, and your user information. Then click **Ok**.
3. From the **Operator** dropdown list, choose the publish operator registry.
4. Click **Log On**.

Logging on sends a query to the publish operator registry for the current user, who is defined according to the user name and password specified in the UDDI browser options. A message displays if the logon is unsuccessful.

Publishing a Web service on an operator site

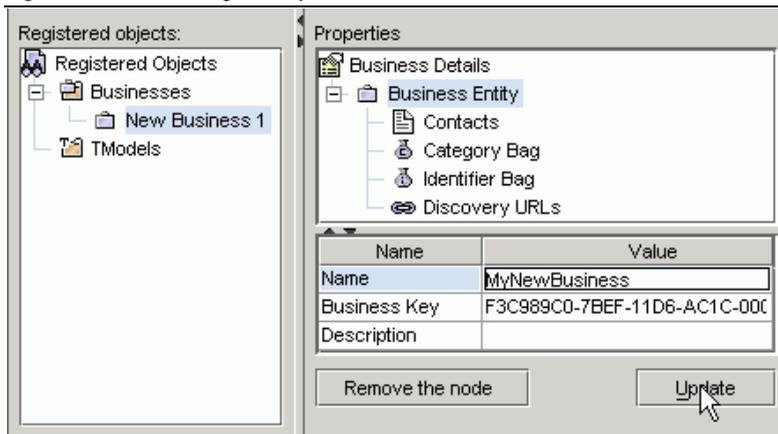
You can publish any Web services that you have already deployed. If your Web service has a corresponding WSDL file, you should use your Web server tools to place the WSDL file on the same Web server.

Published Web services belong to registered businesses. If your business is not already registered, you must register it before attempting to publish a Web service.

To register a business on the operator site:

1. Log on the publish operator site for which you are a registered user.
2. Click the **Publish** tab on the UDDI browser and choose the **Operator** from the dropdown list at the top.
3. In the Registered Objects treeview, right click **Businesses** and choose **New Business** from the menu. This creates a new business named *New Business 1*.
4. Expand the **Businesses** node and choose **New Business 1**.
5. Edit the *Name* at the bottom of the Properties on the right. [Figure 216](#) shows renaming *New Business 1* to *MyNewBusiness*.
6. Click **Update** to save the information on the operator site.

Figure 216 Renaming a newly created business



To publish a Web service:

1. Log on to the publish operator site, click the **Publish** tab on the UDDI browser, and choose the **Operator** where you want to publish.
2. In the Registered Objects treeview, expand **Businesses** and right click the business where you want to offer the service. Choose **New Service** on the right-click menu. This creates a new Web service named *NewService1* for the registered business. You may need to expand the business to see *New Service 1*.
3. In Registered Objects treeview, click **New Service 1**. If desired, rename the service in the Properties listed at the right.
4. In Properties, navigate to Business Service: Binding Templates. Right-click on **Binding Templates** and choose **Add new 'Template' node** from the right-click menu. A new expandable Template node displays in the business service tree.
5. Click the **Update** button.
6. In Properties, navigate to **Business Service - Binding Templates - Template: Access Point**. Enter the *URL Type* and the *Access Point* address below.

Note If you intend to publish the Web service on multiple sites, you can create additional templates using the same process as described in the steps above.

7. If you want to include a WSDL file for your Web service:
 1. Navigate to **Business Service - Binding Templates - Template: TModel Instance Details**. Right click and choose **Add new 'TModel Instance Info' node**.
 2. Navigate to **TModel Instance Details - TModel Instance Info - Instance Details: Overview DOC**. Enter the *Overview URL* and *Description* at the bottom.
8. Click **Update** to publish the Web service on the operator site.

Note You can add to an enumeration type (category bag, identifier bags, tModel bag, or discovery URL) by right clicking the enumeration type node in the Properties pane and choosing *Add New 'Keyed Reference'* node.

Managing registered objects

The registered objects in the Publish tab have right-click menus that enable you to remove them or to use them to create new objects.

The right-click menus of Businesses and TModels have New Business and New TModel commands respectively. The right-click menu of a business has a New Service command.

To remove a published object (a business, service, or tModel), right click its node and choose **Remove <object type>**. This is very useful when the number of services in the corresponding register is limited.

Note You cannot remove service types from the list. When you “remove” them, they simply become inaccessible for searching.

Using templates

The Templates tab displays a collection of templates for businesses, services, tModels, category bags, identifier bags, tModel bags, and discovery URLs. The tab has two sections. The Templates section on the left lists the available templates sorted according to type. When you select a template, the Properties section on the right displays its properties.

You can create new templates for each of these types by copying existing objects or by building one in place. You can apply a template to an existing or new object by copying it and pasting.

Note You can paste an enumeration type template to add to the contents of any existing enumeration of the same type. In order to apply a complex type template, you must replace the original complex type object.

To build a new template:

1. Click the **Templates** tab.
2. In the Template folders on the left, select a template from the list, and right click the folder for the kind of template you want to build. Choose **New <kind>** from the right-click menu. A new template for that kind displays under the folder.
3. Choose the new template from the tree on the left in order to modify the template properties on the right. To add a new element to the template, right click the desired kind of node and choose **Add new <kind> node**.
4. Each time you add a new element to the template, click **Update**.

To apply an existing template:

1. Click the **Templates** tab.
2. Choose the template that you want to copy from the Templates on the left.
3. In the Properties display, right click the node (or the part of the node) you want to apply, and choose **Copy Node** from the right-click menu.

You can use the copied template to change or embellish an existing template or a published Web service.

To change a published Web service, choose the **Publish** tab.

4. Choose the node that you want to apply the template to so that it displays in the Properties pane. The node must be the same kind as you copied in the previous step.
5. Right click the node in the Properties display. Choose one of these commands from the right-click menu:
 - **Replace node:** Replaces the node with the copied node.
 - **Paste node:** Adds the copied node to the contents of the current one. This is available only for enumeration types (category bags, identifier bags, and discovery URLs).
6. Click **Update**.

EXTENDING TOGETHER

- Chapter 44, “Advanced Customizations”
- Chapter 45, “Together Open API”

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 739](#)
- [“Creating custom diagram types” on page 740](#)
- [“Customizing Property Inspectors” on page 747](#)
- [“Customizing View Management Show options” on page 752](#)

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 of the Together installation 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 a backup copy of the original files.

Creating custom diagram types

Together supports the currently-defined UML diagram types, and 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 79](#).

Table 79 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.

Note `element_unique_name` may not contain spaces. Use underscores instead of spaces.

Example:

```
resource.element.sdEntity.icon.small = "Images/tv-deepPit.gif"  
resource.element.sdAttribute.icon.small = "Images/tv-honey.gif"
```

Step 4: Defining toolbar icons

Button definitions dynamically construct Toolbar icons. Each toolbar button represents one element, and every element in a diagram is handled as either a node or a link. Any element drawn as a type of line or arrow between nodes defines a link. In some situations, there are important differences between nodes and links. However, in this task, the name variable to store the new element represents the only difference between handling buttons for a node or a link. Accordingly, the name of the variable to store the new element must be either `node` or `link`, and the internal function used to insert the element into the new diagram must be either `createNode` or `createLink`.

The example that follows demonstrates adding a node element to the diagram. 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}.condit
ion = "{enabling_condition}"

diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name}.icon =
"{path_to_icon}"
```

where:

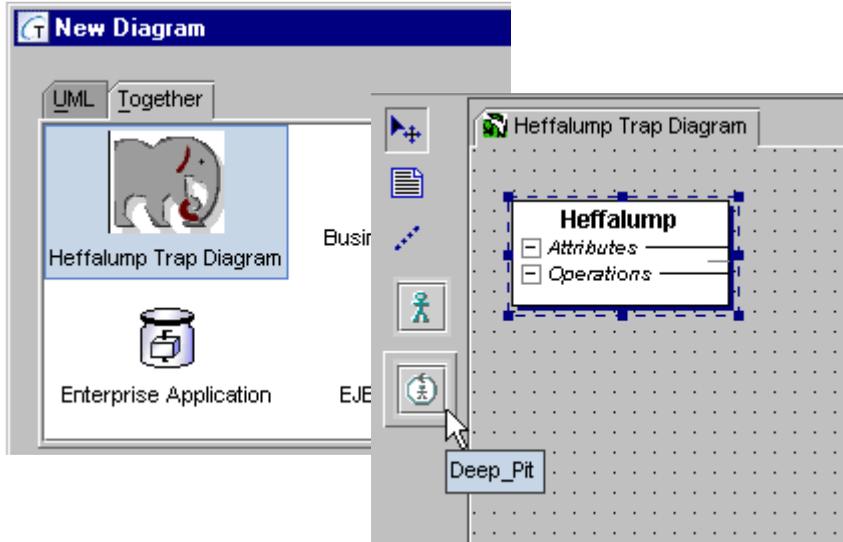
- `element_toolbar_name` is a unique name of a toolbar button.
- `path_to_icon` is the path to the button's icon.
- `node` is a variable that stores the new element.
- `createNode` is an internal function that creates a new node (entity).
- `enabling_condition` is any valid boolean expression that defines the availability of the button.
- `diagram.toolbar.button` is a predefined construction to define a new button on the toolbar.

Tip To create a link element, change `node` to `link`, and change `createNode` to `createLink`.

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 217 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

The diagram manages the nodes in terms of the boundaries each node fits inside of, not the drawn shape inside of those boundaries.

Define the shape of the new element:

```
\ setGraphicObject("{shape_name}");
\ setLayoutConstraints( {parameter}, [parameter], ... );
where:
```

- `setGraphicObject` is a function that sets the `shape_name` of an element. The predefined shapes are `Cube`, `RectangleVisible`, `Circle`, `RoundRectangle`, `Oval`, `Folder`, `$Note`, `Actor`, `BeanComponent`, `RectangleObject`, `$RoundRectangle`, `$GraphSequenceInteraction`, `RectangleObjectBordered`, `MultipleInstance`, `Component`, `Folder`, `Oval`, `DeepHistory`, `History`, `ArcRectangle`, `Rhomb`, `ConcavePentagon`, and `ConvexPentagon`.

Tip To see examples of the `shape_names` and how they are used, see `TGH\config\view.config` file.

- `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));
```

The above example draws the node inside of a cubical boundary.

Defining the element's options

This section defines 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()
```

Tip To see other functions and options that are available for config files, see `TGH\doc\guides\config\index.html`. Choose *Viewmap queries*.

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 the Together Explorer. 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.

Tip API-based customizations are developed using Java. If you are a C++ developer, and unaccustomed to developing in Java, there are tutorials located at the following: <http://java.sun.com/docs/books/tutorial>

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`.

Note Before proceeding with the steps below, create a backup copy of the class `AddPageToInspector.java` (or `AddFieldsToInspector.java`) located in the `TGH\modules\com\togethersoftware\modules\inspector\examples` directory.

To add new pages and fields to the inspector:

1. Open the appropriate manifest file (`*.def`) in `TGH\modules\com\togethersoftware\modules\inspector\examples`. For this example, open `Page.def`.
2. Uncomment these lines:

```
MainClassName =
    com.togethersoftware.modules.inspector.examples.AddPageToInspector
MainClassName
Time = Startup
```
3. Create a Together project with the desired class: `AddPageToInspector.java` (or `AddFieldsToInspector.java`).
4. Add the following archive files to the Search/Classpath tab in the Project Properties dialog box:
 - `TGH\lib\openapi.jar`
 - `TGH\lib\together.jar`
 - `TGH\modules\com\togethersoftware\modules\inspector\inspector.jar`
5. Create a backup copy of the class `AddPageToInspector.java` (or `AddFieldsToInspector.java`) and edit the source code, specifying the desired page and field names as shown below:

```
...
IdeInspectorProperty property;
```

```

// Replace "myStringProperty" with the custom page name
// This creates a string property
property = new RwiInspectorStringProperty(rwiElements, " taskDescription
");

// Specify custom name to be shown in the UI
// Replace "My string property" with the name to be shown in the UI
property.setName(" Task manager ");

// adds it to the page
// default editor - just a text field - is used
page.addProperty(property, null);
// Replace "myBooleanProperty" with the custom name
property = new RwiInspectorBooleanProperty(rwiElements, " isReady ");

// Specify custom name to be shown in the UI
// Change "My boolean property" to the name to be shown in the UI
property.setName(" Is the task completed ");

// adds it to the page
// shown as a checkbox in the UI
page.addProperty(property, null);

// Creates a string property
// Replace "myChoiceProperty" with the custom string property name
property = new RwiInspectorStringProperty(rwiElements, " rate ");

// Specify custom name to be shown in the UI
// Replace "My choice property" with the name to be shown in the UI
property.setName(" Select rate from the list ");

// Creates and sets an editor for it - combobox(choice)
// Specify custom names for the drop-down list

```

```
// Replace "value1", "value2", "value3" with "rate1", "rate2", "rate3"
SwingComboBoxEditor editor = new SwingComboBoxEditor(
    new DefaultComboBoxModel(new String[] {"rate1", "rate2", "rate3"}));
...

```

6. Reassign the compiler destination folder to the same location as the source file. From the Together main menu, choose **Tools | Options | Default Level or Project Level**. Expand the **Builder** and **Built-in Javac** nodes on the left of the dialog, and choose **Compiler options**. On the right side of the dialog choose **Destination directory**.
7. Compile the class.
8. 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. The open API provides some technical documentation of the key classes and interfaces, and a commented example of how to perform a simple Inspector customization. Refer to the documentation for `com.togethersoft.openapi.ide.inspector` through `$TGHS/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** from 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 **Ctrl+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. From the main menu in Together, choose **Tools | Options | Default Level or Project Level**. Choose **General** on the left of the dialog, and check **Support user-defined inspector** on the right of the dialog.
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 Online 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 `filter.config` file:

```
optionsEditor.item.View Management.item.Filters.item.A.item.shortName.name = ["filters/filter_shortName"]
```

The text in brackets 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 `TGH/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 section sequence of the file.

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 string literal instead:

```
filter.m.name = "My New Filter Options"
```

Finally, you can update the other lines with references to resources or literals as required.

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 755
- “Understanding modules” on page 757
- “Extending Together using modules” on page 759
- “Guidelines for developing modules” on page 761
- “Tutorial: Writing and deploying modules” on page 764
- “Accessing more API documentation” on page 774

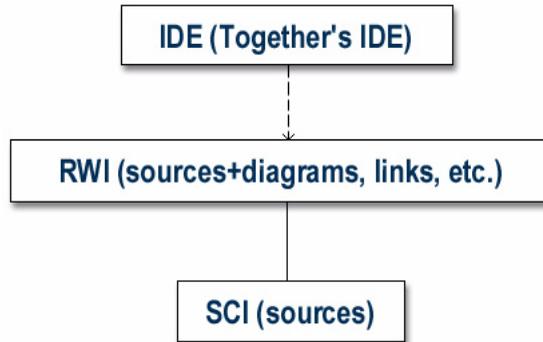
Overview of the Together Open API

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 218](#). The top tier represents the highest degree of constraint and the bottom tier the lowest degree of constraint.

Figure 218 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 755](#)).

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.togethersoft.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.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.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 `$(TGH$)/modules/com/togethersoftware/modules/inserttags/InsertTags.def`. For more information regarding `.def` files, see [“Declaring a module in a .def file” on page 769](#)

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 761](#). For information about other module types, see [“Types of modules” on page 760](#).

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, create the file `TGH/modules/com/togethersoft/modules/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, create the file `TGH/modules/com/togethersoft/modules/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:** Compiled modules are 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 757](#).

Modules of the User type implement the `IdeScript` interface that provides the `run()` method. 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 80 shows how Together represents modules in the Modules tab.

Table 80 Modules Tab

Icon	Description
	Java source file for a module. If a compiler is configured, the source file can be compiled by choosing Run from the right-click menu.
	Compiled Java module.
	Tcl script. The Run 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 `TGH/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 `TGH/config/menu.config` and or `TGH/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 81](#) provides guidelines for

Table 81 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.	<ul style="list-style-type: none"> • <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”	<ul style="list-style-type: none"> • <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.
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.	<ul style="list-style-type: none"> • <code>setFirstName(String aName)</code> • <code>setAccountNumber(int anAccountNumber)</code> • <code>setReasonableGoals(Vector newGoals)</code> • <code>setPersistent(boolean isPersistent)</code> • <code>setAtEnd(boolean isAtEnd)</code> 	Indicates that a method sets the value of an attribute of an object.

Table 81 Naming conventions for modules

Item	Convention	Example	Benefit
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.

naming conventions.

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.

To follow this exercise, create the following directory:

```
$TGH$\modules\com\togethersoft\modules\myFirstModule
```

Create the source file listed below, and name the file: `MyFirstModule.java`

```
package com.togethersoft.modules.myFirstModule;  
  
import com.togethersoft.openapi.ide.IdeContext;  
import com.togethersoft.openapi.ide.IdeScript;  
import com.togethersoft.openapi.ide.IdeStartup;  
import com.togethersoft.openapi.ide.message.IdeMessageManagerAccess;  
import com.togethersoft.openapi.ide.message.IdeMessageType;  
  
public class MyFirstModule implements IdeScript, IdeStartup {  
    public void run(IdeContext context) {  
        //Open the message pane  
        IdeMessageManagerAccess.getMessageManager().setPaneVisible(true);  
        //Regular modules perform this method.  
        IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMATION,  
            "This is my first module called as a regular Together module.");  
    }  
  
    //Startup modules perform this method.
```

```

public void autorun() {
    //This will appear in the Message pane when Together is loaded.
    IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMATION,
        "This is my first module. It was called as a startup module.");
    //This will appear in the Console window at the startup of Together.
    System.out.println("This is my first module, called a startup
        module.");
}
}

```

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 769](#).

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
```

An Example of the analogous 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.

Example:

```
Main-Class: com.togethersoft.modules.ejb.EJB
```

```
Name: EJB Support
```

- **Folder:** The `Folder` 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 an 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 these services 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 right-click menu of the module. The value of this property is the name of the properties page, the same as used in the `IdeConfigManager.showConfigEditor` method (for more information, refer to `TGH\doc\api`).

For example:

```
Main-Class: com.togethersoft.modules.test.export
Time: User
Name: Test Export
Options: General
```

In the example above, the General properties page will open when using the Options command from the module's right-click menu. For Activatable modules, this command displays only when the module is activated.

Note Only the default level options can be shown in 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 if you want to be able to run the module 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 in the Modules tab of the Explorer. 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)
```

If a folder is not defined in the `*.def` file, then the module is placed under the `Early Access` folder by default.

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, 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 cannot 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. For more information, refer to the API documentation located in the `TGH\doc\api` directory.

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/scriptloader.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
Folder = "Sample/MyFirstModuleFolder"
Name = "My First Module"
```

Save the file as:

```
$TGH$\modules\com\togethersoft\modules\myFirstModule\MyFirstModule.def
```

Compiling the module

Use Together or your favorite Java compiler to compile the module source code. 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\togethersoft\modules
```

```
$TGH$\classes\com\togethersoft\modules
```

For example, if your manifest file is:

```
$TGH$\modules\com\togethersoft\modules\coolModule\CoolModule.def,
```

then your `*.class` files must be somewhere in the classpath under the directory:
`com\togethersoft\modules\coolModule\.`

Especially for your first modules, keep `*.class` files in the same directory as the `manifest.mf` (or `*.def`) file and the sources. For example, you can keep them together in a separate directory under `TGH\modules\com\togethersoft\modules.`

Evaluating the results

Once you have compiled the module, run Together. If you compiled the module using Together, then you will need to restart Together. If the console window is open, you should see the module's startup message to indicate when it was called at the startup and when it performed the autorun method.

Note 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**. When right clicking the source file, the Run command will cause Together to compile and run the module. 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
Folder = "Sample/MyFirstModuleFolder"
Name = "My First Module"
and
```

```
;Script = true
Startup = true
Folder = "Sample/MyFirstModuleFolder"
Name = "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 the Together 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 of 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/api`.

Other common problems include:

- Your code compiles without errors, but you can not locate your module on the 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 locate your module on the 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`.



Configuring Together for Multiple Users

The information in this chapter is specifically for the site administrator responsible for setting up Together for multiple users.

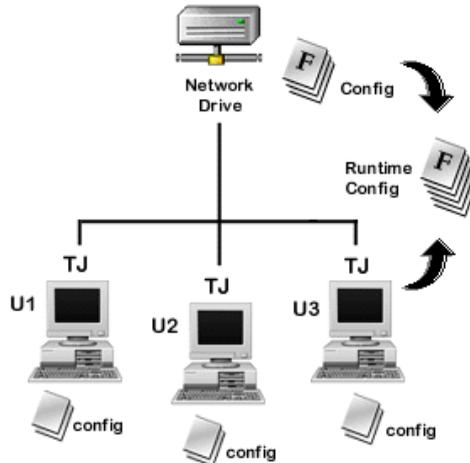
This chapter includes the following topics:

- [“Overview of a shared multi-user configuration” on page 775](#)
- [“Setting-up multiple workstation installations” on page 776](#)
- [“Adding new configuration levels to predefined levels” on page 778](#)

Overview of a shared multi-user configuration

Together uses a tiered, hierarchical approach to configuring properties, meaning that as the Together administrator, you can configure options at three pre-defined levels: Default, Project, and Diagram. This enables you to provide a set of configuration properties (such as code-generation rules) for all users of the installation. You can specify which options users can not override, and which options users can configure with their own settings.

Figure 219 Together running on individual workstations



Setting-up multiple workstation installations

Multiple users with their own copies of Together running on separate workstations can access a shared multi-level configuration. As the Together administrator, you can store configuration properties for the group in a centrally accessible location on the network. Users launch Together from a batch or command file, specifying the path to the central configuration using the `-config.path` command-line switch.

Note In addition to the predefined configuration levels of default, project, and diagram, you can add new configuration levels to Together that users can access. The instructions that follow include a reference to this option.

To set up a multi-workstation shared configuration for a shared installation, complete these steps:

1. Install Together on your (the Together administrator's) workstation. Make a back-up copy of the original `./config` directory of the Together installation.
2. Run Together on your workstation.
3. If additional configuration levels are needed, create these as described in ["Adding new configuration levels to predefined levels"](#) on page 778.
4. Edit configuration options as required by choosing **Tools | Options | applicable_level**, and clicking the **Levels >>** button. Mark individual options as "final" at the desired level for those settings that you do not want individual users to override from a lower level.

See [Chapter 4, "Configuring Options"](#) for detailed instructions for setting-up options.

5. Copy the modified `./config` directory (plus any additional directories created for added config levels) to a shared network location (for example, `<server>/tg_shared_config`) that is accessible to all Together users who use this configuration.

Creating the start-up pointer file

To centrally share the entire configuration, you need to create a pointer file to define the centralized location and override Together's hard-coded configuration locations. Once you create the pointer file and define its location(s), you need to reference the pointer file in the `-config.path` switch when you start Together from the command line (or from a batch or command file). The most convenient way to create the pointer file is to let Together generate a copy of its own defaults.

To generate a default pointer file:

1. Run the Together launcher for your OS and pass:

```
-debug.config.saveDefaultPath=<path to file>  
in the command line. For example:
```

```
Together.exe -debug.config.saveDefaultPath=c:/TogetherSoft/Together6.0/  
lib/pointer.config
```

2. Together starts up and writes the specified file to disk. You can then close Together.

The file name used in the example is just a suggestion. You can use any name that does not conflict with Together property file names. You can store the pointer file anywhere, but it is probably most convenient to keep it in the `./lib` directory of each local installation. Immediately after you generate it, the `pointer.config` file will contain the following lines:

```
config.level.$internal.group = basic  
config.level.$internal.0 = $TOGETHER_LIB$/internal.config  
config.level.$internal.1 = $TOGETHER_LIB$/path.config  
config.level.$default.group = session  
config.level.$default.0 = $TOGETHER_CONFIG$/*.config  
config.level.$default.write = $TOGETHER_CONFIG$/changes.config  
config.level.$commandLine.group = basic  
config.level.$project.group = project  
config.level.$workspace.group = project  
config.level.$workspace.0 = $PROJECT_DIR$/*.config  
config.level.$workspace.write = $PROJECT_DIR$/$PROJECT_NAME$.tw$
```

You need to edit the highlighted (in **bold**) lines to point to your shared configuration. Note that paths must reference mapped drives (for example, this path will not work: `\\appserver\tg_shared_config*.config`). So if your shared configuration is on `appserver` which is locally mapped to drive `s:`, then your edited lines should look like this (Windows style paths shown):

```
config.level.$default.0 = s:\tg_shared_config\*.config
config.level.$default.write = s:\tg_shared_config\changes.config
```

Launching via the command line and pointer file

You should create for each user, or instruct users to create on their workstations, a start-up batch or command file that launches Together locally from the command line using the `-config.path` switch to specify the path to the pointer file. In Windows, the command would look something like this:

```
c:\Together\bin\Together.exe -config.path=c:\Together\lib\pointer.config
```

Note If you are using the Together floating license server, Together may not use the `pointer.config` file to find the `server.config` file. As a workaround, you need to start Together using this command:

```
together.exe -license.config.path=<config_dir> -config.path=$TGH$\lib\
pointer.config
```

Adding new configuration levels to predefined levels

You can add up to three additional configuration levels to the pre-defined levels of Default, Project, and Diagram. New levels must be inserted above the installation-wide Default level in the Options dialog order.

For example, a Corporate level could be added to enforce certain configuration settings across the enterprise. You could define the File prologue option in Source Code options so that all generated source code files contain the corporate copyright. Marking this final at the Corporate level prevents changes from lower levels.

To add configuration levels, follow steps:

1. Copy the contents of the `$TOGETHER_HOME$/config` directory to another location to create a separate set of configuration properties for the new level. Possible locations include a shared network location (as described in the previous section) or a local directory in your Together installation (e.g. `config/corporate`).
2. Create a `path.config` file in the `./lib` directory and point it to the directory you set up for the new levels. Together searches this file and loads additional configuration levels from the locations specified. If the location in the previous step is shared, be sure to perform this second step on all the local machines that need to share the configuration.

Adding new configuration directories to a shared location

If you are creating the new levels for a configuration that will be shared from a central location, you have to create a set of configuration properties files for each new configuration level.

To create configuration properties directories for a shared location:

1. Copy the `./config` directory of your Together installation to the shared location and rename it to `tg_shared_config` (or some other meaningful name).
2. For each configuration level you plan to add, copy the `./config` directory of your Together installation to a new subdirectory of `tg_shared_config`. Rename each new subdirectory with the name of the level it represents, such as `tg_shared_config/corporate`.

Tip Create the subdirectories under your local config directory, test that you have defined the new levels correctly, and then copy the config structure to the shared location and modify the levels from there.

Adding new properties directories to a local installation

If you are creating the new levels for a configuration on a local installation, you have to copy the `./config` directory of your Together installation to another location and rename it with the name of the level it represents such as `./config/corporate`.

For example, to create a Corporate configuration level, first copy everything in the config directory to some other directory, such as `$TOGETHER_HOME$/config/corporate`.

Creating the path.config file

Before you create the `path.config` file, make sure that it does *not* already exist. If you are setting up a shared configuration, you need to place a copy of this file in the Together installation of *each* user.

Use a text editor to create the file `$TOGETHER_HOME$/lib/path.config`.

Add the following lines to the file:

```
=== path.config ===
optionsEditor.level.$corporate.name = Corporate
optionsEditor.level.$corporate.visible = true
config.level.$corporate.name = Corporate
config.level.$corporate.visible = true
config.level.$corporate.0 = <path to this level>
=== path.config ===
```

Replace `<path to this level>` with the path to the properties files for the level you are defining, such as `$TOGETHER_CONFIG/corporate/*.config`.

Note `config.level.<level name>.<number> = <path>` defines the source for config files. For shared locations, use the full path. For local paths, specify the full path if outside your Together installation; within that structure you can substitute an appropriate Together System Macro (such as `$TOGETHER_LIB$, $TOGETHER_CONFIG$, $TOGETHER_HOME$`) as part of the path specification.

If you are adding more than one level, you should add some additional lines for each one. For example, to add a Division level:

```
...
optionsEditor.level.$division.name = Division
optionsEditor.level.$division.visible = true
config.level.$division.name = Division
config.level.$division.visible = true
config.level.$division.1 = $TOGETHER_CONFIG$/division /*.config
```

Viewing added configuration levels

To verify that you have correctly added a new level, run Together, open the Options dialog from the main menu, and click the Levels button. (If you created the new levels in a shared location, you will need to launch Together from the command line specifying the path to the configuration.)

In the case of our Corporate example, you should see the new Corporate level in the drop-down list of available levels.

In this way you can configure local installations to include company-wide configurations. Company-wide levels will be configured through `path.config`, and the installation-wide Default level will become user-specific.

Modifying default configuration levels

To get a server installation with user-specific configuration, the entire configuration order should be redefined. Together has the command line parameter

```
-config.path=<path to file which defines all levels>.
```

This option overrides hard-coded levels configuration. To get the hard-coded settings written to a file, pass

```
-debug.config.saveDefaultPath=<path to file>
```

in the command line. You will get a file, which can be used in the `-config.path` option to set the default behavior. This file can be modified to add new levels.

To make a new level visible in the Options dialog, you should include additional lines in the config file (such as `misc.config`):

```
optionsEditor.level.$corporate.name = Corporate
optionsEditor.level.$corporate.visible = true
optionsEditor.level.$department.name = Department
optionsEditor.level.$department.visible = true
```

[...].name is optional. If absent, the internal name will be shown (“corporate”).
[...].visible is necessary, because the level is not visible by default.

B

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 783.](#)
- [“Language-specific configuration files” on page 786.](#)
- [“Using Together with C++” on page 795.](#)

Language Support

LiveSource™ is the Borland technology that keeps class diagrams and source code in sync. You can turn LiveSource on or off for each supported language.

Tip Turn off LiveSource support for unused languages, as using LiveSource can impact system performance.

Switching languages on or off

Together ControlCenter provides LiveSource for all languages. For other Together products, the product type determines which languages have LiveSource support.

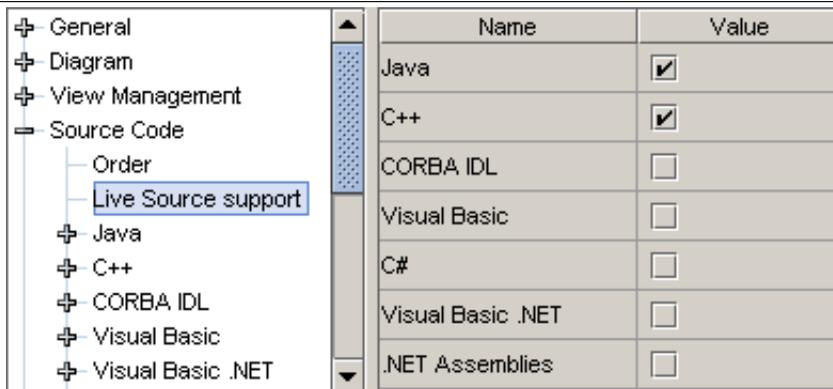
To switch LiveSource for different languages on or off:

1. Choose **Tools | Options | Project** or **Tools | Options | Default** from the main menu in Together.
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 220 is a snapshot of the panes for the Project or Default Options. This figure shows LiveSource turned on for Java and C++ and disabled for other languages.

Figure 220 Language support toggles



You can optionally turn on or off language support through the 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.

Design level modeling

Together includes design-based projects. Design level modeling allows for the creation of language-neutral class diagrams for designing language-independent models. Source code will not be generated for this type of project. The currently-defined UML diagram types and special Together diagrams can be included in a design-based project.

To use design level modeling, choose **Design** as the default language when creating a new project.

If desired, source code can be generated for a design-based project. To generate source code from a design-based project, from the main menu, choose **Tools | Generate | Source Code for Design Project**.

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 JavaBean 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 **JavaBeans / C++ Properties**.
4. Check the boxes in the right pane as desired to view properties.

Note Together does not recognize the CORBA IDL type `union`. Together parses the `union` type elements and presents them on the diagram with empty contents and read-only status.

[Table 82](#) provides summary information on the features for the supported languages and brief notes about language-specific properties.

Table 82 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	yes	yes	yes
Events	no	no	no	yes	yes	yes
Syntax Highlight	yes	yes	yes	yes	yes	yes
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` directory, 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 Visual Basic 6 files
- `vbnet.config`: for VB.NET files.

You can edit these files to specify settings such as codegen and parsing options.

Language-specific Inspectors and usages

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 and the right-click class menu complies with the language of each component.

Visual Basic 6 Inspectors

Visual Basic 6 component inspectors have these special fields:

- **Class and Interface Properties:**
 - **Bean tab:** Use this tab in the inspector to add and remove properties and events. You can set recognition of Visual Basic 6 properties in the View Management page of the Options dialog. See [“Essential features” on page 785](#) for a discussion about properties.
- **Function Properties:**
 - **return type:** Boolean, Byte, Currency, Date, Decimal, Double, Integer, Long, Object, Single, String, and Variant.
 - **visibility:** public, private, friend.
 - **static:** checkbox for adding a static modifier to an operation.
- **Subroutine Properties:** these properties are the same as the function properties listed above. Subroutines do not have a return type property.
- **Attribute Properties:**
 - **type:** Boolean, Byte, Currency, Date, Decimal, Double, Integer, Long, Object, Single, String, and Variant.
 - **visibility:** public, private.
 - **const:** checkbox for adding a const modifier to an attribute.

Visual Basic.Net Inspectors

Visual Basic.Net component inspectors have these special fields:

- **Class Properties:**
 - **Bean tab:** Use this tab in the inspector to add and remove properties or events. You can set recognition of Visual Basic.Net properties in the View Management page of the Options dialog. See [“Essential features” on page 785](#) for a discussion about properties.
 - **namespace:** text area for the class namespace.
 - **inherits:** text area and file chooser button for the list of inherited interfaces.
 - **visibility:** public, protected friend, protected, friend, and private.
 - **MustInherit:** checkbox for adding a `MustInherit` modifier to a class.
 - **NotInheritable:** checkbox for adding a `NotInheritable` modifier to a class.
 - **Shadows:** checkbox for adding a `shadows` modifier to a class.
- **Interface Properties:**
 - **Bean tab:** Use this tab in the inspector to add and remove properties and events. You can set recognition of Visual Basic.Net properties in the View Management page of the Options dialog. See [“Essential features” on page 785](#) for a discussion about properties.
 - **namespace:** text area for the interface namespace.
 - **inherits:** text area and file chooser button for the list of inherited interfaces.
 - **visibility:** public, protected friend, protected, friend, and private.
- **Function Properties:**
 - **return type:** Boolean, Byte, Currency, Date, Decimal, Double, Integer, Long, Object, Single, String, and Variant.
 - **visibility:** public, protected friend, protected, friend, and private.
 - **Overloads:** checkbox for adding an `Overloads` modifier to a function.
 - **Overrides:** checkbox for adding an `Overrides` modifier to a function.
 - **Overridable:** checkbox for adding an `Overridable` modifier to a function.
 - **NotOverridable:** checkbox for adding a `NotOverridable` modifier to a function.
 - **MustOverride:** checkbox for adding a `MustOverride` modifier to a function.
 - **Shadows:** checkbox for adding a `Shadows` modifier to a function.
 - **Shared:** checkbox for adding a `Shared` modifier to a function.

- **Subroutine Properties:** these properties are the same as the function properties listed above. Subroutines do not have a return type property.
- **Attribute Properties:**
 - **type:** Boolean, Byte, Currency, Date, Decimal, Double, Integer, Long, Object, Single, String, and Variant.
 - **visibility:** public, protected friend, protected, friend, and private.
 - **const:** checkbox for adding a const modifier to an attribute.
 - **Shadows:** checkbox for adding a Shadows modifier to an attribute.
 - **Shared:** checkbox for adding a Shared modifier to an attribute.

C# Inspectors

C# component inspectors have these special fields:

- **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 and internal for a class. In addition to public and internal, inner class elements use protected internal, protected, and private.
 - **new:** checkbox for adding a new modifier to a class (visible for inner classes only).
- **Interface Properties:**
 - **namespace:** text area for the class namespace.
 - **extends:** text area and file chooser button for the list of inherited interfaces.
 - **visibility:** public and internal.
- **Operation Properties:**
 - **return type:** bool, byte, char, decimal, double, float, int, long, object, sbyte, short, string, uint, ulong, ushort, and void.
 - **new:** checkbox for adding a new modifier to an operation.
 - **visibility:** public, protected internal, protected, internal, and private.

- **static**: checkbox for adding a `static` modifier to an operation. `virtual`, `abstract`, and `override` modifiers are not valid selections when choosing this property.
- **virtual**: checkbox for adding a `virtual` modifier to an operation. `static`, `abstract`, and `override` modifiers are not valid selections when choosing this property.
- **override**: checkbox for adding a `override` modifier to an operation. `new`, `static`, and `virtual` modifiers are not valid selections when choosing this property.
- **abstract**: checkbox for adding a `abstract` modifier to an operation. `static`, `extern`, and `virtual` modifiers are not valid selections when choosing this property.
- **extern**: checkbox for adding as `extern` modifier to an operation. The `abstract` modifier is not a valid selection when choosing this property.
- **Field Properties**:
 - **type**: `bool`, `byte`, `char`, `decimal`, `double`, `float`, `int`, `long`, `object`, `sbyte`, `short`, `string`, `uint`, `ulong`, and `ushort`.
 - **new**: checkbox for adding a `new` modifier to an attribute.
 - **visibility**: `public`, `protected internal`, `protected`, `internal`, and `private`.
 - **static**: checkbox for adding a `static` modifier to an attribute.
 - **readonly**: checkbox for adding a `readonly` modifier to an attribute.

CORBA IDL Inspectors

CORBA IDL inspectors have these special fields:

- **Struct Properties**:
 - **module**: text area to enter a module name for a struct.
- **Struct Attribute Properties**:
 - **type**: `any`, `boolean`, `char`, `double`, `float`, `long`, `long double`, `long long`, `octet`, `short`, `string`, `unsigned long`, `unsigned long long`, `unsigned short`, and `wchar`, and a file chooser button to browse to your own type.
- **Interface Properties**:
 - **module**: text area to enter a module name for an interface.
- **Interface Operation Properties**:
 - **return type**: `any`, `boolean`, `char`, `double`, `float`, `long`, `long double`, `long long`, `octet`, `short`, `string`, `unsigned long`, `unsigned long long`, `unsigned short`, and `void`, and a file chooser button to browse to your own return value.

- raises: text area and file chooser button to associate an exception with an operation.
- oneway: checkbox for adding a `oneway` modifier to an interface operation.
- Interface Attribute Properties:
 - type: any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short, and wchar, and a file chooser button to browse to your own type.
 - read only: checkbox for adding a `read only` modifier to an interface attribute.
- Interface Struct and Exception Properties:
 - abstract: checkbox for adding an `abstract` modifier to an exception or struct inside of a valuetype.
 - persistent: checkbox for adding the `@persistent` javadoc comment above the code of an exception or struct inside of an interface.
- Valuetype Properties:
 - module: text area to enter a module name for a valuetype.
 - supports: text area and file chooser button for an interface name.
 - custom: checkbox for adding a `custom` modifier to a valuetype.
- Valuetype Operation Properties:
 - return type: any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short, and void, and a file chooser button to browse to your own return value.
 - raises: text area and file chooser button to associate an exception with an operation.
 - oneway: checkbox for adding a `oneway` modifier to a valuetype operation.
- Valuetype Attribute Properties:
 - type: any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short, and wchar, and a file chooser button to browse to your own type.
 - read only: checkbox for adding a `read only` modifier to a valuetype attribute.
- Valuetype Struct and Exception Properties:
 - abstract: checkbox for adding an `abstract` modifier to an exception or struct inside of a valuetype.
 - persistent: checkbox for adding the `@persistent` javadoc comment above the code of an exception or struct inside of a valuetype.
- Exception Properties:

- **module:** text area to enter a module name for an exception.
- **Exception Attribute (Member) Properties:**
 - **type:** any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short, and wchar, and a file chooser button to browse to your own type.
- **Generalization Link Properties:** A generalization link can be drawn between two interfaces or two valuetypes. It can also be drawn between an interface and a valuetype, where the client for the link is a valuetype and the supplier is an interface. When drawing a generalization link between two valuetypes, the `truncatable` property displays.
 - **truncatable:** checkbox for adding the truncatable keyword to a valuetype to specify that a valuetype can safely be truncated.

CORBA IDL tips

The following are tips regarding CORBA IDL language support in Together:

- Together does not recognize the CORBA IDL type `union`. Together parses the `union` type elements and presents them on the diagram with empty contents and read-only status.
- CORBA IDL exception-type classes do not contain operations. These class types contain only attributes.

C++ Inspectors

C++ inspectors have these special fields:

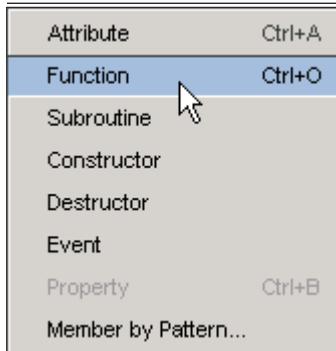
- **Class and Interface properties:**
 - **C++ Properties tab:** Use this tab in the inspector to add and remove properties. You can set recognition of C++ properties in the View Management page of the Options dialog. See [“Essential features” on page 785](#) for a discussion about properties.
 - **definition file:** lists the filename of the associated definition file.
 - **namespace:** text area for the class namespace.
- **Operation properties:**
 - **return type:** bool, char, double, float, int, long, long double, short, string, unsigned, unsigned long, unsigned short, and void.
 - **definition file:** text area and file chooser for the *.cpp file.
 - **throw:** text area and file chooser button to help with exceptions.
 - **visibility:** public, protected, and private.
 - **virtual:** checkbox for adding a virtual modifier to a function.

- `const`: checkbox for adding `const` keyword to a function.
- `inline`: checkbox for indicating an operation as `inline`.
- `volatile`: checkbox for adding the `volatile` keyword to an operation.
- Attribute properties:
 - `type`: `bool`, `char`, `double`, `float`, `int`, `long`, `long double`, `short`, `string`, `unsigned`, `unsigned long`, and `unsigned short`.
 - `visibility`: `public`, `protected`, and `private`.
- Generalization Link properties:
 - `visibility`: `public`, `protected`, `private`.
 - `virtual`: checkbox for adding a `virtual` modifier to the link.

Visual Basic 6 class members

The class right-click menu for every language has a New command for creating additional class members. The choice of members varies according to language of the class. Figure 221 shows the cascading menu choices for Visual Basic 6.

Figure 221 Choice of new class members for a Visual Basic 6 class.



Since Visual Basic 6 distinguishes between functions and subroutines, *Function* and *Subroutine* (rather than *Operation*) display on the Visual Basic 6 class right-click menus. Additional subroutines on that menu are *Constructor* (for `Class_Initialize()`) and *Destructor* (for `Class_Terminate()`).

To create a Visual Basic property, select *Member by Pattern* from the class right-click menu. This displays a dialog box offering two kinds of properties:

- Property Get & Let: creates a primitive type attribute with Get and Let operations.
- Property Get & Set: creates an object type attribute with Get and Set operations.

If your project recognizes Visual Basic properties, then a class with a Visual Basic property displays in a class diagram with the characteristic property tab in the upper left corner.

Tip You can toggle recognition of VB Properties by selecting **Tools | Options | <Level>** from the main menu. In the resulting dialog box, expand the View Management node, select Properties and Events, and check *Recognize properties and events*.

Visual Basic.Net class members

The Visual Basic.Net class right-click menu is similar to that of Visual Basic 6. The choice of members varies somewhat. [Figure 222](#) shows the menu choices for Visual Basic.Net.

Figure 222 Choice of new class members for a Visual Basic.Net class.



Visual Basic.Net also distinguishes between functions and subroutines, *Function* and *Subroutine* (rather than *Operation*) display on the class right-click menu. Additional subroutines on that menu are *Constructor* (for `Sub New()`) and *Event* (for `Public Event event1 As System.Delegate()`).

To create a Visual Basic.Net property, choose **Property** from the class right-click menu.

If your project recognizes Visual Basic.Net properties, then a class with a property displays in a class diagram with the characteristic property tab in the upper left corner.

Renaming properties

Bean properties and events are recognized in the C#, VB.Net and VB6 classes. When a property name or type is modified in a class diagram, the relevant changes are reflected in the getter and setter headers in the source code. However, the getter and setter bodies do not change. It is the user's responsibility to provide consistent renaming, to avoid incompatible code.

Using Together with C++

Together was originally designed to support only a single language, C++. This section explains the special features of Together that enable you to use it effectively for developing C++ projects.

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 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 parses all files in the `#include` searchpath.

Together uses the standard convention to discriminate the project internal and external headers. External files are included via `#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 **Tools | Options | Project Level** or **Tools | Options | Default Level** from the main menu.
2. Expand **Source Code** then **C++** on the left panel of the resulting dialog box. [Figure 223](#) shows the available C++ options.
3. Check **Skip Standards Include** on or off as desired.
4. Click **Ok** to save the option and close the dialog.

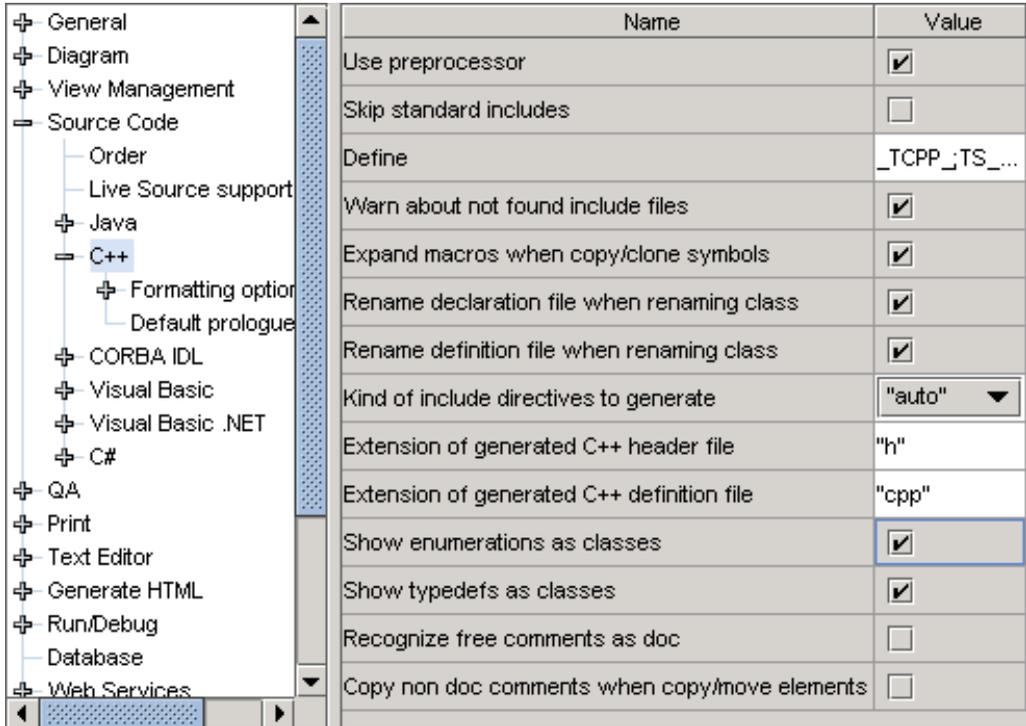
Adding Standard Library Paths

If you use a standard library (or libraries) for your C++ projects, you can add it to your default project settings. Each time that you create a new C++ project, the library or libraries will automatically be added to the project's Search/Classpath.

To add standard library paths:

1. Choose **Tools | Options | Default Level** from the main menu.
2. Expand **General** then **New Project** on the left panel of the resulting dialog box.
3. Update the **Standard Paths** field to define paths to directories where standard libraries are located.
4. Click **Ok** to save the option and close the dialog.

Figure 223 C++ Options



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.

Note There are two macros in `preinclude.inc` that are named `BEGIN_MESSAGE_MAP`. One is necessary for MFC and the other for Borland. The presence of the two macros may cause a conflict.

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. Also, make sure 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 Level** or **Tools | Options | Project Level** from the main menu.
2. Expand **Source Code** then **C++** in the left panel, as shown earlier in [Figure 223](#).
3. 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 (;).
4. 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 can change these default values through the default or project options for C++. [Figure 223](#) shows the C++ options.

To change the file extensions for C++ code generated by Together:

1. Choose **Tools | Options | Default Level** or **Tools | Options | Project Level** from the main menu.
2. Expand **Source Code** then **C++** on the left panel, as shown earlier in [Figure 223](#).
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 does not accept any other extensions.

Incidentally, the file `$(TGHS)/config/cpp.config` determines 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 does not provide a default C++ compiler. However, you can set up a C++ compiler and make utility as external tools using the Options dialog. To configure the compile and make commands:

1. Choose **Tools | Options | Default Level** or **Tools | Options | Project Level** from the main menu.
2. Expand **Tools** on the left panel to reveal a tree-view of the available tools.
3. Select the tool on the left panel that you want to configure.

Tip Do not overwrite default compiler/make settings. Use the empty "Tool #x" slots instead.

4. In the right panel, fill in the settings. For proper setup, follow the help commentary displayed in the Description section of the dialog box.
 1. **Language:** from the language dropdown 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 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 executable.
 4. Set the remaining options as desired.
5. In the left panel, expand the selected tool. Then click **Show in menu**.
6. In the right panel, 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, using the **Show in menu** option, you must also specify the menus where these commands should be displayed. To execute your compiler and make utilities, go to the menu(s) that you designated when the tool was configured. For example, if you chose to show your tool in all diagram elements right click menus, then you would access the tool by right clicking on a class and choosing **Tools | C++ Compiler**.

Working with C++ language features

This section discusses how Together works with special C++ language constructs and organization.

Header and implementation files

There are two kinds of C++ source code files: header files and implementation files. When you place a new class on a class diagram, Together creates a header file with a name that matches the class. When you add anything to the class that is usually defined outside the header such as an operation, Together creates an implementation file as well. The C++ language options determine the suffixes for each file (as shown in [Figure 223](#)).

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.

If a header file is read-only, Together does not allow you to change the file in the Designer pane or in the Editor. If the implementation file is read-only, you can change the header file in the Designer pane or in the Editor. In that case, however, the Message pane displays an error message.

If all of the operations of a class are inline, then there is no implementation file for the class.

To inline an operation:

1. Right click the operation to invoke its right-click menu.
2. Click **Inline**.

Together removes the definition of the operation from the implementation file and writes it into the header file. If the implementation file has no other code (such as definitions of operations or static attributes), Together deletes the file.

The inline process described above serves to remove inlining from an operation as well. When you remove the inline from an operation, Together creates an implementation file if necessary and moves the definition of the operation from the header file to the implementation file.

Templates, structures, unions, and overloaded operators

Together provides class patterns for templates, structs, unions, and classes with constructors. In addition, there are member patterns for overloaded operators.

Note The pattern for a class with constructors specifies a null constructor, copy constructor, and virtual destructor.

To create a C++ template:

1. Create a class.
2. Right click the class on the diagram and choose **Choose Pattern** from the menu.
3. In the left panel of the resulting dialog box, click **Default Template**.
4. Click **Ok** to apply the pattern and quit the dialog.

The same technique for creating templates is appropriate for creating a struct or union.

Template patterns can apply to any existing class, struct, union, or interface. If you apply a struct or union pattern to a template, Together removes the template designation from the code and the diagram.

Important Together automatically generates the correct syntax for template operations. However, if you manually corrupt the implementation for a template operation by incorrectly modifying the angle braces (the `<class T>` for example), Together does not issue a warning.

If you create a C++ template as described above, Together creates a class template with a single parameter, `class T`. For a C++ class template pattern with additional parameters, you can modify Default Template or create a new C++ template pattern. The modification is straightforward. Simply change the template definition to this:

```
template <$Template_Parameters$> class %Name% { };
```

With this modification, the Pattern dialog adds a string field named *Template Parameters*. When you apply the pattern, you can enter a string for the formal parameter list.

Instructions on editing Together template patterns are in [“Custom code templates” on page 508](#).

Note If you create an association from a class to a template or between two templates, Together draws the link on the diagram but comments out the actual declaration of the link in the class. You must uncomment the code and supply the template type and the actual template parameters.

Interfaces

When you put a new interface on a class diagram, Together creates an abstract class and gives it the stereotype *Interface*. When you add a new operation to an interface, Together declares it to be a pure virtual function.

Enumerations, typedefs and global symbols

Together treats global enumerations, typedefs, and functions and variables according to the C++ language options.

- Global enumerations: If *Show enumerations as classes* is checked in the project options, global enumerations display in class diagrams as class-like nodes with the stereotype *enum*. [Figure 223](#) shows the C++ options.

Enumeration nodes do not support inheritance or operations. You can add a new enumerator to an enumeration by selecting **New | Attribute** from its right-click menu.

- Global typedefs: If *Show typedefs as classes* is checked in the project options, global typedefs display in class diagrams as class nodes or rectangles. The Explorer lists these types as ordinary model elements. Furthermore, if you use the Inspector of an attribute or an operation to select its type, global typedefs are listed among the available model element types.

- If the typedef declaration simply creates another name for a known type, it appears on a class diagram as a rectangle with the stereotype `typedef` and the typedef identifier as the name. The notation 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 the typedef declaration defines a new structure, it appears on the class diagram as a class-like node with the stereotype `struct`. If the structure is anonymous, the name on the diagram node is the name of the structure variable (or the first such variable if there are more than one).
- 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.

Properties

Data members that have getters or setters are properties. By default, Together displays class properties in separate compartment of the class node and places a small rectangle in the upper left corner of the node. The project options determine the display of C++ properties.

To change the view options on C++ properties:

1. Choose **Tools | Options | Project Level** or **Tools | Options | Default Level** from the main menu.
2. Expand **View Management** then click **JavaBeans/C++ Properties** on the left panel of the resulting dialog box.
3. Check **Recognize C++ Properties** on the right panel as desired. Then click **Ok** to save the option and close the dialog.

You can create a C++ property in a class by right clicking the class node and choosing **New | Property** from the menu. This creates a private data member along with public getter and setter methods.

Note You can edit the name and the type of property with the in-place diagram editor. You cannot change the type of a property data member to a reference type (a type of the form `type&`). You can, however, use a reference type as the formal argument of the setter or the return type of the getter.

To edit the definitions of the getter and setter of a property, right-click the property on the class node and choose **Edit Definition**. Together opens the implementation file and selects the definition of the getter or setter. The Edit Definition operation cycles through getter and setter, selecting one of the methods one time and the other method the next.

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 );
}
```

--- 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.

To avoid this behavior, use either short names in both places (`string`) or full names in both places (`std::string`).

Tip When you specify an initial value for a static attribute that is a string or array of characters, be sure to enclose the value in double quotes to designate that the value is of type `string`. This is true regardless of whether you specify that initial value in an Inspector field or directly in the class diagram.

Together patterns for member operations

C++ operations are typically defined in implementation files rather than inlined with the class definitions on the header files. When you create an operation by pattern, Together now follows this convention, defining the operation in the implementation file and declaring it in the header file.

Documentation of C++ member definitions

C++ Definition Doc-Comment is an activatable feature for documenting the definitions of C++ class member functions. This feature can be used to create general descriptions as well as Javadoc tags to describe parameters, return values, related code, and version information inside the implementation files.

To activate the C++ Definition Doc-Comment 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. The feature becomes active immediately.

Using the definition documentation feature

When the C++ Definition Doc-Comment feature is active, the Inspector for C++ class member functions displays a *Definition documentation* tab with five fields (see [Figure 224](#)):

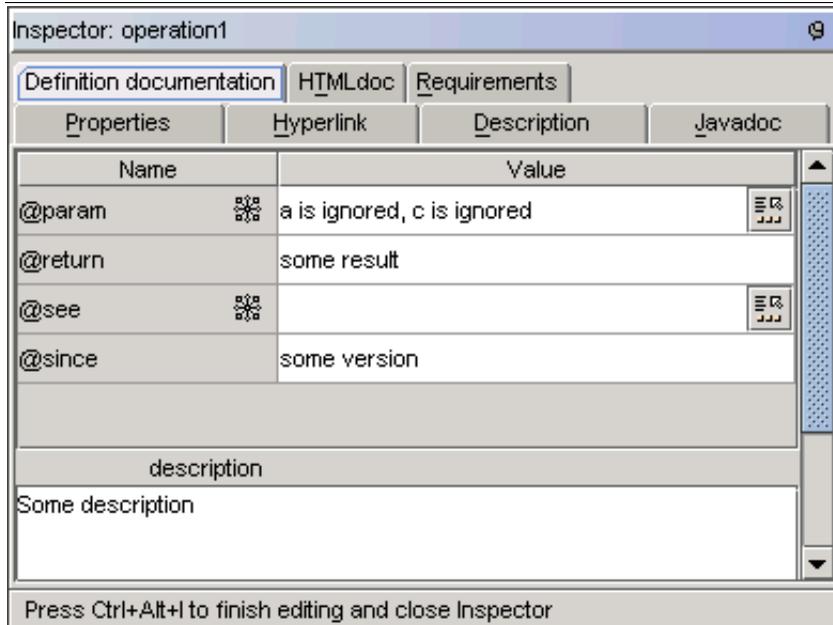
- *@param*: formal parameter descriptions
- *@return*: return value description
- *@see*: related code
- *@since*: versions

- *description*: comments to be embedded in the documentation before the tags (the previous four fields).

You can edit documentation of member definitions on the *Definition documentation* tab the same way as on the *Description* and *Javadoc* tabs.

Together adds documentation in the *Definition documentation* tab to the definition of the member (in the implementation file, unless the definition is inlined); documentation from the *Description* and *Javadoc* tabs goes in declaration file instead.

Figure 224 Definition documentation tab for C++ member function.



Accessing the defDocComment module through the API

The C++ *Definition Doc-Comment* feature uses the *defDocComment* module. You can access this module directly by using code as follows:

```
import com.togethersoft.openapi.rwi.*;
import com.togethersoft.openapi.rwi.enum.*;
...
RwiElement member = ...;
RwiPropertyMap defDocs = null;
if(member.hasProperty("DefDocComment")) {
    RwiPropertyEnumeration e = member.properties("DefDocComment");
    if(e.hasMoreElements()) {
        defDocs = e.nextRwiProperty().getSubproperties();
    }
}
String description = defDocs.getProperty(null);
String since = defDocs.getProperty("since");
RwiPropertyEnumeration authors = defDocs.properties("author");
```

```
RwiPropertyEnumeration paramDocs = defDocs.properties("param");
```

The module registers support for the boolean property `DefDocComment`, which represents the document comments in the definition. An instance of this property has subproperties for each of the different documentation tags. The description part (that is, the text preceding any tag) is the subproperty with a null name.

Enabling HTML, PDF, and RTF documentation of member definitions

You can make documentation of member definitions available to Together's Generate Documentation Using Template feature (described in [Chapter 22](#)) by editing the `MetaModel.mm` file:

```
$TGH$/modules/com/togethersoft/modules/gendoc/templates/MetaModel.mm
```

To activate this functionality, three modifications are required:

1. At the top section of the file under the comment

```
# RWI specific properties
```

add the following line:

```
DefDocComment = "Definition doc-comment", boolean;
```

immediately after line:

```
visibility = "Visibility";
```

2. Add `DefDocComment` to the list of properties for the metatype `GENERIC_OPERATION` by editing the `properties` line below the line `name=GENERIC_OPERATION` from:

```
properties = { $fullname;
```

to:

```
properties = { DefDocComment;  
              $fullname;
```

3. Add `DefDocComment` to the list of properties for the metatype `ATTRIBUTE` by changing the `properties` line below the line `name=ATTRIBUTE`. Make the changes identical to those listed in the previous step.

These modifications enable the documentation generation module to recognize the `DefDocComment` property as a standard property of elements.

Using the sample documentation template

The `defDocComment` module comes with a sample documentation template file named `ClassDefDoc.tpl`. This template contains a stock section named `Definition documentation` that demonstrates using the C++ Definition Doc-Comment feature.

The pathname of the `ClassDefDoc.tpl` template is:

```
$TGH$/modules/com/togethersoft/modules/defDocComment/ClassDefDoc.tpl
```

To see how `ClassDefDoc.tpl` supports the C++ Definition Doc-Comment feature, follow these steps:

1. Activate the C++ Definition Doc-Comment feature.

2. Open the sample project:

```
$TGH$/modules/com/togethersoft/modules/defDocComment/sample/sample.tpr
```

3. Edit the `MetaModel.mm` file as described in the previous section.

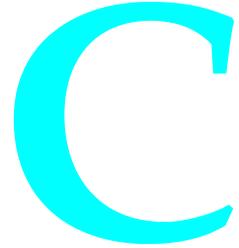
4. Generate the documentation for the sample project using the standard template `ClassReport.tpl`. Observe the results.

5. Remove the original `Class.tpl` file in the gendoc library:

```
$TGH$/modules/com/togethersoft/modules/gendoc/templates/lib/Class.tpl
```

6. Copy the sample `ClassDefDoc.tpl` template from the `defDocComment` folder to the old `Class.tpl` location shown in the previous step. Rename the copied file to `Class.tpl`.

7. Generate documentation as in step #4. Observe the results for the member function `operation1()`.



Commands, Macros, and Keyboard Shortcuts

This appendix includes the following topics:

- [“Command Line Parameters” on page 811](#)
- [“Parameters for the Together.exe Launcher” on page 815](#)
- [“System Macros” on page 816](#)
- [“Template Macros” on page 820](#)
- [“Keyboard shortcuts” on page 821](#)

Command Line Parameters

This section provides the command line parameters for Together and includes the following topics:

- [“Basic command line syntax” on page 811](#)
- [“Using the Windows launcher” on page 813](#)
- [“Invoking the Together main class” on page 813](#)

Basic command line syntax

[Table 83](#) lists the command line syntax for:

Together_starter [Options] [ProjectFile].

Table 83 Command line syntax

Together_starter	<p>Command that starts Together.</p> <p>On the Windows platform, Together has the following launchers:</p> <ul style="list-style-type: none">• Together.exe with disabled console output,• TogetherCon.exe, that uses the existing console, or opens a new console for output. <p>The 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 -gui.</p> <p>For other platforms, you can create Together.sh, Together.cmd, etc. depending on the operating system.</p> <p>Alternatively, this can be a complicated command that calls your Java VM, specifying parameters for it, followed by the Together main class name (com.togethersoft.together.Main) followed by any parameters for that class. See “Invoking the Together main class” on page 813 for details.</p>
------------------	---

Table 83 Command line syntax

Options	<p>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:</p> <pre>-script=com.togethersoft.modules.helloworld.HelloJava</pre> <p>or</p> <pre>-script:com.togethersoft.modules.helloworld.HelloJava</pre> <p>Note: Use the colon (:) symbol under Windows, since "=" is not recognized by the Windows command line interpreter.</p>
ProjectFile	<p>Path to a Together project file to be opened. The file name must have .tpr extension. Optional for invoking modules that do not need to access model information from a specific project.</p> <p>Tip In UNIX systems, the launcher takes spaces for parameters separators. Thus, the project file names, which contain spaces, are treated as the different file names. To avoid this problem, the spaces should be preceded with the backslash characters. For example, if the project file name is Java Project.tpr, it should be written as Java\ Project.tpr.</p>

Using the Windows launcher

When Together is started using one of the Window-specific launchers (`Together.exe`, `TogetherCon.exe`, or `umldoc.exe`), the Environment variables and JVM parameters are taken from the `Together.bat` file (`$TGHS\bin\Together.bat`). If there is no built-in Java-machine in the Together installation, the launchers make use of the JVM that corresponds to the value of JDK variable as defined in `Together.bat` file.

For complete parameters of `Together.exe` see ["Parameters for the Together.exe Launcher" on page 815](#).

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 84](#) lists the parameters for main class.

Table 84 Parameters of the main class.

<code><Project.tpr></code>	Specifies the fully qualified name of Together 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, Together 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 a 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 85](#) lists valid options for the following:

Together.exe [options] [-c[class_name]] [parameters]

Table 85 Options for Together.exe

Option	Description
-?, -h, -help	Print this usage message and exit
-con, -nocon	Show/don't show console (default without new console)
-cmd	Print launched command line
-nowarn	Do not show warning messages
-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 (same as -cp:p <path>)
-cp:a <path>	Append <path> to classpath
-verify, -noverify	Verify/don't verify classes when loaded
-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
-J<runtime flag>	Pass argument to the Java interpreter
-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 category of the Options dialog. [Table 86](#) uses **bold** text to indicate references to the Tools category.

Table 86 System macros

Macro	Description
\$TGH\$	Contains the full path to your Together's installation. For example (c:\Together).
\$TOGETHER_HOME\$	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 <i>Destination directory</i> option, which defines the destination directory for class files (Builder Built-in Javac Compiler options in 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 returns "\$MAINCLSS\$" and displays the error message in the Messages tab "No class with main() method found".
\$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".

Table 86 System macros (continued)

\$PROJECT_EXT\$	Contains the extension of the project file. For example, if the project file is <code>c:\together\myprojects\CoolProject\myproj.tpr</code> , then this macro will contain <code>"tpr"</code> .
\$PROJECT_FULLNAME\$	Contains the name (with extension) of the project file. For example, if the project file is <code>c:\together\myprojects\CoolProject\myproj.tpr</code> , then this macro will contain <code>"myproj.tpr "</code> .
\$PROJECT_SPEC\$	Contains the full name of the project file (path, name and extension). For example, if the project file is <code>c:\together\myprojects\CoolProject\myproj.tpr</code> , then this macro will contain the value: <code>"c:\together\myprojects\CoolProject\myproj.tpr"</code> .
\$FILELIST\$	Contains the <code>"@somefile"</code> , where <code>somefile</code> 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, <code>somefile</code> will contain only the file with the selected class, but if a diagram is clicked, <code>somefile</code> 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 <code>c:\together\myprojects\CoolProject</code> directory, then this macro will contain <code>"c:\together\myprojects\CoolProject"</code>
\$FILE_NAME\$	Contains the name (without extension) of the selected file. For example, if the selected file is <code>c:\together\myprojects\CoolProject\MyClass.java</code> , then this macro will contain <code>"MyClass"</code> .
\$FILE_EXT\$	Contains the extension of the selected file. For example, if the selected file is <code>c:\together\myprojects\CoolProject\MyClass.java</code> , then this macro will contain <code>"java"</code> .
\$FILE_FULLNAME\$	Contains the name (with extension) of the selected file. For example, if the selected file is <code>c:\together\myprojects\CoolProject\MyClass.java</code> , then this macro will contain <code>"MyClass.java"</code> .
\$FILE_SPEC\$	Contains the full name of the selected file (path, name and extension). For example, if the selected file is <code>c:\together\myprojects\CoolProject\MyClass.java</code> , then this macro will contain <code>"c:\together\myprojects\CoolProject\MyClass.java"</code> .
\$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 <code>c:\together\myprojects\CoolProject</code> directory, then this macro will contain <code>"c:\together\myprojects\CoolProject"</code> . Definition files has <code>.cpp</code> extension

Table 86 System macros (continued)

<code>\$DEF_FILE_NAME\$</code>	Contains the name (without extension) of the selected definition file (C++ only). For example, if the selected definition file is <code>c:\together\myprojects\CoolProject\guest.cpp</code> , then this macro will contain "guest".
<code>\$DEF_FILE_EXT\$</code>	Contains the extension of the selected definition file (C++ only). For example, if the selected definition file is <code>c:\together\myprojects\CoolProject\guest.cpp</code> , then this macro will contain "cpp".
<code>\$DEF_FILE_FULLNAME\$</code>	Contains the name (with extension) of the selected definition file. For example, if the selected definition file is <code>c:\together\myprojects\CoolProject\guest.cpp</code> , then this macro will contain "guest.cpp".
<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 " c:\together\myprojects\CoolProject\guest.cpp".
<code>\$PROMPT\$</code>	<p>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.</p> <p>Variants:</p> <ul style="list-style-type: none"> • <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. <p>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.</p>

Referencing configuration properties as macros

Besides using the predefined macros listed in [Table 86](#), 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 87 Template macros

Macro	Definition	Used in	Language
<code>%Class_Name%</code>	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.	Source-generating templates	All languages
<code>%Date_Created%</code>	This macro is expanded into the date of creation in the format: Month XX, YYYY	Prologue and epilogue	All languages
<code>%Time_Created%</code>	This macro is expanded into the time of creation in the format: hh:mm:ss AM/PM	Prologue and epilogue	All languages
<code>%FILE_NAME%</code>	Name of the class source code file.	Prologue and epilogue	All languages
<code>%FILE_EXT%</code>	Extension of the class source code file.	Prologue and epilogue	All languages
<code>%Name%</code>	Name of a generated class / attribute / operation, editable in the Choose Pattern dialog.	Source-generating templates	All languages
<code>%Dst%</code>	Name of the destination class of a generated link, editable in the Choose Pattern dialog.	Source-generating templates	All languages
<code>%Type%</code>	Type of attribute or return type of operation, editable in the Choose Pattern dialog.	Source-generating templates	All languages
<code>%Header_File%</code>	The header file path.	Epilogue for generation of the <code>#include</code> directive	C++
<code>%Any%</code>	Matches any token.	Source-generating templates	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, grouped by their location in the Main Menu, and by their functions. For this reason, some of the shortcuts are duplicated in various tables (e.g. SHIFT+F7 shortcut for Make Project is included both in the table of the Project menu, and the table of all Compile/Make/Run commands).

- Main window and main menu shortcuts
- Diagrams
- Editor/Compiler/Debugger
- Miscellaneous shortcuts

Main Window and 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.

Note Attention, MacOS users! Though normally in MacOS the Command key is used for the keyboard shortcuts, in Together CTRL key is used instead.

Table 88 Main Window shortcuts

Switching between the inner tabs	ALT+ RIGHT/LEFT ARROW
Expand node	Right arrow, ENTER, double click
Collapse node	Left arrow, ENTER, double click
Next Pane	CTRL+F12

Table 88 Main Window shortcuts

Previous Pane	CTRL+SHIFT+F12
Next Tab	ALT+RIGHT
Previous Tab	ALT+LEFT

Table 89 File shortcuts

New project	CTRL+SHIFT+N
New (Object Gallery)	CTRL+N
Open	CTRL+O
Save	CTRL+S
Save all	CTRL+SHIFT+S
Print Diagram	CTRL+P
Print Documentation	CTRL+SHIFT+P
Close	CTRL+W
Close all	CTRL+SHIFT+W
Exit	ALT+X

Table 90 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
Select All	CTRL+A

Table 91 Search shortcuts

Find	CTRL+F
Replace	CTRL+H
Find Next	F3
Find Previous	SHIFT+F3
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
Go to Line	CTRL+G

Table 92 View shortcuts

Toggle Explorer Pane	CTRL+ALT+Z
Toggle Editor Pane	CTRL+ALT+E

Table 92 View shortcuts

Toggle Designer Pane	CTRL+ALT+D
Toggle Message Pane	CTRL+ALT+M
Toggle Inspector	CTRL+ALT+I
Toggle Toolbox	CTRL+ALT+T
Full screen view	F12

Table 93 Project shortcuts

Make Project	SHIFT+F7
Rebuild Project	CTRL+SHIFT+F7
Version Control/ System	CTRL+Q

Table 94 Run shortcuts

Run / Debug Configuration	SHIFT+F10
Run	F9
Debug	CTRL+F9
Attach to Remote Process	SHIFT+F5
Step Over	F8
Step Into	F7
Run to Cursor	F4

Table 95 Selection shortcuts

New attribute	INSERT, CTRL+A
New operation	INSERT, CTRL+O
New Property	INSERT, CTRL+B
Open Inspector	ALT+ENTER
Rename	F2
Choose Pattern	CTRL+R
Clone	CTRL+SHIFT+C
Delete	Delete
Refactoring/Rename	SHIFT+F2

Table 96 Help shortcuts

Contents	SHIFT+F1
----------	----------

Designer shortcuts

Table 97 Designer shortcuts

Deselect element	ESC
Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Invoke right-click menu of the selected element, when focus is lost	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
Rename	F2

Table 98 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 99 Class diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 100 Class shortcuts

New attribute	CTRL+A
New operation	CTRL+O
New member by pattern	CTRL+T
New property	CTRL+B

Table 101 Package diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 102 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 103 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 104 Compile and Run/Debug shortcuts

Make	SHIFT+F8
Rebuild	CTRL+SHIFT+F8
Run configuration dialog	SHIFT+F10
Run	F9
Debug	CTRL+F9, F7
Terminate debug process	CTRL+F2
Resume debug process	CTRL+SHIFT+F2
Step over	F8
Step into	F7
Run to cursor	F4
Attach to remote process	SHIFT+F5
Navigate to the next error in the Builder pane	N
Navigate to the previous error in the Builder pane	P

Table 105 Editor shortcuts

Import Assistant	ALT+ENTER
Insert snippet	CTRL+J
Next snippet tag	CTRL+SHIFT+J
Fast expand snippet	Space or tab
Code Sense	CTRL+SPACE
Advanced Code Sense	SHIFT+SPACE
Parameter tooltip	CTRL+F8
Tag Library Helper	CTRL+F1 / Command+F1 (Mac OS)
Toggle Rectangular / Line block modes	CTRL+L
Select rectangular block	ALT+mouse
Select a word	Double click
Select a line	Triple click
Collapse / Expand all method bodies	CTRL+SHIFT+NumPad - / CTRL+SHIFT+NumPad +
Collapse / Expand current method body	CTRL+NumPad - / CTRL+NumPad +
Format Source	CTRL+T
Invoke Surround with	CTRL+SHIFT+R
Insert Bookmark	CTRL+M
Invoke Bookmark Dialog	CTRL+D

Table 105 Editor shortcuts

Navigate to the next/previous Global Bookmark	CTRL+Minus or CNTL+Equals
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 matching right curly brace	CTRL+}
Navigate to the matching left curly brace	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
Move selection to the right n tab spaces	TAB
Move selection to the left n tab spaces	SHIFT+TAB

Note You can configure keyboard shortcuts for the editor using Text Editor Options (*Text Editor - Keyboard - <scheme>*)

Miscellaneous

Table 106 Chooser shortcuts

Invoke the right-click menu in the file chooser dialog	SHIFT+F10
Expand node	Numeric keypad plus (Gray +)/ Enter
Collapse node	Numeric keypad minus (Gray -)/ Enter
Open any chooser dialog in the inspector or Options dialog	ALT+INSERT

Table 107 Version control shortcuts

System dialog	CTRL+Q
Refresh dialogs of an SCC provider	ALT+TAB

Table 108 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

INDEX

A

- accelerators, defining 338
- activity diagram 131, 188
 - elements 188
 - shortcuts 825
 - techniques for 189
- addToCompartment function 746
- Apache SOAP 705, 706
 - generating proxy Web service client 716
- API
 - documentation 774
 - IDE 756
 - overview 755
 - RWI 756
 - SCI 756
- applet 620
 - creating 620
 - deploying 621
 - running and debugging 620
- application client 645
 - creating “one-click” 648
- application client diagram 557
 - deploying 649
 - security support 558
 - using 649
 - visual design elements 648
- Application server
 - BEA WebLogic 678
 - IBM WebSphere 678
 - Sun EE reference implementation 677, 678
- array
 - evaluating 294
- assertion
 - inserting in visual script 488
- attribute 763
 - adding using the XML editor 261
 - breakpoints, setting 292
- audits and metrics 523
 - additional sources 550
 - automatic correction of audit violations 537

- customizing and extending 549
- language specific, table of 539
- metrics for audit violations 526
- output for 528
- printing results 536
- running from command line 547
- running in Together 525
- saving results 529
- See also* metrics
- sets of, creating and using 538
- sorting results 528
- UI Builder 345

authentication and authorization

- See* security

AWT 470

B

- bar graph (metrics results) 533
- BEA
 - See* WebLogic
- bookmarks
 - classifying global 249
 - deleting 250
 - editing 247
 - global 248
 - local 250
 - reordering 249
- bootstrap loader 498
- BorderLayout 318
- BoxLayout 319
- breakpoints 251
 - attribute, modifying 293
 - debugging 290
 - properties, modifying 292
 - setting 291
- browse symbol 257
- business process diagram 131
 - elements 200
 - notation 201

C

- C++ 795
 - documentation 807
 - header file 803
 - implementation file 803

- SHIFT+Click to open files 803
- C++ Definition Doc-Comment 807
- C++ projects
 - compile and make utilities 802
 - file extensions 801
 - include directives 795
 - macro definitions 795
 - member documentation 807
 - preinclude file 797
- Cactus 477
- CardLayout 318
- checkbox menu items 339
- child element 261
- Choose Pattern command 518
- Choose Pattern dialog 504, 512, 517
- class diagram 131, 168
 - attributes and operations, rearranging 175
 - compartment controls, setting 175
 - elements 169
 - inner class, defining 172
 - members and properties, editing 174
 - search paths, showing classes of 172
 - shortcuts 825
- class diagrams
 - JavaBeans, showing 172
 - links, creating 173
 - physical and logical 170
- class shortcuts 825
- classes
 - UI 307
 - creating new 308
- ClearCase
 - advanced version control 375
 - interface, setting up 371
 - operation semantics 373
 - results and errors, reporting 377
 - Unified Change Management 377
- clipboard assertion 489
- CMP
 - EJB 605, 617
- code completion feature 253
- Code generation 114
- code sense 244, 253
 - advanced 254
- code template
 - editing 507
 - properties 504
- code templates 503
 - browsing code 506
 - browsing using Directory tab 506
 - custom 508
 - custom, creating manually 510
 - custom, creating using the expert 508
 - default properties 505
 - displaying custom names 511
 - editing 507
 - grouping 510
 - macros, user-defined 512
 - patterns, using with 517
 - properties of 504
 - source code, formatting 508
 - using macros 505
- collaboration diagram 175
 - elements 177
 - messages 180
 - sequence diagram, converting to 178
- collection suites 483
- command line
 - parameters 811

- syntax 811
- compile-make-run
 - alternative compiler, using 284
 - compile and make tools, executing 282
 - makefile, generating a 125
 - programs, running 285
 - standard compiler, using 282
- component diagram 131, 190
 - elements 190
 - techniques for 191
- components 49
 - menu bar 331
 - showing in container 332
 - popup menu 335
 - UI 319
 - Object Gallery 308
 - user interface 313
- config file 739, 741
 - referencing configuration properties as macros 819
 - sample 746
- configuration files 739
 - language specific 786
- configuring
 - overview of levels 74
 - See* options
 - UI Builder 306
- container transaction 601
 - creating diagram element 602
 - diagram element 602
 - linking to a method 603
- container transaction elements
 - definition 601
- container-managed relationship 583
- context help
 - using 259
- Creation Scenario field 98
- customizing Together
 - advanced 739
 - toolbox 347
- CVS 358

D

- database modeling 118
 - activating in Together 118

- deadlock state 297
- debugging
 - arrays, evaluating 294
 - breakpoints 290
 - breakpoints, setting 291
 - monitors 297
 - program execution, controlling 289
 - remote debugging, starting 298
 - remote process, attaching 297
 - watches, adding 295
- declarative security role 660
- deep parsing 785
- def file 766
- default level 74
- default target
 - setting for a test 495
- defDocComment module 808
- deployment
 - view in modeling 130
- deployment descriptor 556, 557
 - CheckMustUnderstands 708
 - encoding 700
 - limitations 649
 - message 708
 - resource adapter diagram 655
- deployment diagram 131, 192
 - elements 192
 - techniques for 193
- deployment expert
 - resource adapter diagram 656
 - Web services, running for 720
 - WebLogic 720, 722, 723
- design
 - view in modeling 130
- design elements, standalone 158
- Designer pane
 - Menu Designer view 304, 312
 - UI Designer view 304, 310
- diagram level 74
- diagrams 153
 - annotating 157
 - archived file, restore from 115
 - autodependency links, filtering out 142
 - bidirectional links, converting 146
 - cloning 135

- configuring options 136
- creating custom types 740
- creating using main menu 134
- default of a project 96
- dependency links 142
- drag-and-drop support 145
- editing properties 151
- element size 138
- elements and links 143
- elements, copying and cloning 145
- elements, drawing 136, 138
- elements, moving and copying 144
- elements, resizing 145
- elements, selecting 143
- graphics, improving 136
- grid, using the 137
- hyperlinking 153
- hyperlinking feature 135
- images 159
- layout 148
- layout, automated 149
- layout, manual 149
- link to self, drawing 142
- links, drawing 140, 141
- links, labeling 147
- list of supported 130
- multiple elements, adding 139
- opening and closing 150
- overview for creating 133
- printing 160
- renaming 135
- right-click menus 144
- 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 137
- zooming view 149
- disabled 263
- distribution graph (metrics results) 535
- DocGen
 - See* documentation generator
- documentation generation
 - automating 390, 815
 - for user interface components 346
 - options 392
 - RTF options 386
 - starting 381
- documentation generator 384, 395
- documentation template 384, 395
 - body sections 396
 - calls to stock sections 397, 408
 - calls to template sections 397, 409, 427
 - controls 411
 - current model element 397
 - data controls 415
 - designer 395
 - DG functions 438
 - DG variables 435, 436
 - element iterator scopes 405
 - element iterators 397, 404
 - element property iterators 397, 406
 - enabling condition 403
 - folder sections 397, 407
 - footers 411
 - formula controls 417
 - frameset templates 423
 - headers 411
 - hypertext links 420, 429

- image mapping 430
- images 413
- JavaDoc link references 431
- labels 413
- metamodel 398
- panels 413
- root object metatype 397
- static sections 397, 411
- stock sections 408
- text controls 418
- variables 417

documentation templates

- DG functions 438

drag-and-drop support 145

E

early access folder 761

editing 243

- bookmarks 247
- breakpoints 251
- browse symbol 257
- code completion feature 253
- code templates 507
- commands, edit menu 246
- commands, editor right-click menu 246
- configuring the editor 243
- deleting bookmarks 250
- editor features, advanced 252
- Editor pane, showing and hiding 267
- external editor 269
- help database, creating 258
- import assistant 255
- inplace, how to enable 746
- no open project 267
- open project 267
- parameter tooltip feature 252
- rectangular blocks 265
- snippets 255
- split pane 265
- surround with... feature 252
- Tag Library Helper 264
- UI classes 307
- UI component properties 314, 316
- using the editor 245
- XML editor 259

- XML, JSP, and HTML files 263

editor

- advanced features 252
- override/implement methods 252
- surround with...feature 252
- toggle comment 252
- watches, adding 295

editor, built-in 243

editor, external 269

- configuring 269
- using 269

EJB 561

- assembler diagram relates to J2EE deployment expert, how 598
- Cactus support in testing framework 477
- component synchronization options 563
- configuring using EJB inspectors 569
- Copy options 563
- correction 597
- creating a project using existing EJB code 569
- creating using the Object Gallery dialog 568
- entity 566
- message-driven 566
- resource adapter 651
- session 566
- shortcuts, creating 594
- show options 563
- verification 597

EJB assembler diagram 556

- and J2EE Deployment Expert 598
- creating 592
- display 595
- properties 593
- security support 558
- toolbar elements 593

EJB assembler diagrams 591

EJB components

- class diagram toolbar buttons 566
- compiling 581
- creating 561
- customizing default code 589
- editing 563
- entity beans 582
- Inspectors 569

- local interfaces 566
- message-driven beans 586
- methods 575
- Object Gallery 566
- properties 561
- references 571
- session beans 585
- sharing interfaces 588
- EJB deployment
 - properties 572
- EJB modules 591
 - creating 592
 - creating a project using existing EJB code 569
 - creating EJBs 562
 - security support 659
 - visual design elements 566, 593
- EJB modules
 - See also* EJB 591
- EJB pattern 567
- EJB properties
 - diagram element 604, 616
 - using 605, 617
- EJB Properties element 605
- EJB reference
 - and reference attribute 608
 - creating diagram elements 607
 - diagram element 604
 - EJB resource-environment reference, creating 610
- EJB reference element 604
- EJB references
 - creating 605
 - environment references, creating 610
 - environment variable, changing type 611
 - resource references, creating 609
 - supported types 606
- EJB shortcuts 594
- element
 - options, defining 745
 - shape of, defining 745
 - unique name 745
- element, Together 37
- encoding 700
- enterprise application
 - creating using Object Gallery 638, 646
 - security support 666
- enterprise application diagram 556
 - and J2EE deployment expert 642
 - creating 638, 646
 - elements 639
 - importing files 641
 - shortcuts, creating 639
- enterprise applications 637
 - visual assembly 637
- entity bean
 - See* entity EJB 582
- entity beans
 - default code 582
 - fields 574
 - importing from database 567
 - persistence 570, 583
 - primary key 571, 583
- entity EJB 566
 - code template 506
 - references 571
- entity relationship diagram 131, 195
 - attributes 198
 - cardinality 200
 - entities 198
 - logical and physical view 197
 - notation 196
 - relationship links 199
- event handlers 327
- event sheet diagram 217

- Explorer pane 42
 - Components tab 49
 - Copying and pasting 51
 - Diagrams tab 47
 - Directory tab 43
 - Favorites tab 47
 - finding locations 52
 - Model tab 44
 - Modules tab 48
 - opening diagrams and files 45
 - quick project access 51
 - Server tab 44
 - Test tab 50
 - toolbars 46
 - UI Builder tab 49
 - using right-click menus 51
 - XML tab 50

- export 117
 - database 119
 - DDL 119
 - IDL 121
 - JDBC supported types 118
 - XMI 123
- extension 512
 - codegen default 512
 - custom 513
 - source 512

F

- filter
 - UI component properties 314
- filter mappings 631
- filters 631
 - creating 631
 - deploying 632
- find and replace 272
- FlowLayout 318
- focus pane 37
- frames 295
 - using 297

G

- generating documentation
 - See* documentation generator
- getter 762

- go-to-line search 277
- graphics
 - improving in diagrams 136
- GridBagLayout 318
- GridLayout 318
- GUI Builder. *See* UI Builder

H

- Hello World example 758
- HTML editor 264
- HTML files
 - editing 263
 - viewing 264
- HttpUnit 477
- hyperlinking 153
 - browsing 156
 - diagrams and elements 153

I

- IBM
 - See* WebSphere
 - Test Registry 727

- icons
 - in menus 343
- IDE 755, 756
- IDE Profile (UI Builder option)
 - choosing 306
 - options 306
- IdeScript interface 757
- IdeStartup interface 757
- implementation
 - view in modeling 130
- import 117
 - database 120
 - existing files to an enterprise application diagram 641
 - JDBC supported types 118
 - JUnit tests 483
 - XMI 122
- import assistant 255
- inline operation 803
- Inspector
 - EJB component 569
- inspector 37, 56
 - EJB 569

- UI components 314, 316
 - web application diagram 639, 648
 - inspectors 787
 - C# 789
 - C++ 792
 - CORBA IDL 790
 - test suite 495
 - Visual Basic 787, 788
 - instances, running multiple 107
 - Integrated Development Environment
 - See* IDE
 - interface
 - sharing home and remote 588
 - interface, definition of 757
 - items, menu. *See* menus, designing, new menu items 337
- J**
- J2EE Connector Architecture 651
 - J2EE deployment 669
 - Command Line File 687
 - fast track 676
 - full featured 676
 - required installations 671
 - transitioning among application servers 696
 - J2EE Deployment Expert 598, 670
 - BEA WebLogic Run-Time Deploy 689
 - common server options 679
 - config files 676
 - deployment diagrams 674
 - IBM WebSphere 3.5 691
 - IBM WebSphere 4.0 695
 - process options 679
 - simple JSP client 685
 - Starting application servers 672
 - Sun EE Deployment 688
 - supported application servers 670
 - Verify / Correct Sources 684
 - J2EE Module Import command 115
 - J2EE support 553
 - See also* references support
 - J2EE-supported diagrams 554
 - application client diagram 557
 - EJB assembler diagram 556
 - enterprise application diagram 556
 - resource adapter diagram 557
 - Taglib diagram 557
 - web application diagram 556
 - JAR 645
 - JSP 624
 - creating new 626
 - debugging 626
 - Web application diagram 624
 - JSP editor 263
 - JSP files
 - editing 263
 - JUnit 476
 - importing tests 483
 - See also* testing framework
 - JUnitX 477
- K**
- keyboard shortcuts 821
 - designer 824
 - editor/compiler/debugger 826
 - for the editor 244
 - main menu 821
 - keyboard shortcuts, defining 338
 - Kiviat graph (metrics results) 532
- L**
- launcher
 - Together.exe, invoking 815

- layout managers 317
 - BorderLayout* 317, 318
 - BoxLayout* 319
 - CardLayout* 317, 318
 - classes 317
 - FlowLayout* 317, 318
 - GridBagLayout* 317, 318
 - GridLayout* 317, 318
 - learning about 325
 - null* 319
 - OverlayLayout* 319
- listeners 632
 - and Web applications 633
 - creating 633
 - event types 632
- Live Source 783
- LiveSource 111, 121, 122
- look and feel 327

M

- macros
 - for code templates 512
 - system 816
 - template 820
- main
 - menu 40
 - window 37
- main class, Together 813
- makefile
 - generating 125
- managers, layout 317
- manifest file
 - See* Manifest.mf file, def file
- Manifest.mf file 766
- menu bar
 - creating 331
 - showing in container 332
- menus
 - accelerators 338
 - colors 344
 - component identifiers
 - renaming 344
 - designing 304, 312, 330, 345
 - new menu items 337, 338
 - visually 336

- fonts 344
- keyboard shortcuts 338
- menu bar (UI Builder) 331
- menu items
 - adding icons 343
 - checkbox 338, 339
 - enabling, disabling 338, 342
 - moving 341
 - properties 342
 - radio-button 338, 340
 - separators 341
 - showing, hiding 338, 342
- menu text 338
- mnemonics 338
- nested 341
- popup (UI Builder) 335
- message-driven beans
 - default code 587
 - deployment properties 587
 - project properties 586
- message-driven EJB 566
- method 762
- metrics
 - comparing results 531
 - graphs 532
 - reports 529
 - saving and loading results 530
 - See also* audits and metrics
 - UI Builder 345
- mnemonics, in menus 338
- model
 - existing code, create from 111, 115
- modeling
 - deployment view 130
 - design view 130
 - implementation view 130
 - list of supported diagrams 131
 - overview 129
 - patterns 130
 - process view 130
 - See also* diagrams
 - See* UML modeling
 - use case view 130
- module
 - definition 757

- testing framework 471
- modules 48
 - activating and deactivating 73
 - compiling 771
 - declaring 766
 - declaring in `def` 769
 - declaring in `Manifest.mf` 766
 - evaluating results 772
 - extending Together 759
 - guidelines for developing 761
 - Hello World example 758
 - interfaces implemented by 760
 - locating 757
 - naming conventions 762
 - registering 758
 - running 761
 - running at start-up 815
 - source code 765
 - starting 758
 - storing compiled class 772
 - tab, description of contents 761
 - troubleshooting 773
 - tutorial for writing and deploying 764
 - types of 760
 - viewing 760
- monitors 297

N

- New Project dialog 100
- New Project Expert 98
- non-deadlock state 297
- null* layout manager 319

O

- Object Gallery
 - EJB, creating a 566
 - User Interface 308
- operator site 732
- options
 - checkboxes, value of 79
 - context-sensitive help 79
 - reference guide 80
 - resizing the Options dialog 79
 - UI Builder 306
 - viewing and editing 78

- Options dialog 753
- OverlayLayout 319

P

- package diagram
 - shortcuts 825
- package diagrams 170
 - working with 171
- package prefix
 - specifying in a project 99
- pane 37
 - expand 39
 - hiding title bars 39
 - resize 39
 - undocking 39
 - view or hide 38
- parameter function 745
- parameter tooltip feature 252
- parameters
 - command line 811
 - Together main class 813
 - virtual machine preferences 815
- Parser blueprints 114
 - defining 114
- pattern
 - resource adapter 653
- patterns 513
 - and modeling 130
 - behavior 515
 - choosing for member 520
 - classes and links, creating 518
 - code templates, using with 517
 - creating 515
 - creating members by 519
 - descriptions, providing 517
 - elements, generated 514
 - languages supported 514
 - refactoring with 521
 - user interface 308
- peer element 261
- Perforce 369
- performance tuning
 - large projects 109
- popup menu
 - creating 335

- displaying at runtime 335
- process
 - view in modeling 130
- profile
 - IDE 306
- programmatically security role 660
- project
 - creating a 97
 - creating using existing EJB code 569
 - default diagram 96
 - multiple instances, running 107
 - New Project dialog, using 100
 - New Project Expert, using 98
 - performance tuning 109
 - properties, editing 101
 - resource options, setting 105
 - resources, adding 103
 - resources, removing 105
 - root directory, primary 96
 - tips for large projects 108
 - tpx file 96
 - understanding basics 95
 - version control, setting-up 108
 - views with referenced content 109
- project level 74
- project properties
 - application server 561
- properties
 - editors, UI components 316
 - exposure levels, UI components 314
 - of UI components 314
- properties file 740
- PVCS 370
- PVCS Version Manager 370

Q

- QA module 549
 - extending 549
 - See also* audits and metrics
- query-based search 275

R

- radio-button menu items 340
- Read-Write Interface
 - See* RWI

- real-time modeling 205
 - concurrent state 217
 - event sheet diagram 217
 - functional requirement 211
 - interaction diagram 219
 - maximum response time 214
 - minimum response time 213
 - object interaction group 220
 - ops simulation unit 224
 - process delay link 222
 - quality of service requirement 211
 - rate of occurrence 214
 - receiver polling reply message 221
 - sender waiting reply message 221
 - simple asynchronous messages 220
 - subsystems 207
 - system architecture diagram 214
 - system context diagrams 210
 - system requirements and architecture 207
 - time stamp message 222
 - trtelement 211
 - use case diagrams 208
- recorder 487
- recorder, for testing 470
- refactoring
 - encapsulating attributes 459
 - extracting interfaces, superclasses 460
 - extracting operations 461
 - move class, interface 456
 - patterns 521
 - pull up operation, attribute, member 456
 - push down operation, attribute, member 456
 - renaming 458
- Reference attribute
 - Inspector 606
- reference attribute 604
 - in EJB client 605
 - types 606
- references support
 - security support 558
- remote debugging, starting 298
- resource adapter
 - connection classes 653
 - Connector Architecture 651
- resource adapter diagram 557

- creating 652
- creating by a pattern 653
- elements 652
- security support 558
- using 655
- resource adapters 651
- resource files 740
- resources
 - adding to a project 103
 - removing from a project 105
 - setting options 105
- reverse engineering 111
 - into an existing project 113
 - J2EE archive files 115
 - J2EE Module Import command 115
- right-click menu 37
 - editor commands 246
 - in diagrams 144
- robustness analysis diagram 131, 201
 - elements 202
 - key elements and properties 203
- roles 68
 - Business Modeler 68
 - Designer 68
 - Developer 68
 - Programmer 68
- root directory
 - of a project 96
- run 285
- RWI 756
 - See* Read-Write Interface
- RWI properties 398

S

- sample folder 760
- schemes 244
- SCI 755, 756

- script, definition of 757
- search facilities 271
 - diagrams, searching 274
 - find and replace 272
 - find and replace in files 272
 - go to line 277
 - query-based search 275
 - usages, searching for 278
- search for usages 278
- Security 657
 - method permission 659
 - EJB modules 659
 - enterprise applications 666
 - method permissions 660
 - principals 659
 - resource adapters 666
 - resource authentication mechanisms 667
 - security constraints 664
 - security role 658
 - security role reference 661
 - security role references 659
 - Web applications 663
 - Web resource collection 664
- security
 - authentication and authorization 664
 - declarative security role 660
 - EJB modules 659
 - enterprise application 666
 - J2EE platform support 657
 - programmatic security role 660
 - roles 658
 - Web applications, in 663
- security role 663
- See* keyboard shortcuts 821
- separators, menu 341
- sequence diagram 131, 175

- collaboration diagram, converting to 178
- elements 177
- generating from operation 178
- lifelines 183
- messages 180
- Options dialog 179
- source code, generating 184
- Server Explorer
 - starting servers from 672
- servlet 622
 - Cactus support in testing framework 477
 - creating 622
 - creating using Object Gallery 623
 - running and debugging 624
 - Web application 624
- servlets
 - creating using pattern 622
- session bean
 - See session EJB*
- session beans
 - default code 585
 - stateful and stateless 585
- session EJB 566, 585
- setGraphicObject function 745
- setLayoutConstraints function 745
- setter 762
- shortcuts
 - creating 143
 - EJB, creating for 594
 - in the Explorer 47
 - keyboard, in menu design 338
- Show options
 - adding 753
 - display text, changing 752
 - removing 753
- snippets 255
- Source Code Control interface 371
- Source Code Interface
 - See Source Code Interface*
- Star Team 370
- startup module 769
- statechart diagram 131, 184
 - elements 185
 - entry/exit actions, specifying 186
 - internal transition, creating 186
 - self-transitions, drawing 185
 - shortcuts 825
 - substates, nested 186
 - transition sources and targets, showing multiple 187
- stereotype
 - UML diagram support of 166
- stylesheet
 - HTML reports from testing framework 496
- submenu. *See* menus, nested
- Swing 470
- syntax
 - UI Builder
 - menu definition 338
- system context diagrams 210
- system folder 760
- system macros 816

T

- tag handler 627
 - properties 629
- tag library
 - See TagLib*
- TagLib 627
 - creating 629
- taglib
 - Cactus support in testing framework 477
 - Tag Library Helper 264
- TagLib diagram 628
 - elements 628
 - properties 628
- taglib diagram 557
- template
 - default properties 505
 - using macros 505
- template macros 820
- templates
 - See* code templates
 - test cases, creating 483
- test assets, definition of 470
- test case
 - See* testing framework
- test cases
 - writing 481
- test plan, definition of 470

- test results, definition of 470
- test server
 - known problems 502
 - network security 501
 - running 498, 500
 - setting-up bootstrap loader 498
 - system requirements 498
 - troubleshooting 501
- test step
 - of Junit test 481
- testing framework 469
 - activating 471
 - bootstrap loader 498
 - CashSales test project 471
 - collection suites 483
 - default target, setting 495
 - JUnit step options 476
 - JUnit tests, importing 483
 - JUnit tests, running 485
 - options, configuring 474
 - options, recording 477
 - overview 470
 - process description 470
 - recording options 489
 - report of test results 496
 - stopping tests 485
 - stubs, generating 480
 - stylesheet 496
 - templates, using to create test cases 483
 - test cases, writing 481
 - test collection, creating 487
 - test project, creating 472
 - test server, setting up 498
 - test suite options 476
 - test suite, creating 482
 - Tests tab 471
 - unit test builder 484
 - unit test builder options 476

- unit tests, developing 479
- visual script, editing 490
- visual script, recording 487
- visual step options 475
- Tests tab 471
- text (in menus) 338
- The Type Mapping Properties tabs describes types that have no standard representation in XSD. This tab contains Web service
 - Type Mapping Properties 709
- threads 295
 - using 297
- Together diagrams
 - See UML Extension Diagrams
- Together.exe launcher 815
- Together_starter command 812
- Tomcat 624, 720
- toolbox
 - customizing 347
 - UI Builder 313
- toolbox shortcuts 825
- toolbox, Designer 37
- tpr file 96
- transaction attribute
 - See container transaction
- troubleshooting
 - modules 773
- trtelement 211

U

- UDDI Browser 725
 - functionality 725
 - object properties 729
 - operator 725
 - options 727
 - publish 731
 - removing published objects 734
 - search 726

- search options 726
 - search results 728
 - searching for Web objects 726
 - templates 734
 - Web service, publishing 731
 - WSDL files 729
 - UI Builder 303
 - activating 305
 - adding UI components 313
 - audits and metrics 345
 - component properties 314, 316
 - components
 - alignment 327
 - appearance order 328
 - spacing 327
 - containers 319
 - adding components to 319
 - deactivating 305
 - documentation generation 346
 - editable classes
 - creating 308
 - definition of 307
 - event handlers 327
 - layout managers 317
 - Menu Designer view 304, 312
 - menu items 337, 338
 - menus
 - designing 330
 - options 306
 - overview 303
 - tips and tricks 327
 - toolbox 313, 347
 - UI Designer view 304, 310
 - UML extension diagrams 195
 - business process diagram 200
 - entity relationship diagram 195
 - robustness analysis diagram 201
 - UML modeling
 - activity diagram 188
 - class diagram 168
 - collaboration diagram 175
 - component diagram 190
 - deployment diagram 192
 - list of supported diagrams 165
 - notation in diagrams 165
 - sequence diagram 175
 - statechart diagram 184
 - stereotypes 165
 - use case diagram 166
 - Unified Change Management 377
 - unit test builder 476, 484
 - unit testing 470
 - See* testing framework
 - unit tests, developing 479
 - unittestbuilderconfig.xml file 484
 - use case
 - view in modeling 130
 - use case diagram 131, 166
 - browse-through sequences 167
 - elements 166
 - extension points 168
 - user interface
 - advanced customization 739
 - designing. *See* UI Builder 303
 - diagram configuration file 741
 - icons and icon references, creating 741
 - modifying elements of a visual test 490
 - toolbar icons, defining 742
 - treeview icons, defining 742
 - viewmaps, defining 744
 - Using 569
- ## V
- variables
 - structured context, displaying 294
 - version control 355
 - configuring Together for 357
 - Continuous/CM support 369
 - interacting with 365
 - multi-user development support 355
 - overview 356
 - Perforce support 369
 - product-specific notes 366
 - projects, enabling for existing 364
 - projects, enabling for new 363
 - PVCS support 366
 - PVCS Version Manager support 370
 - SCC-compliant version system 361
 - setting-up for projects 108
 - Star Team support 370

- VCS 358, 359
- view management
 - customizing Show options 752
- virtual machine 815
- visual script
 - recording 487
- visual tests 470
 - options 475
 - results 492
 - running 491
 - See testing framework

W

- watches
 - display format, changing 296
 - display range, changing 296
 - values, changing 296
- Web application
 - security 664
 - security support 663
- Web application diagram 613
 - and J2EE Deployment Expert 634
 - creating 614
 - elements 615
 - properties 616
 - shortcuts 619
- web application diagram 556
 - security support 558
- web applications 613
- Web Service 705
- Web service 705, 719
 - Apache SOAP 706
 - client for Apache SOAP, generating 716
 - client with WSDL file, generating 715
 - client, generating for BEA WebLogic 716

- creating 706
 - creating from a class 706
 - creating from an EJB 712
 - creating using patterns 713
- deployment expert 720
 - deployment expert for Apache SOAP 720
 - deployment expert for WebSphere 723
- deployment properties 719
- exposing methods 713
- generating a proxy client from a WSDL file 715
- generating clients 715
 - generating clients for Apache SOAP 716
 - generating clients for BEA WebLogic 717
 - generating proxy clients using patterns 716
 - generating type serializers/deserializers 716
- Generating WSDL files 714
- Message-style 712
- mustUnderstand 708
- properties 706
 - properties, server-specific 708
 - properties, type mapping 708, 709
- publishing 731
- relationship to the Internet 731
- RPC-style 712
- scope 709
- Server-specific Properties 708
- target server 706
- Web service Inspector
 - Properties tab 706
- Web services
 - publishing 731

Web Services Description Language

See WSDL

WebLogic 705

message-style Web service 712

RPC-style Web service 712

WebLogic Inspector tabs 578

CMP entity EJBs 579

entity EJBs 579

message-driven EJBs 581

session EJBs 577, 578

WebLogic 6.0 /6.1 578

WebLogic 7.0 578

WebSphere 705

deployment expert 723, 724

workspaces

and roles 67

default workspace 65

managing 66

roles 67

saving 67

WSDL 707, 725

default namespace settings 707

types 709

WYSIWYG 304

X

XML 655

XML editor 259

attribute, adding 261

child element, adding 261

peer element, adding 261

unit test builder configuration file 485

XML file structure 261, 263

XML file, creating an 260

XML file, opening an 261

Z

zoom shortcuts 824