User's Guide

VERSION 5.5

Table of Contents

Version 5.5

Thank you for choosing Together®- the exclusive Platform plus Building Blocks software development infrastructure solution for the 21st century.

About TogetherSoft and Together

If this is your first contact with Together® products or TogetherSoft Corporation, this section provides a brief "10,000-foot flyover" to help you get acquainted.

TogetherSoft Corporation - Dedicated to improving the ways people work together™

TogetherSoft Corporation is the adaptive business-process automation™ company. TogetherSoft's software and services enable enterprises to develop better assets faster by providing proven and predictable ways to manage change, mitigate risk, and deliver frequent, tangible, team-driven results.

Together ControlCenter

TogetherSoft's flagship offering, Together® ControlCenter™, delivers adaptive business-process automation for teams building software solutions. Together ControlCenter brings your e-solutions team together, allowing business users, developers, and operations to collaborate using a common language, diagrams, and software. Together ControlCenter enhances productivity and process management in critical areas: automating mundane business processes (e.g. adaptive documentation generation); automating tedious and error-prone business processes (e.g. work required to adapt and deploy an application on an application server); and automating expert-level insights with guidance on how to adapt and apply those insights correctly (e.g. patent-pending expert-level pattern technology).

For more information, see Introducing Together.

Together's Documentation

Available formats

Together provides a choice of electronic Help documentation formats.

JavaHelp(tm) Installed with all products. Provides Table of Contents, Index, and Full-text Search. Requires the Sun JRE and JavaHelp runtime (both installed with Together).*

Adobe Acrobat (PDF) Available for download at **www.togethercommunity.com**. Requires Adobe Acrobat Reader, available free at **www.adobe.com**. Same content as Together Help documentation for those who want to print hard copy. (No internal hyperlinking.)

Together's Help is also available on the Web. To access it, either go to Together home page at **www.togethercsoft.com**, or select **Help | On the Web** of the main menu and click on an appropriate item from the drop-down list:
www.togethersoft.com - main Together home page
www.togethersoft.com/support - Technical Support info
http://www.togethersoft.com/order - Customer Service
http://www.togethercommunity.com/ - Together Community site
http://www.togethercommunity.com/docs/ - Application Help
http://www.togethercommunity.com/contriblist.pl?display=module - Download Building Block (download the free Together Community Edition 5.0)

Together's Help is also available online at **www.togethercommunity.com/docs/**. Updates and corrections are posted to this URL between released.

** (**Note:** URLs for Web resources are provided as text only, as JavaHelp does not currently support hyperlinking to external targets.)*

Help documentation volumes

Help documentation is arranged in three volumes, each with a particular scope.

Getting Started with Together is a startup guide intended primarily for reviewers, evaluators, and other new Together users who are installing and setting up the product for the first time. This volume also includes copies of the relevant license agreements.

User's Guide covers general introductory issues and provides in-depth explanations about how to accomplish your work using Together. Besides, it includes technical, system, UI, and other reference information related to Together's architecture, User Interface, and API.

Context help, or F1 help, delivers the context-sensitive help information for config options and dialogs. When a dialog is open, press F1 or click *Help* button on the dialog to view the appropriate Help topic.

Besides the main helpset, Together provides separate documentation for the modules that implement some integration or interoperability with the third-party software products.

Other documentation

Help documentation is your main source of information, but it is not the only documentation provided. In addition to Help, you will find:

Readme file: Resides in same directory as the installer program (before you install Together from CD) and in the root directory of your *Together* installation. It contains pre-installation information and tips, and/or late-breaking information not yet incorporated into documentation.

What's New: The file `whats_new.html` is written to the root directory of your installation and contains information about new features and enhancements for the current release. There's also a version history file in the same location if you're interested.

API Documentation: JavaDoc(tm) technical reference for the Together Open API. To access the main index file, open `%TOGETHER_HOME$/doc/api/index.html`

Comments in files: Files in the installation contain comments that you will find useful in the specific context. These include:

- Source files for modules
- Example projects (source and diagrams)
- All the configuration properties files (./config/*.config, *.properties)
- Example batch and command files (*.bat, *.cmd, *,sh) for launching Together.

Let's get started

People generally tell us that Together is quite easy to set up and begin using. We try to put the basics right up front, making them as easy to find and as intuitive to use as possible. But don't be deceived by the apparent simplicity. There is tremendous capability built into Together, powerful functionality just off to one side waiting until you're ready to unleash it.

Copyright and Trademark Notice

Other Acknowledgments

The "Coad Modeling Components" that ship with Together are based upon the components described in the book: Java Modeling in Color with UML: Enterprise Components and Process, by Peter Coad, Eric Lefebvre, and Jeff De Luca, Prentice Hall, 1999, ISBN 013011510X, and are included with the kind permission of the publisher.

Portions of the Together documentation are derived from concepts and terminology published in The Unified Modeling Language User Guide by Booch, Rumbaugh, and Jacobson. Addison-Wesley, 1998. ISBN 0-201-57168-4.

Other portions of documentation may include material that is Copyright ©1998-99 Object Management Group, Inc. (OMG), and is reproduced by permission.

Portions of on-screen descriptions of Metrics and Audits derived from documentation published online by Sun Microsystems at **http://java.sun.com/docs/codeconv/**.

Together includes software, which is Copyright ©1989, 1991 Free Software Foundation, Inc. All Rights Reserved. That software is subject to the terms of the GNU General Public License available at **www.gnu.org**. You must accept the terms of that agreement to use that software. See copyright statement and disclaimer below.

Together includes Transformational Patterns as developed in the TROOP (EP 27291, Transformation of Object Oriented design using design Patterns) project funded by the European Commission.

Together includes software, which is Copyright ©1996;1997 Original Reusable Objects, Inc. All Rights Reserved. That software is subject to the terms of the Original Reusable Objects OROMatcher License available at www.savarese.org/oro. You must accept the terms of that agreement to use that software.

Together includes software developed by the Apache Software Foundation (http://www.apache.org/), which is Copyright ©1999 The Apache Software Foundation. All rights reserved. See copyright statement and disclaimer below.

Sun J2EE Patterns

The Sun J2EE Patterns Catalog from Sun's Java CenterSM consulting organization is listed in its entirety within this product and will be implemented in future product releases.

The Sun J2EE Patterns are used with permission from the book "Core J2EE Patterns" by Deepak Alur, John Crupi, and Danny Malks, published by Sun Microsystems Press/Prentice Hall.Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303. All rights reserved. SUN PROVIDES EACH J2EE PATTERN "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

GNU Copyright Statement and Disclaimer for bundled CVS version control and gnuMake software

The following information is provided in compliance with the GNU Public License Agreement:

The CVS version control and gnuMake software bundled with some Together editions is Copyright © 1989-1999 Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111, USA

NO WARRANTY

> BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

See also: GNU General Public License Agreement

Apache Copyright Statement and Disclaimer for bundled Apache Tomcat server

The following information is provided in compliance with The Apache Software License, Version 1.1

Copyright ©1999 The Apache Software Foundation. All rights reserved.

> THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://**www.apache.org**.

User's Guide

About this volume

The User's Guide provides in-depth coverage of Together's features and generally explains how to do different things with the product.

Part 1 of the User's Guide contains introductory information on the architectural concept and overview of Together's features. The other sections deal with the issues that need to be approached first, before embarking to work with Together. Quick Tour chapter takes you through the main elements of the user interface. Chapter Configuring Together contains information on the common customization issues and the use of the Options dialog. Chapter Command Line Operations and macros provides reference information on command line parameters, various macros, and keyboard shortcuts. You can also find basic materials on Projects and Project Management, Information Import (relevant when setting up a project), integration with a Version Control system, and View Management.

Part 2 of the User's Guide describes working with Together, adhering to the development lifecycle and major feature categories. Thus, the volume includes chapters that cover various modeling issues, usage of diagrams, software and e-commerce development features. Special chapter is devoted to the currently supported languages. Extensibility and Advanced Customization section covers the most subtle issues of the system and advanced UI customization, and enhancing Together by means of the Building Blocks. Web Services chapter provides information about creation and deployment of the web services.

Part 1. Introductory Topics

Introducing Together

Introducing Together

The Together product line consists of two products:

Together ControlCenter is for building enterprise-level software. Together ControlCenter is the Model-Build-Deploy (MBD) Platform... comprehensive and complete encompassing *model-EJB-pattern-edit-compile-debug-version-doc-metric/ audit-provision-assemble-deploy-run*.

Together Solo is for building software on a smaller scale, without the need for Together ControlCenter's exclusives features. Together Solo spans *model-pattern-edit-compile-debug-version-doc*. Together Solo includes comprehensive UML-diagram editors (with auto-layout and snap-to-grid for class diagrams), syntax-aware customizable programming editor, GoF patterns, and simultaneous round-trip engineering for Java, plus C++ and IDL.

Get the full story on features at the TogetherSoft website: **www.togethersoft.com/together/**.

Exclusive "Platform plus Building Blocks" Architecture

The "hidden secret" behind TogetherSoft™ technology is that, instead of acquiring different products from different companies, gluing code together, and shoving the result into the marketplace (yes, some companies actually do such things!), TogetherSoft builds its products from the ground up using **The TogetherSoft Platform** plus **Building Blocks** to deliver a comprehensive, team-enabling, enterprise software development environment that fits naturally and comfortably into your company infrastructure.

The TogetherSoft Platform

The Platform is the "heart and soul" of Together that delivers the key core services:

Diagram engine: supports creation and editing of all the major UML diagrams, along with several other commonly used diagrams like Entity Relationship (for data modeling) and Business Process, plus some innovative and highly useful new diagram types such as our exclusive EJB Assembler, Web Application, Enterprise Application, and XML Structure diagrams. The Diagram engine provides full control over diagram editing, storage representation, and documentation. Speaking of storage representation: it's done with simple ASCII text so your diagram files are easily kept under source control just like source code. And if all this isn't enough, using Together's open Java™ API, it's possible to define new, custom diagram types of your own which the Diagram engine will automatically support.

Simultaneous round-trip engines: special parsers designed for immediate synchronization between design diagrams and implementation code.

Simultaneous round-trip engineering means that your UML classes are always synchronized to the source code that implements them. Change something in a Class diagram and the relevant source code updates immediately. Change the code and the visual model updates to stay in sync. There's no intermediate repository, no batch code generation, and no risk of losing code.

Launch-Catch-Navigate engine: delivers better integration across various source-code products. For example, you can launch the compiler, catch any errors, and navigate from each error message to the corresponding diagram element, line of source code, and property editor tab. Speaking of compilers, Together is pre-configured to work with Sun's Java2™ SKD, invoking its compiler with a simple right-click operation. If you want to use another Java compiler, or if you work with C++, you can easily configure Together so that it launches the compiler of your choice ... ;again, with just a speedmenu click.

Version control integration: Together is not a version control system (VCS). Rather, the Platform's architecture is designed to tightly integrate with your preferred versioning product. The version control integration system supports check-in, check out, and other versioning functions... via CVS (included with Together and ready-configured) or the SCC interface (supported by every major Windows version control system).

Multiple OS compatibility: It's worth mentioning here that Together is implemented in Java™. It was one of the first true Java apps to deliver breadth, power, and performance that rival native code. Theoretically, Together will run on any OS for which there is a Java Virtual Machine. However, different JVMs exhibit different bugs, among other things, so rather than claim that Together runs on everything, TogetherSoft listens to it's customers and concentrates on testing and supporting those operating systems that customers tell us are the most important. As of this writing the list of supported operating platforms includes:

- Microsoft Windows® (NT, 2000, 9x)
- Sun Solaris™
- LINUX
- Hewlett-Packard HP UX
- Compaq Tru64

With Together, the Platform is only the beginning. To it, TogetherSoft adds *Building Blocks*...; "pluggable" units of functionality...; and then bundles the Platform with different levels of Building Blocks into products. Let's see how that works.

Building Blocks

TogetherSoft takes the Platform, adds some Building Blocks, and the result is Together Solo - a good choice for smaller scale development that doesn't encompass EJBs and Web components.

Even more Building Blocks get added in producing TogetherSoft's flagship and premium product, Together ControlCenter. Best of all, this architecture is highly extensible. You can develop your own custom Building Blocks and plug them into the Platform in the same way we do with Together's core features and TogetherSoft extensions!

But what's probably more interesting for you is the fact that what *we* can do to create our products, *you* can do too. You can create your own Building Blocks to extend, enhance, and customize Together, and plug them right into the Platform using the Together Open API.

The Together Open API

Together comes with a comprehensive, multi-tier open Java API that provides almost unlimited extensibility. Your Java programmers can develop their own custom Building Blocks that plug right in to the Platform just like many of Together's own features. Develop custom documentation formats that extract information from the model (stored as source, of course!) to meet company or government standards. Create custom support for your preferred application server or JDBC database. Tightly integrate with your own company's, or a third party's software. The list of possibilities goes on and on!

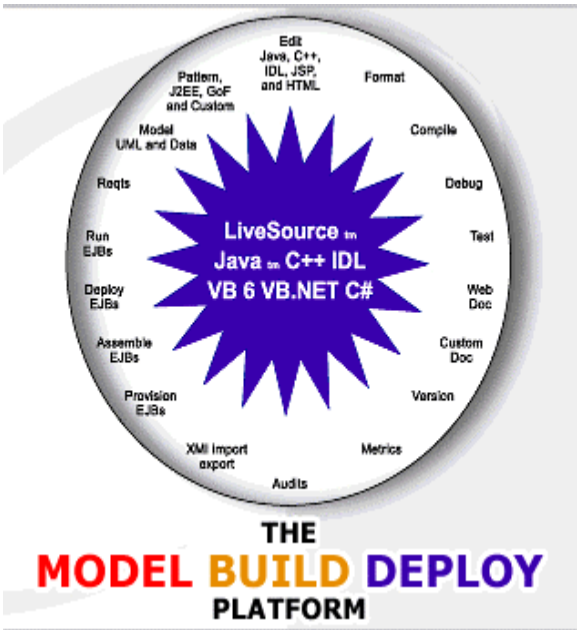
More about Together ControlCenter

Together ControlCenter, delivers adaptive business-process automation for teams building software solutions. Together ControlCenter brings your e-solutions team together, allowing business users, developers, and operations people to collaborate using a common language, diagrams, and software.

Together ControlCenter enhances productivity and process management in critical areas:

Automating mundane business processes: adaptive documentation generation, for example. Automating tedious and error-prone business processes: work required to adapt and deploy an application on an application server, for example.

Automating expert-level insights with guidance on how to adapt and apply those insights correctly using. patent-pending expert-level pattern technology.



1. EJB provision-assemble-deploy to 16 app-server products
2. The entire environment works with source code; always in- sync, always up-to-date
3. Track requirements with model properties, or use the building block available for DOORS
4. Building block available for JUnit.

What next?

To get started with some key Together features, see Feature Fly-over and Extended Features.

Feature Fly-Over

This topic presents a run-down of features supported by Together, linked to the appropriate chapters in the help system.

Feature overviews

Modeling and design features

Simultaneous Round-trip Engineering
Hyperlinking to artifacts & information
UML diagrams and support plus special purpose diagrams
Diagram Printing

Information import-export

IDL and DDL Support
Data modeling
Rational Rose Import
Other information Import-Export

Enterprise development

View Management
Multi-level, Flexible Documentation Generation
Multi-user Team Support
Robust programming editor
Distributed, multi-threaded Java debugger
Re-use support (patterns and templates)
Quality Assurance (Metrics and Audits) for Java projects
Testing

eCommerce rapid development

eCommerce/EJB development and deployment support

Extensibility and customization

Customization capabilities
Advanced customization
Together Open API

Where to get the latest info on features

The features listed were current at the time of this release of Help. Note that not all features are present in every Together product. For comprehensive, up-to-date information on what features are available, and what products have them, visit: **www.togethersoft.com/together/**

Extended Features

A wide variety of building blocks, plugins and bundled products extend Together's functionality beyond the fundamental features outlined in the introduction. These can be tentatively divided into three groups:

1. **Building blocks created via API** realize certain specifications and technologies. These are: XML modeling, QA features, search for usages, doc generation, etc.
2. **Integration modules** are designed to integrate Together with third-party products and support data interchange with them. These are: HP E-Speak, Doors, PowerTier, Versant, JProbe, etc. You can find documentation for these modules in the Building Blocks Help on the *Help* menu.
3. **Bundled software** allows to use Together in conjunction with various freeware and commercial products. The most vital tools for enterprise software development are integrated with Together and delivered as whole products in the `$TGH$/bundled` folder.

The building blocks display on the Modules tab of the Explorer. The treeview contains three nodes: Early Access, Samples and System.

Modules declared as *Early Access* are not fully developed and/or undocumented. However, they are still accessible from the Modules tab. The *Samples* node contains sample modules delivered with the source code. *System* contains completely developed and documented modules.

You are welcome to extend Together even more with the building blocks of your own. To learn how to develop your own modules, refer to the Extensibility and Advanced Customization chapter of the User's Guide.

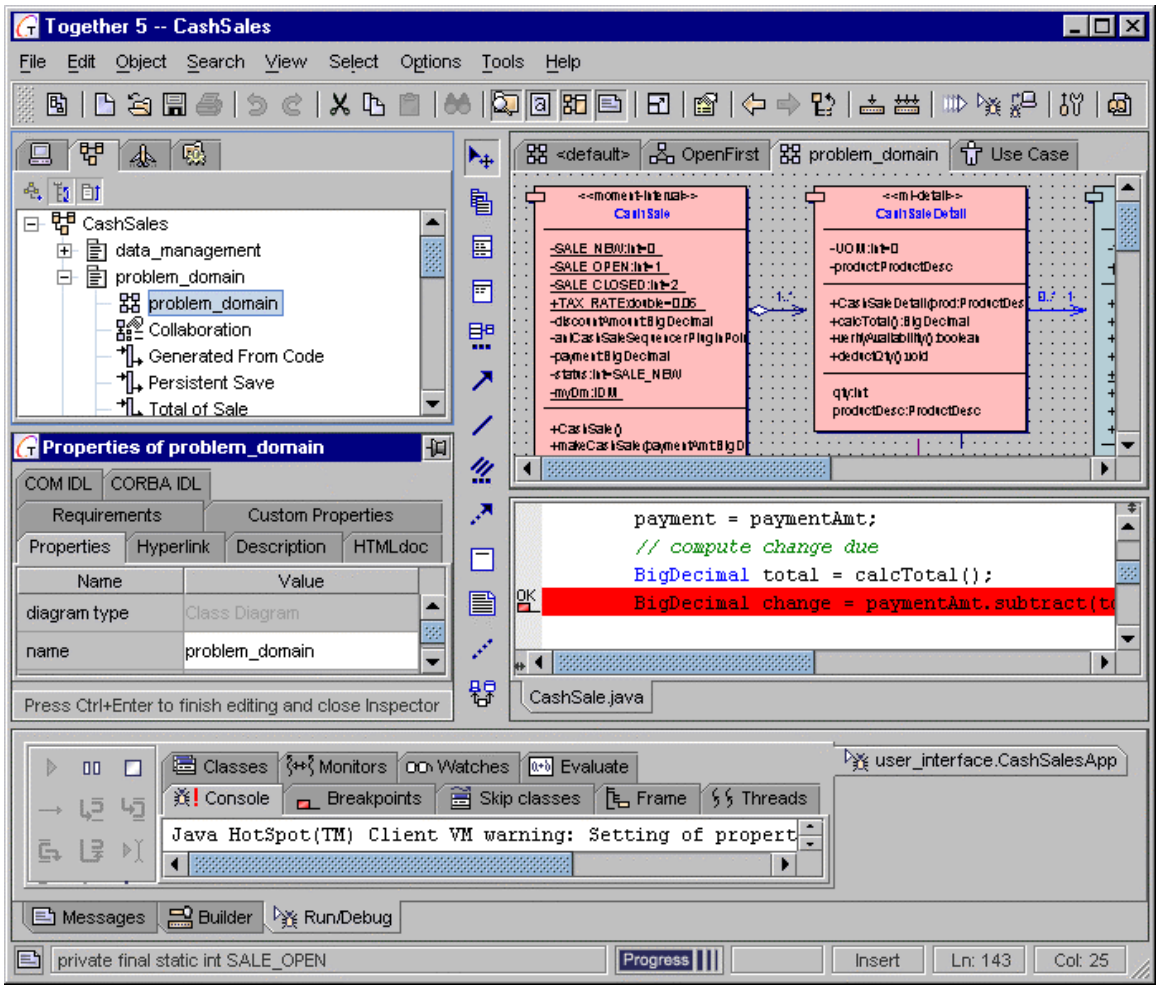
Together Quick Tour

Together's user interface is easy to learn and use. The topics in Quick Tour describe the principal components of the user interface and the main functions of each one. The links below will take you to additional information about the major UI elements.

Note that the appearance of the main window varies according to your currently configured role. For more information, see Role Based Workspace.

Main Window

Together's main window gives you centralized access to all your work in progress. You can open multiple diagrams or files and work on them concurrently in the same window. Each open diagram or file displays in its own tab page in the appropriate pane. The Main Window is comprised of three main regions or panes.



Explorer pane enables navigation of your system, provides overview and detail views of your project, and provides easy access to system and extension modules, and reusable components. Use the speedmenu to access properties, version control, compile and make tools, QA analyses and more.

Note: Property Inspectors are attached to individual elements. Inspectors are accessed from speedmenus of elements and diagrams.

Diagram pane is for drawing models visually

Editor pane is for viewing and editing diagram elements' source code, module source code, or text files. You can customize a number of options for this pane on the Text Editor page of the Options dialog.

Message pane dynamically displays several tabs that provide information (system messages) or enable you to perform a specific task (debug code or navigate to problem spots).

> *Message page* displays a queue of messages from the system. Non-critical error messages display here rather than interrupting work flow with modal dialogs. This pane is hidden by default. Use the speedmenu to save or copy messages, navigate to problem spots described by a message line, or clear the message queue.
>
> *Debugger page* displays when the integrated debugger is invoked. Use the speedmenu to navigate to the bug described by a debugger message line.

Click on any of the above links to go to the Getting Started topic for the specific pane.

Main Menu

Of particular interest on the main menu are the following:

Object: The Object menu replicates the speedmenu of a currently selected object such as a class or diagram. It is only available when an object having a speedmenu is selected, and the menu content changes dynamically to reflect the speedmenu of the selection.

Search: find and/or replace text strings in source files, diagrams, and across multiple files in your project.

View: You can use the View menu to hide and show any of the panes quickly and easily. You can also maximize any of the panes. Pane toggles are replicated on the Main Toolbar. Note the keyboard shortcuts displayed on the menu.

Select: Provides a quick easy way to select the different panes or select among open diagrams. Check out the keyboard shortcuts on this menu.

Options: Use this menu to customize your Together configuration. You can set configuration options to apply at different levels: installation-wide, project-specific, or diagram specific.

Tools: This menu provides access to several system modules (also accessible from the Modules tab of the Explorer) such as documentation generation and Rose Import/Export. Special features such as information import/export, compile-make-run for integrated tools, and change synchronization for task-switched tools are also found here.

Main Toolbar

This toolbar replicates a number of commonly needed commands from the Main menu. Of particular interest are the following:

> **Pane toggles:** These buttons duplicate the View menu commands that show, hide, or maximize the different main window panes.
>
> **Diagram View Management:** Launches the Options dialog at the diagram configuration level. There you can set view management options to show or hide different types of diagram content. (For information about *Together*'s multi-level configuration architecture, see Configuring Together: Multi-level Architecture Overview).

Debug project: Launch the integrated debugger (not available in all products).

The main toolbar can be undocked from the main window by dragging the "handle" at the left edge. Dock the undocked toolbar by clicking the close button in the toolbar's frame.

Status bar

The status bar cells, left-to-right are:

Messages pane: Hide or show the Message pane

Diagram View Management: launches Options dialog with only View Management options visible. The settings are those in force for the current diagram. You can view and change these options to control what is displayed in the Diagram pane. Changes apply only to the diagram.

General info: this cell shows information about elements in diagrams as you mouse-over them.

Progress: Bar shows progress of internal processes that you invoke at different times while working in Together.

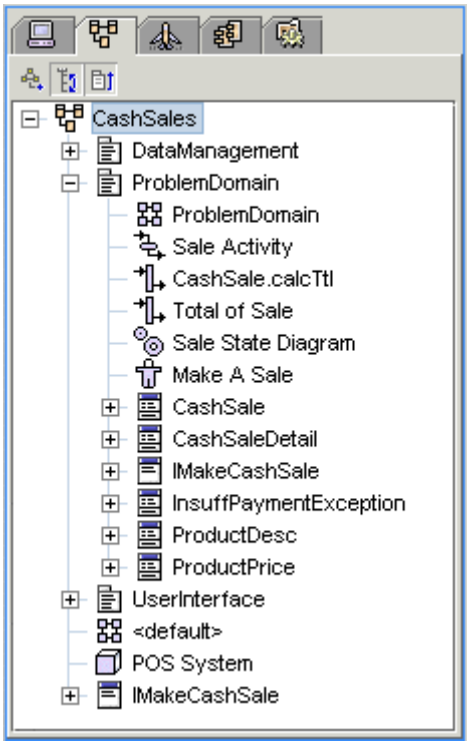
Editor info: the remaining cells display information related to the Editor pane:

Changed file indicator
Insert/Overwrite mode
Current line number
Current character column position

Resizable frames

Some of the Together dialogs and frames are resizable. This is marked with a hatching in the lower right corner  .

The Explorer



The Explorer pane has multiple tabs that provide different views of your project's content and enable access to the related files such as modules, and other information in your *Together* installation. You can...

-View the physical and logical structure of your project.

- Navigate within the project, and also in your physical directory system.

- View project resources such as source files, classes and members, diagrams, and modules. Open diagrams and files for editing.

- View and run Together system and feature building blocks and any custom ones you develop.

You can show or hide the Explorer using the View menu or Main Toolbar. The tabs, from left to right are: Directory, Model, Overview, Components, Modules.

Directory tab

The Directory tab presents views of the contents of your physical system and enables you to navigate both inside and outside of a project's physical structure. You can find and open text type files in the Editor pane independently of your current project, or when no project is open. You can navigate to and open a project file. When a project is open, you can also open a diagram file.

The actual physical files that Together supports for the Editor pane (source code and text files) display the Edit File icon. Together project files display the icon. Other files that reside on your physical system but cannot be opened in the Editor pane show a "generic file" icon:.

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

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

When you open a project, a *Current Project* node is added to the Directory tab enabling you to see the physical files that comprise your project.

Model tab

The Model tab displays when you open a Together project and shows a logical view of the elements that comprise the model encompassed by the project.

The Model tab shows what root-level packages comprise your project, and enables you to logically navigate the contents to see what subpackages, diagrams, and diagram elements exist in each one. As you browse the project contents, you can open diagrams for editing using the speedmenu. The Model tab's view is not strictly hierarchical in the same sense as a file system explorer because the project's root-level packages can physically reside *anywhere* on your system. The Model tab displays *secondary* root packages in relation to the *primary root* -- i.e., 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 secondary root directory for the project. In this case, `mysources` displays as a package node under the primary root node (`Project1`). From that point, the contents of `mysources` displays hierarchically.

Drag and drop is supported in the Model tab.

Model tab toolbar

As shown in the opening figure, the Model tab displays a small toolbar. The icons on this bar control the presentation of information in the treeview. They toggle their respective functionality on and off. From left to right:

Expandable diagram nodes: Controls whether or not you can expand nodes representing diagrams to show diagram content. Toggled *off* by default.

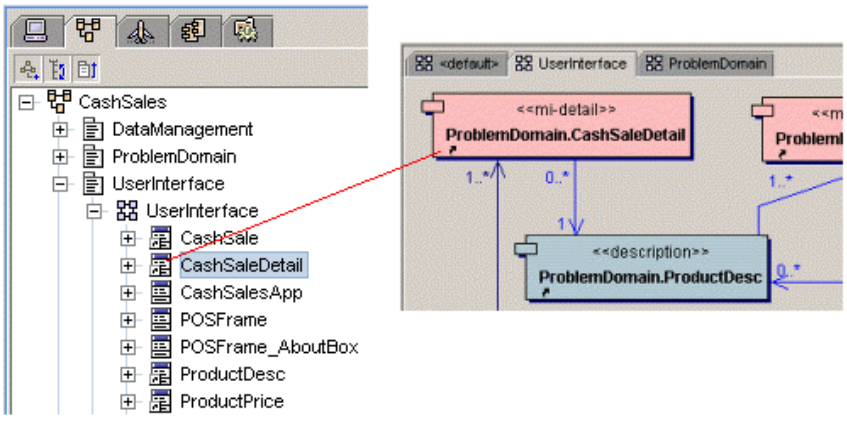
Sort nodes: Controls whether or not nodes in the treeview are sorted alphabetically. Toggled *on* by default.

Packages first: Controls whether or not packages display first in the treeview before other content. Toggled *on* by default.

Understanding shortcuts

First of all, understand that Model tab treeview nodes are just representations of your project's content... mainly diagrams, classes, interfaces (Java), and members.

Sub-nodes in the Model tab can sometimes display shortcut symbols. If a sub-node represents something that is represented in, but not contained by the parent node, then a shortcut displays. For example, if a Class diagram displays classes that reside outside the Class diagram's package, these classes show up with shortcuts in both the treeview and the diagram as shown in the figure below.



Classes from a different package display as shortcuts

If a sub-node represents something that is contained in or by the element that the parent node represents in the treeview, no shortcut displays. So in the above figure, CashSaleDetail has a shortcut because it doesn't live in the UserInterface package, which the diagram depicts. But POSFrame has no shortcut because it is native to the UserInterface package.

Class diagrams can show classes and interfaces from packages that reside on the Classpath and/or other search paths as defined in File | Project Properties. Such content is also referenced and therefore displays a shortcut symbol both on the Model tab and in the diagram where it is shown.

> **Note:** You will not usually see shortcuts in the Model tab when the control *Expandable Diagram Nodes* is toggled *off* on the Model Tab Toolbar.

Favorites tab

You can optionally add the *Favorites* tab to the Explorer. To add it, choose Options | Default, expand the General node, check the *Show Favorites* option and click OK. The setting will take effect immediately. You can opt to display the names as either fully qualified or short, by setting the *Show fully qualified names* flag under the *Show Favorites* option.

Favorites in Together works much like the feature of the same name in Microsoft Windows. You can use the Favorites speedmenu to add subnodes that link to the elements in the project that you need to access most frequently. You can then use the Favorites tree for easy navigation to these elements without having to find their actual location in the Model tab. Though useful for any project, it is particularly so when the project is quite large, as it saves you time navigating through a huge hierarchy just to get to the parts you are working on.

To add an element to Favorites:

1. On the Model tab, find and select the element to be added to Favorites.
2. Right-click the element and choose *Add to Favorites* from the speedmenu.

You open elements in Favorites in the same way you would open them from their "real" location on the Model tab.

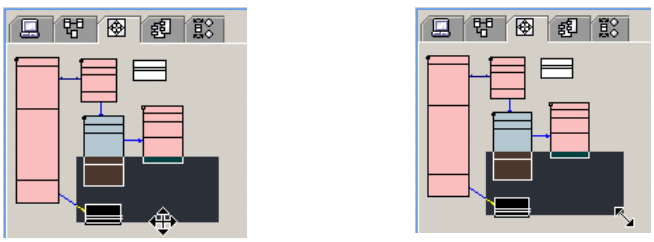
To remove an element from Favorites:

1. Select the element in the Favorites tree.
2. Right-click and choose *Remove from Favorites* from the speedmenu.

Overview tab

The Overview tab  enables you to visually navigate the diagram currently selected in the Diagram pane and to quickly adjust the zoom level. The tab displays a "thumbnail sketch" of the entire diagram, and a shadow over the region currently visible in the Diagram pane. The Diagram pane scrolls proportionally with any movement of the shadow.

- To move around the diagram, drag the shadow to scroll the Diagram pane to the region of the diagram you want to view.
- To adjust zoom level, resize the shadow by grabbing the lower right corner diagonally to increase or decrease the Zoom level in the Diagram pane.



Overview tab: drag shadow to scroll, drag corner to zoom in-out.

Modules tab

Together is highly extensible. Using Java and the Together API you can develop your own modular *building blocks* to handle custom metrics or documentation, generate custom outputs based on model information... almost anything. In fact, many of Together's own features are implemented as modular building blocks. These appear in the *System* folder of the Modules tab.

The Modules tab provides quick access to system, sample, and any added-in building blocks (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 speedmenu of individual nodes. If you develop your own modular building blocks or acquire third-party building-block extensions, you can install them so they display in and run from this tab.

The following table shows how building blocks are represented in the Modules tab.


Java source file for module. Can be compiled from "Run" if a compiler is configured.


Compiled Java module


TCL script. "Run" executes the script in interpreted mode.

For more information, see Developing modular building blocks.

Components tab

This tab enables you to access and reuse component models. By default you'll find "Coad Modeling Components"... over 60 enterprise component models *in color* that you can reuse or modify.

The Components tab visually represents the `$TOGETHER_HOME$/modules/components` directory of your installation. You can add your own components to the Components tab by placing them in one or more directories that you create under the `components` directory.

You can copy classes (or entire packages) from the packages shown on the Components tab to your Class diagrams. This creates new source files and/or packages in your project. You can also copy classes (or entire packages) from the packages shown on the Components tab to any package in your project. These 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.

For information on copying between Explorer tabs and diagrams, see the Tips section below.

The Components tab is displayed if the appropriate flag is checked in the File | Project Properties dialog.

Diagrams tab

The Diagrams tab  appears if *Show Diagrams tab* is checked on the General page of the Options dialog. This tab displays the treeview of all types of diagrams available in Together. All diagrams in the current project show up in the appropriate nodes. You can access each diagram from its speedmenu.

Tips and Tricks

Tabs display

The following tabs are always present in the Explorer:

 Directory

 Modules

The following tabs appear only when you open a project:

 Model

 Overview

 Components (only when you include components directory in project)

 Diagrams

Navigation

Double-click on diagrams on the Model tab to open them for editing.

When you open a diagram in the Diagram pane, you can select a key element such as class, member, or use case (depending on the diagram type) in the Model tab, and this element is simultaneously selected in the diagram. The diagram scrolls if necessary to display your selection. This is a handy way to navigate in large diagrams when you want to locate a specific class, member, or other element.

Clicking on a class or member on the Model tab loads its source code on the tab of the Editor pane regardless of whether or not its associated diagram is open in the Diagram pane.

Speedmenus

Check for a speedmenu on the various node types on different tabs. The speedmenus enable such actions as opening, clipboard operations, and version control directly from the Explorer. Explorer nodes for classes, use cases, and other main diagram elements have the same speedmenu as the element in the diagram. You can perform the same operation from either place with identical effect.

Express project access

If 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 them and open them from there. You can customize the default location for projects on the General tab of the Default Options dialog.

Using the Explorer for Copy / Paste

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

To copy between Explorer tabs:

1. Select the Explorer tab containing the source, for example Component tab.
2. Select the item you want to copy (Explorer does not currently support multiple selection).
3. Choose copy from the item's speedmenu.
4. Select the Explorer tab containing the destination, for example, Model tab.
5. Select the destination package and choose Paste from its speedmenu.

To copy between Explorer and a diagram:

1. Select the Explorer tab containing the source.
2. Select the item you want to copy (Explorer does not currently support multiple selection).
3. Choose copy from the item's speedmenu.
4. Right-click on the diagram background and choose Paste from its speedmenu.

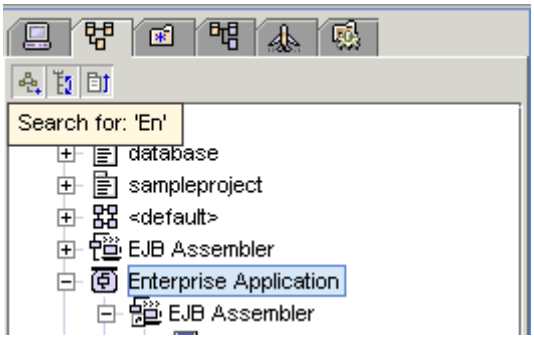
To clone something from the Explorer:

1. Navigate to the source item in the treeview.
2. Right-click on it and choose Clone from the speedmenu.

Note: You can also clone Node elements in the diagram pane using their speedmenus.

Find Location

On the Directory, Model, Modules, Diagrams and Components tabs, you can navigate to the desired location without actually scrolling the treeview. 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 treeview.



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 automatically complete the name and expand the node, and go on typing the name of the required nested node. Keep in mind that the entry is case sensitive, and using the wrong case will produce no result.

See also

Main window
Creating and opening a project

Editor pane

The Editor pane is located immediately beneath the Diagram pane on the right-hand side of the screen. Together features a robust, configurable programmers editor that rivals the best stand-alone editors with features like code-completion, bookmarks, symbol browsing, pane splitting, color and indentation schemes, and keyboard customization. If you prefer to use an external source code editor or IDE, no problem. You can configure *Together* to launch your favorite development tools.

The Editor highlights reserved words in the target programming language of the current project (or the default language supported by your *Together* product if no project is open). Presently syntax highlight is supported for Java, C++ and HTML.

Editor features

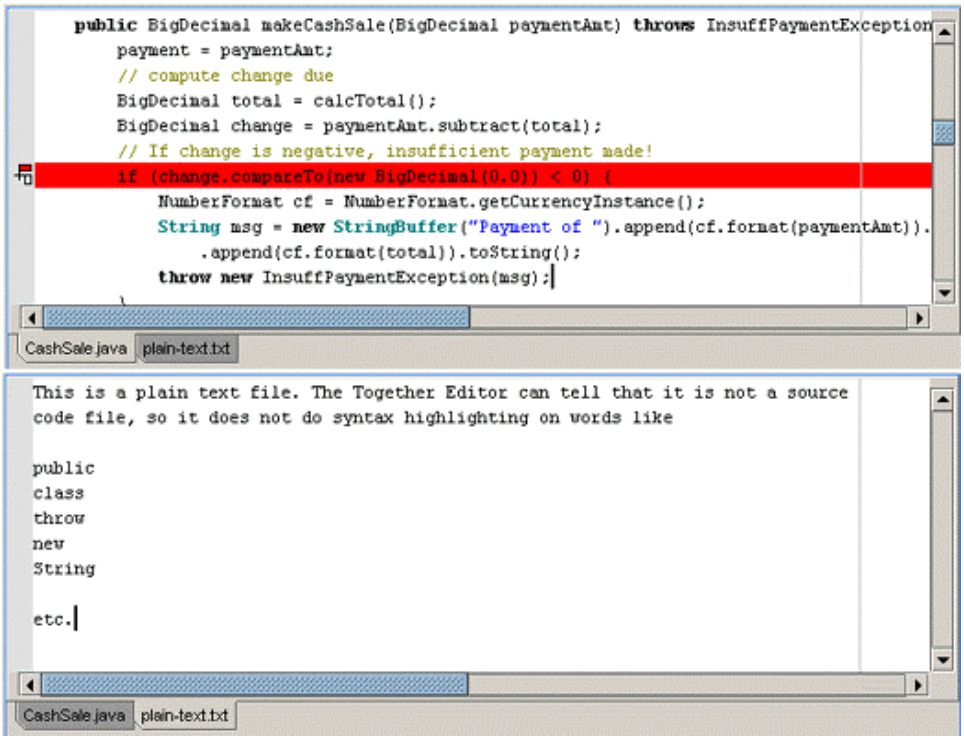
The Editor pane can hold multiple pages, each one with its own tab (similar to the Diagram pane). Each tab displays the filename of the file opened in it. When you first start Together, the editor displays a new *untitled* document.

When you select a diagram that has source code, the source opens and replaces the *untitled* tab.

You can open any text-based file in the Editor using File | Open on the Main menu. Files you open this way display in separate tabbed pages. Alternatively, you can navigate to the desired file in the Explorer, and choose Edit or Edit in New Tab on its speedmenu.

The Editor can tell the difference between source code and other text files. If the file is a source code file for the target programming language of the current project (or the default language supported by your Together product if no project is open), the Editor highlights reserved words.

The Editor pane has a speedmenu with commonly-needed commands. Configuration of the speedmenu depends on whether the Editor pane is used with an open project, or without it. From the Editor's speedmenu, you can configure the Editor settings (Text Editor options), control breakpoints and bookmarks, perform clipboard operations, and invoke external tools.



The Editor Pane: source code with breakpoint set, and plain text

Tips

- The Editor pane has its own speedmenu for commonly used operations.
- You can easily hide or show this pane using the View menu or the Main toolbar.
- You can also open files by selecting them on the Directory tab of the Explorer and choosing *Edit* or *Edit in New Tab* from the node's speedmenu.

See also

Using the Editor

Diagram pane

When using Together for visual modeling, the Diagram pane is your focal point. This pane is hidden, replaced by the Editor pane, until you open a project and a diagram. If the saved desktop feature is enabled in your configuration (Options | Default | General | Desktop Options), any diagrams open in your previous project session open automatically. Otherwise you need to open diagrams yourself by double-clicking on them in the Explorer.

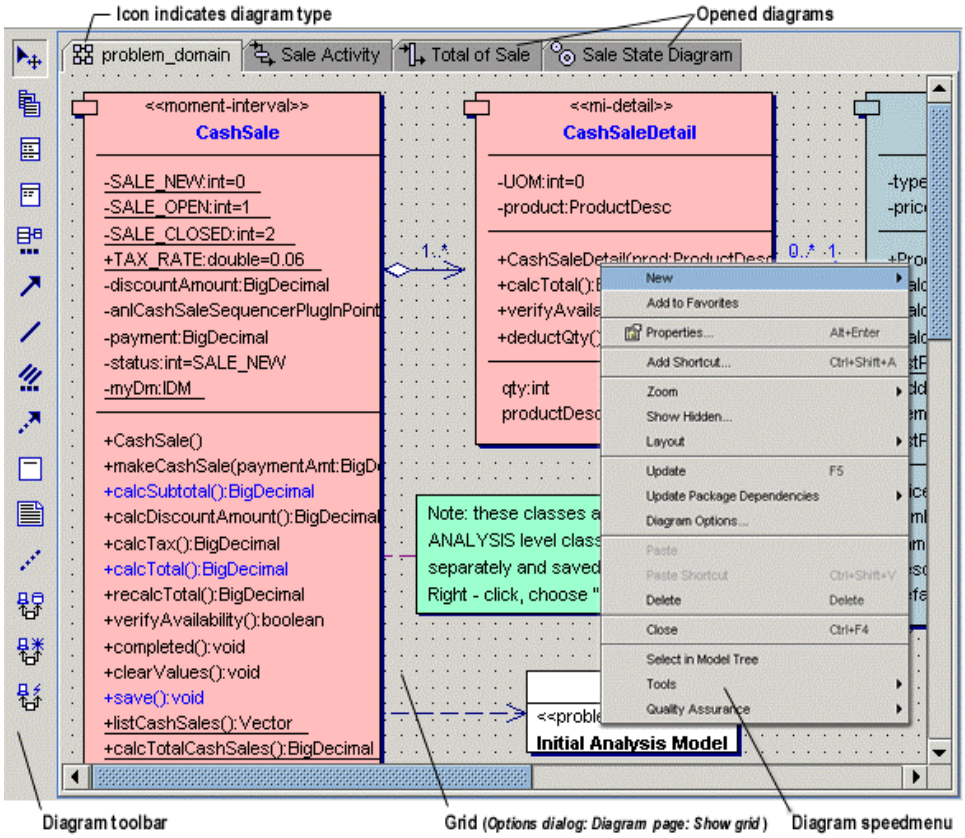
A tab is added to the Diagram pane each time you open or create a diagram. The tab displays an icon for the UML diagram type, and the name of the diagram. If the tabs for all the diagrams you open won't fit in the horizontal space available, a new row is created. You can switch between open diagrams by clicking on the tab of the one you want to see.

You can hide or show the Diagram pane using the Main toolbar or the View menu. To increase the size of the workspace you can hide the Editor pane and/or the Explorer pane.

Diagram Elements Toolbar

The Diagram Elements Toolbar displays at the left side of the Diagram pane. You use it to place or draw icons representing the structural and behavioral elements and interactions of your model on the background of the Diagram pane.

The Diagram Elements Toolbar is static but its composition varies dynamically depending on the type of diagram currently selected in the Diagram pane. Principal elements defined by the UML for the current diagram type are displayed along with any Together-specific enhancements (class by Pattern in Class diagram, for example). Use the mouse-over tool-tips to identify the toolbar icons. *Note* and *Note Link* elements are common to all diagram types.



The Diagram pane showing Diagram Elements toolbar for a Class diagram

Diagram properties and speedmenu

Diagrams themselves are objects that have properties. You can view a diagram's properties in the diagram properties Inspector. Click on the diagram workspace (this deselects any selected diagram elements) and choose Properties on the speedmenu.

The diagram's speedmenu contains a number of commands that operate within the context of the diagram. You can access such functions as adding elements, Zoom, Auto-layout, un-hiding elements, clipboard operations, and Quality Assurance. You can also access diagram-specific configuration options and, when properly configured, version control and external tools.

Related topics

Creating diagrams in projects
Opening diagrams for editing

Message pane

This pane dynamically displays several pages that provide information (system messages) or enable you to perform a specific task (debug code or navigate to problem spots).

Messages page displays a queue of messages from the system. Non-critical error messages display here rather than interrupting work flow with modal dialogs. This pane is hidden by default. Use the speedmenu to save or copy messages, navigate to problem spots described by a message line, or clear the message queue.

Run/Debug page displays when the integrated debugger is invoked, or displays runtime information when classes are run from within Together. Use the speedmenu to navigate to the bug described by a debugger message line.

Builder page displays build information when you make a project.

You can show or hide this pane from the View menu, Main toolbar, or keyboard shortcut. When the Message pane is hidden, an icon in the first cell of the status bar indicates the presence of messages in message queue. You can clear the queue by opening the Message pane (if hidden) and removing either single messages or all messages (choose Remove or Remove All, respectively, in the Message pane's speedmenu).

Using the speedmenu, it is also possible to sort the existing messages by time in ascending or descending order, and to store the contents of the message pane in a text file. By default, Together suggests `%TGH%/out/` as the target folder.

The capacity of the Message pane is configurable via the *Options* dialog. Choose *General* node and enter the desired value in the field *Messages maximum count*. The default amount of messages is 200. When the limiting value is reached, each next message pushes out the first message in the list.

When the Message pane is hidden, it is still possible to get notification about the new messages. The icon in the lower left corner of the Message pane allows to open or close the pane and see whether the new messages have appeared.

Icon	Message pane state
	Message pane closed; no new messages
	Message pane closed; new message appeared
	Message pane open

Messages page

Non-critical error messages display in the Messages page rather than in modal dialogs. For example, if you draw a link to an invalid receptor, the link isn't created, a message is written to the end of the queue, and the message icon displays on the status bar. If something you do appears not to work, check Messages.

Messages that Together intercepts from external tools such as version control or a compiler also display in the Message pane. Compiler error messages are of particular interest because you can double click on an error message in the Message pane and navigate directly to the code that caused the error.

You can double-click on messages displaying the  icon to open them in your registered text editor application. You can use the Message pane's speedmenu to copy messages to the clipboard or save them to a file.

Run/Debug page

This page appears when you start the debugger, or run the application. It displays the Debugger tabset that enables selection of different debug processes (console, threads, classes, etc.). It also contains the Debugger toolbar that enables you to start, stop, trace into, step over, etc. as you work with the Debugger.

Tip: Leave the Message pane open until you have gained some experience using Together.

Properties Inspectors

Diagram and element properties are accessible in property Inspectors displayed with the *Properties* command on the element or diagram speedmenu, or from the main menu *Object | Properties*. Property Inspectors are generally multi-tabbed, and their content is dynamic, depending on the selection.

To display properties:

1. Select the element or diagram in the Diagram pane or in the Explorer.
2. Do one of the following:
   - right-click and choose *Properties,*
   - use one of the keyboard shortcuts *(Alt+Enter* , or *Alt+double click*),
   - click on *Properties* icon on the toolbar, or
   - select *Object | Properties* from the main menu

Overview of Inspectors

Inspectors generally (but not always) have multiple tabs that enable you to...

- View and/or edit properties of the selected diagram or element.
- Create and navigate hyperlinks between...
  - diagrams (existing or created "on the fly")
  - diagram elements and other diagrams (existing or created "on the fly")
  - diagrams (or elements) and files or URLs
- Enter and edit comments in source code; add comments to non source-generating diagram elements.
- Add JavaDoc comments for the selected element such as @author, @version, etc.
- Enter and edit Requirements information.
- Edit properties specific to the selected element.

The properties' editors in the Inspector are different, depending on the value types. Where multiple values are enabled, the fields are marked with an asterisk  .

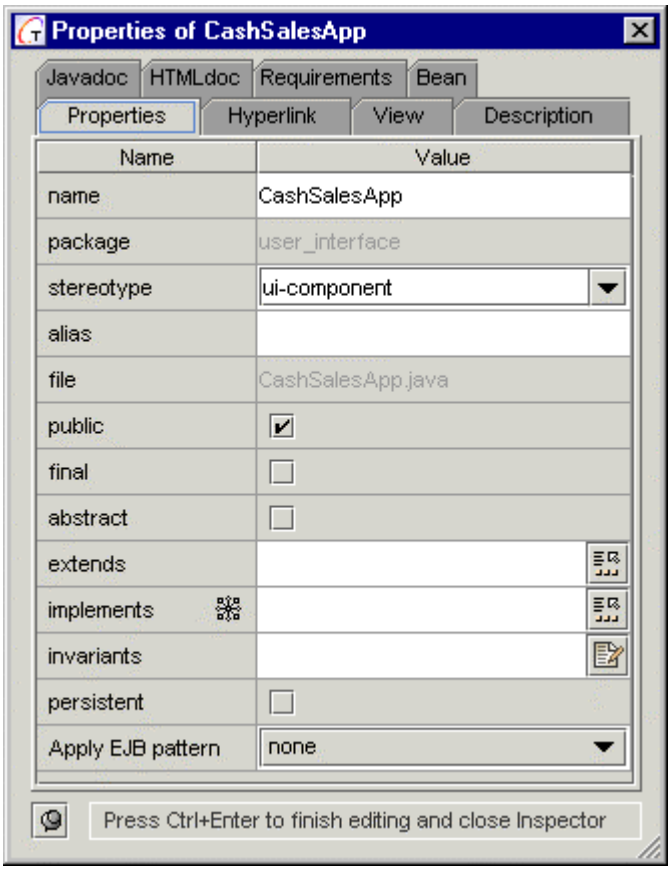
Some fields have the file/path chooser  button, that invokes selection dialog. Button  invokes a dialog that displays the list of current property values and enables adding or removing them. To add or modify textual values use editor button  .

The button  is used in the view adjustment fields. Pressing this button invokes a dialog where you can choose foreground and background colors.

Once entered, a value can be cancelled or deleted. ESC cancels entry in text fields only unless Enter is pressed to complete. If a value is already entered, you can use Undo/Redo icons 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 just select it in the inspector and press Delete.

Inspector dynamic tabset

The composition of the Inspector's tabset changes dynamically depending on what you have selected in other parts of the user interface. This section describes the basic function of each Inspector tab along with the scenarios related to its visibility.



View and edit properties in dynamic properties Inspectors. Class inspector shown.

Properties tab

This tab shows the properties of the currently selected element in the active diagram, or the diagram itself. You can view and modify values of properties.

Some properties (*implements* of a class, for example) can have multiple values. When more than one value is specified, the values are comma-delimited. The fields that enable multiple values are marked with an asterisk.

Each property has a control field that you use for supplying property values. The type of control displayed depends on the type of property.

For more information on properties and property editors, see Editing properties in the Inspector.

Hyperlinks tab

The Hyperlinks tab enables you to create hyperlinks to different types of artifacts and browse directly to them. You can create hyperlinks from the current diagram as a whole, or from a selected diagram element to:

A new diagram (created on the fly)
An existing diagram or diagram element anywhere in the project
A URL on your company's intranet or on the Internet

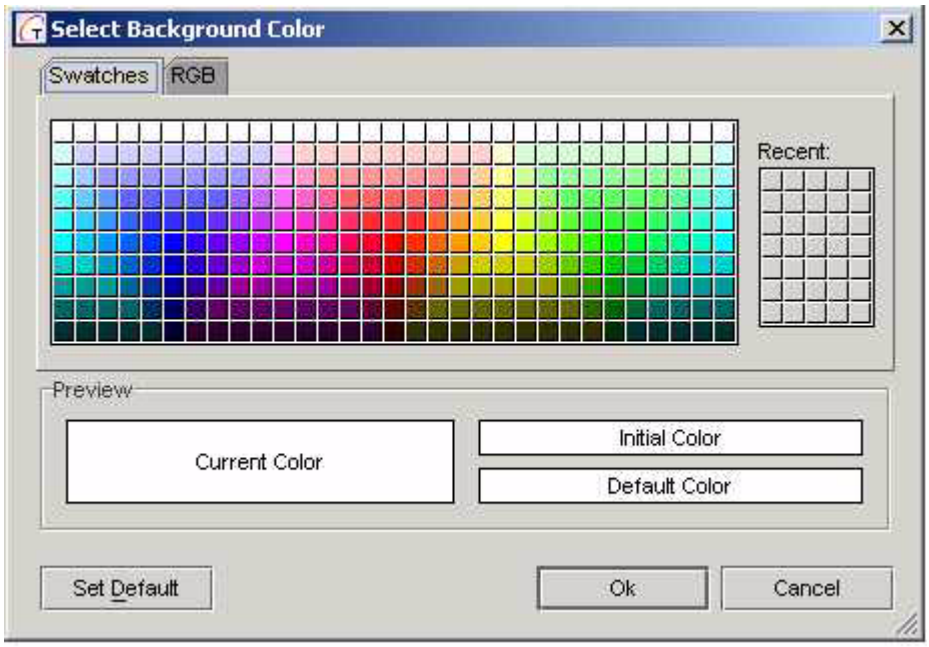
You create, view, remove, and browse hyperlinks with the Hyperlink tab speedmenu. For more information, see Hyperlinking Diagrams.

View tab

This tab, when displayed, enables you to set the foreground and background colors of the selected element.

When you start working with Together diagrams the foreground and background colors are the default Windows colors.

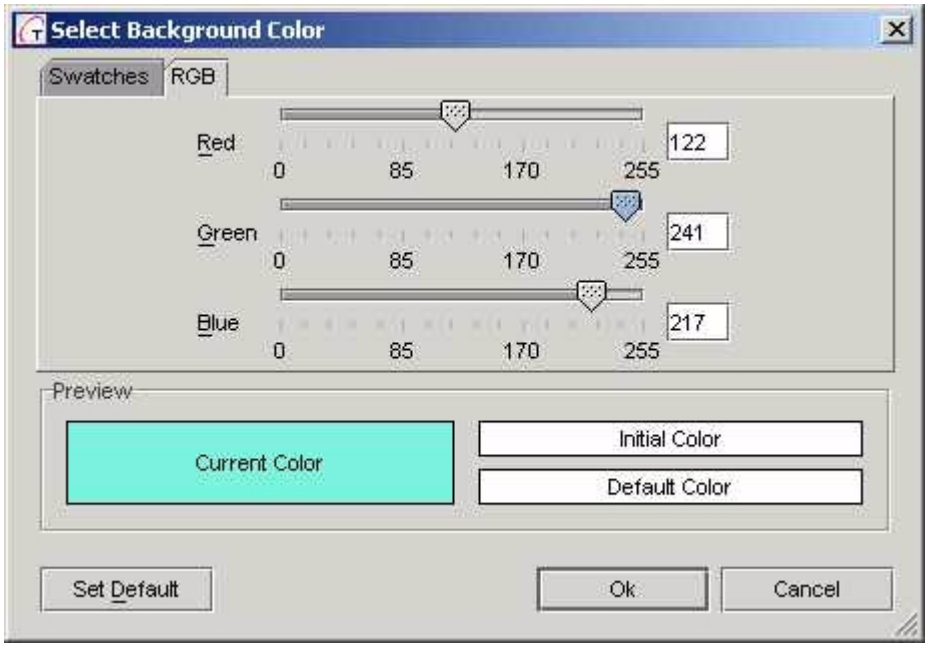
If you want to change the background color of some selected node, press  button in *background* field. *Select background color* dialog appears:



You can choose background color from the standard Swing scale of colors using *Swatches* tab, or define customer color using *RGB (Red-Green-Blue)* tab.

All history of chosen colors is represented in *Recent* matrix.

If you select *RGB* tab, you can choose the background color moving sliders:



If you want to restore the default color, press *Set Default* or D hot key.

You can also change the default color in `config` file.

Description tab

For diagram elements that are round-trip engineered, the Description tab displays source code comments when you select the element. For example, if you select an Interface, source code comments in the relevant source file display in the Description tab where you can edit them.

For diagram elements that aren't round-trip engineered (a Use Case for example) you can enter comments for the element that are stored with the diagram file.

Caveat

Avoid using '*/' characters in the tags. These characters are treated incorrectly, and the information in diagrams may be lost.

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. The values you enter are used when you generate JavaDoc using the Documentation Generation feature.

Filling in the javadoc fields automatically generates appropriate tags in the javadoc tags in the source code. `@author` and `@see` tags allow multiple values. In this case a separate tag is generated for each value in the inspector field. Checking the flag `@deprecated` creates an empty tag; use editor button  to enter specific comments.

You can find guidelines for javadoc comments at
www.java.sun.com/products/jdk/javadoc/writingdoccomments.html

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 that 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 adds to the inspector when *Recognize Java Beans / C++ properties* options are selected in the *View Management* page of the *Options* dialog. For classes, it displays JavaBean/EJB properties and events on the following lower tabs:

General tab: general JavaBean attributes
Properties tab: Bean properties like getter, setter, bound, constrained, etc.
Events: Bean event sets

Operation tab

The Operation tab is hidden until you set a value for a diagram element property *Operation* (for example, a Message in a Sequence diagram for example*).* The value of this property is an operation. The Operation tab displays the properties of the linked Operation. Once a value is set for the property, the Operation tab displays whenever the element (e.g. Message) is selected in the diagram.

DDL

DDL tab is hidden unless you check *persistent* checkbox in the Properties tab of the inspector. This tab provides entries for the database table name and primary key.

Custom properties

This tab appears in the inspector when Custom properties module is activated. For more details refer to Customizing Properties' Inspector.

Dockable inspector

Properties inspector displays as a separate tabbed frame, but if you so wish you can tack the inspector to the Explorer pane. To do so, press thumb-tack icon in the lower left corner of the inspector frame, or press Shift-Enter. The contents of the inspector dynamically changes as you select elements in the Diagram pane.

To undoc the inspector, press thumb-tack in the upper right corner, use Shift-Enter hotkey, or just drag out the title bar of the inspector.

Alt+Enter hot key completes changes in the inspector fields and moves focus from the docked inspector to the previously selected pane.

See also

Editing Properties in the Inspector
Customizing properties inspector

Main Window Tips

Speedmenu: Most components of the Together user interface have speedmenus (also known as "context" or "right-click" menus). If you're not sure how to do something, check these menus for appropriate commands.

Icon: Commonly-needed Main menu commands have parallel icons on the Main toolbar and display the same icon that appears on the toolbar.

Show/Hide Panes: Use the View menu or Main Toolbar to show/hide panes. Note the keyboard shortcuts for these too.

View: You can create diagram-only or code-only views by hiding the pane you don't need. Use the View menu or pane toggles on the Main Toolbar. Create wide views by hiding the Explorer pane.

Resize panes: There are movable separators between the major panes. Use them to resize an individual pane or window region.

Use Explorer: When you have a diagram open in the diagram pane, you can use the Explorer to select specific elements (Classes or Use Cases for example).

> Use the **Model tab** for an Explorer-style view of the main diagram elements. Click on the element you want to select in the diagram and the diagram will display it selected.
>
> Use the **Overview tab** for a thumbnail view of the current diagram. Drag the shadow to move the view; resize the shadow to adjust the zoom level.

If something is missing in a diagram that you think should be there, check

> **View Management Options** (Options | Default | View Management). These options may be set to hide some diagram elements. The status bar displays an icon when one or more of the Show options is set to filter something out from view.
>
> **Show Hidden dialog** (Diagram speedmenu). This will show you any elements hidden with the *Hide* speedmenu command.

Opening files: You can open files from the Directory pane or the Editor pane using their speedmenus.

Configuring Together

Configuring Together

Together is extensively customizable. There are many configuration options you can set to change the way different subsystems work, or to handle data in the ways that fit with your own requirements. For example, corporate managers or team leaders may wish to modify round-trip engineering blueprints or create custom documentation templates so that code generation and generated documentation conform to the corporate standards or conventions. Analysts or designers may want View Management options set to hide implementation details; engineers may want their code formatted a certain way. All these customizations and many more are quite easy to do.

This topic presents an overview of the configuration architecture and categories of configurable options. The other topics in this section explain how the configuration interface works and how to do the most common configuration management tasks.

Overview of the multi-level configuration architecture

Configuration settings are applied at multiple *levels*. The pre-configured default levels are:

Default: configuration settings are always in effect installation-wide unless overridden at a more local level.

Project: settings apply to a specific project overriding the same settings at the *default* level.

Diagram: settings apply to a specific diagram overriding the same settings at the *project* and/or *default* level.

Configuration options can be marked "final" at any level, meaning that they cannot be overridden at another "lower" level. When Together is run as a server-based application, the configuration is centralized and shared by all users. An administrator or manager can customize the configuration so that some customizations apply globally across the enterprise (*default* level-marked *final*) and/or globally for a team (*project* level - marked *final*). This enables unobtrusive enforcement of conventions, guidelines, and standards for code construction, formatting, diagram content, and more.

For example, Version Control and Source Code options might be set at the *default* level, and Diagram and Print options at the *project* level, marking them *final* in both cases to prevent users on a shared installation from overriding them.

Modifying the levels

It is possible to modify the definitions of the pre-defined configuration levels and to add additional levels (see Modifying the default config levels for more information).

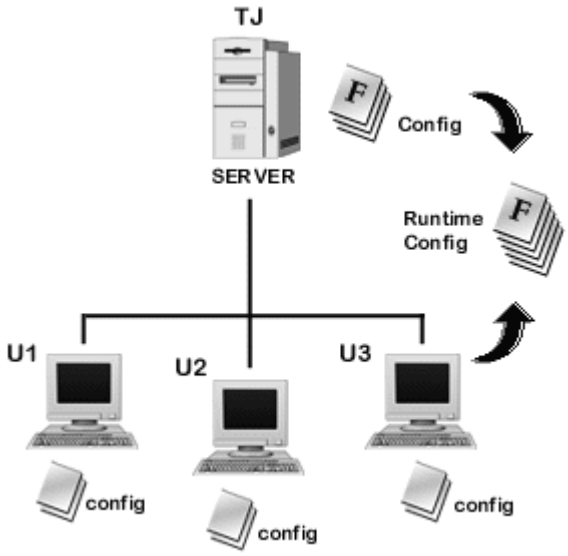
Where to go next

- For information on setting options at different levels, see Using the Options dialog.
- For an overview of the categories of customization options and how to access them, see Guide to the Options pages.
- For a list of frequently-needed customizations and how to do them, see Common Configuration Tasks.
- C++ users: be sure to see Configurations for C++.

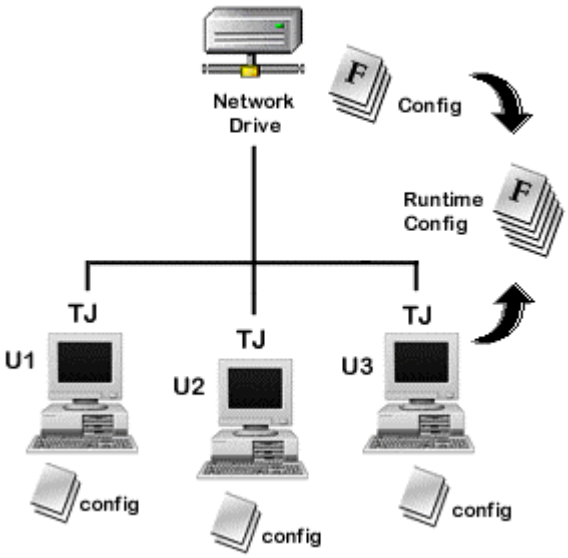
Creating a Shared Multi-User Configuration

The multi-tier approach to configuration properties enables a project or system administrator to provide a set of configuration properties (code-generation rules for example) for all users of the installation. By marking some configuration options as *final* at the Default (installation-wide) level or project level, an administrator can prevent users from overriding the configuration settings. Options not marked *final* leave individual users free to use their own settings. There are two basic scenarios for shared configurations:

Together runs on a server, all users access the server installation's global configuration settings which merge with individual settings at runtime:



Together runs on individual workstations and users share a centralized set of global configuration properties which merge with individual settings at runtime:



Shared configuration for a server-based installation

To set up a server-based configuration for a shared installation:

1. The administrator or manager (ADMIN) should run *Together* on the server console and edit the configuration options as described in the following configuration topics, marking as *final* those options that individual users may not override :

    Multi-level architecture overview
    Using the Options dialog
    Guide to the Options pages

2. If additional one or more configuration levels are desired, create them as described below in *Adding new levels*.

3. Next, the ADMIN should make a copy of the `./config` directory (and the properties directories of any added configuration levels) for each user of the shared installation. This copy can reside on the user's local drive or on the network in a separate folder intended for an individual user.

4. Finally, the ADMIN should create a start-up batch or command file for each user that launches the shared Together application from the command line. This file should be installed to the user's computer.

To get server-based installation with developer-specific configuration settings, see Modifying the default configuration levels below.

For information on launching Together from the command line, see Reference: Command Line Parameters.

Sharing configuration among workstation installations

It is also possible for multiple users with their own copies of Together running on separate workstations to access a shared multi-level configuration. In this case, the configuration properties customized by an administrator or manager reside in a centrally accessible location on the network. Individual users launch Together from a batch or command file specifying the path to the central configuration using the *-config.path* command-line switch.

To set up a multi-workstation shared configuration for a shared installation:

1. Install Together to the ADMIN's workstation. Make a back-up copy of the original config directory of the Together installation.

2. ADMIN run*s Together* on the ADMIN workstation.

3. If additional configuration levels are desired, ADMIN should create these as described in *Adding new levels* below.

4. Open the Options dialog, click *Levels*, and edit any or all configuration options. Mark individual options as *final* at the desired level for those options that you don't want to allow individual users to override from a "lower" level.

5. Next, ADMIN should copy the modified `./config` directory (plus any additional directories created for added config levels) to a shared network location (e.g. `<server>/tg_shared_config`) that is accessible to all *Together* users who should use this configuration.

Creating the start-up pointer file

When the entire configuration is to be centrally shared, it's necessary to create pointer file to define the centralized location and override Together's hard-coded configuration locations. Once it is created and the correct location(s) defined, you need to reference the pointer file in the `-config.path` switch when you start Together from the command line (or a batch or command file). The easiest way to create the pointer file is to let Together generate a copy of its on 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:\Together5\lib\pointer.config
```
2. Together starts up and writes the specified file to disk. You can then close Together.

The file name is just a suggestion. You can name it anything that doesn't conflict with Together's property file names. You can store the pointer file anywhere but it's 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 = $PROJECT_DIR$/$PROJECT_NAME$.tws
```

The highlighted lines are the ones you need to edit to point to your shared configuration. Note that paths must reference mapped drives (`\\appserver\tg_shared_config\*.config` will not work, for example). Thus, if your shared configuration is on *appserver*, and that is locally mapped to drive **s:**, then the edited lines would 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

Finally, ADMIN should create for each user, or instruct each user to create on their workstation, 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
```

For more information on launching Together from the command line, see Command Line Parameters.

Adding new levels to the predefined ones

It is possible to add up to three additional configuration levels to the pre-defined ones (i.e. *Default*, *Project,* and *Diagram*). For example, a *Corporate* level could be added to enforce certain configuration settings across the enterprise. As a simple example: 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. New levels can only be inserted *above* the installation-wide *Default* level in Options Dialog order.

To add configuration levels, you must do two things:

1. Copy the contents of the `$TOGETHER_HOME$/config` directory to some other location to create a separate set of configuration properties for the new level. This could be 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 file `path.config` file in the `./lib` directory and point it to the directory you set up for the new level(s). Together searches this file and loads additional configuration levels from the location(s) specified. If the location in 1 above is shared, be sure to do this second step on all the local machines that need to share the configuration.

Adding new properties directories to a shared location

If you are creating the new level(s) for a configuration that will be shared from a central location, you must create a set of configuration properties files for each new configuration level. To do this:

1. Copy the `./config` directory of your Together installation to the shared location and rename it `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. For example, `tg_shared_config/corporate.`

TIP: You may find it easier to create the subdirectory(ies) under your local `config` directory, test that you have defined the new level(s) 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 level(s) for a configuration on a local installation, you have to copy the `./config` directory of your Together installation to some other location and rename it with the name of the level it represents. For example, `./config/corporate`.

For example, to create a *Corporate* configuration level, you first copy everything in the config directory to some other directory... let's say `$TOGETHER_HOME$/config/corporate`.

Creating the *path.config* file

Use a text editor to create the file `$TOGETHER_HOME$/lib/path.config` (check to see that someone else hasn't already created it).

If you are setting up a shared configuration, you'll need to place a copy of this file in the Together installation of all users.

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; for example `$TOGETHER_CONFIG/corporate/*.config`).

> ***TIP:*** Note that `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 (`$TOGETHER_LIB$, $TOGETHER_CONFIG$, $TOGETHER_HOME$` for example) part of the path specification.

If you are adding more than one level, you must 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 the added configuration level(s)

To verify that you have correctly added a new level, run Together, choose *Options* dialog on the main menu and click the *Levels* button. (If you created the new levels in a shared location, you'll 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 configuration. Company-wide levels will be configured through `path.config`, and installation-wide *Default* level will become user-specific.

Modifying the default configuration levels

To get server installation with user-specific configuration, the entire configuration order should be redefined. Together has command line parameter `-config.path=<path to file which defines all levels>`. This option overrides hardcoded levels configuration.

To get the hardcoded 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.

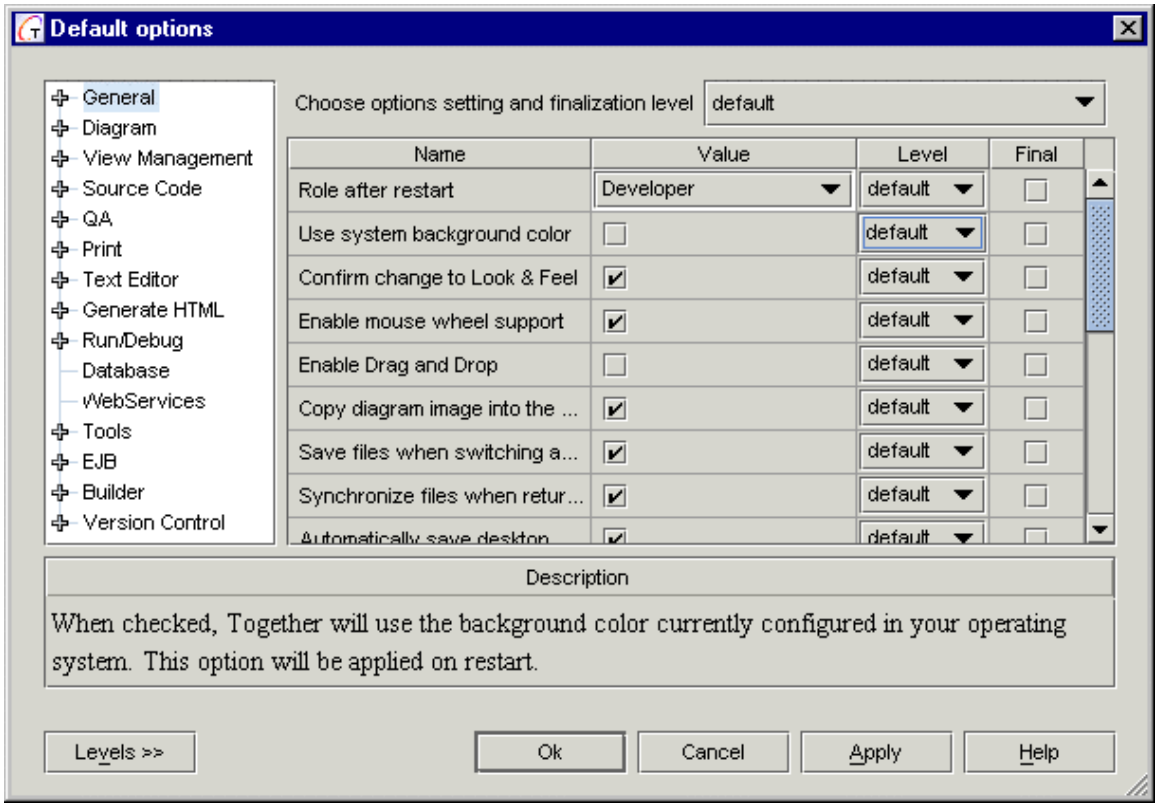
To make level visible in the Options dialog, you should add to config (e.g. misc.config) some more lines:

```
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.

Guide to the Options Pages

This topic summarizes the pages of the *Options* dialog and the types of customizations you can do on each page. Help texts for the pages and the individual options display directly in the *Options* dialog.



The full Options dialog for Default configuration level

Keep in mind that you can set the options on each page at any one of three *pre-defined configuration levels*. Be sure you know what level you are working on before you change any option settings. For more information see *Using the Options dialog*.

General

The General options enable you to customize a number of behaviors and appearances in the user interface. In this tab you can:

- Control background color and font properties
- Define user's role after restart
- Auto-synchronize files when returning from other apps (e.g. an 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 (folders with version control data for example).
- 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 e-mail exception reports to the Together development team.
- Control whether Together will search archives for diagram files

Diagram

The Diagram options allow to control a number of default behaviors and appearances that apply only to diagrams. In Diagram options you can:

- Control the default link routing method
- Control the alignment, layout, justification, and initial maximum width of classes etc. 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 display of the print grid in diagram pane.

View Management

View Management options let you control what your want to see and when. Specify how different kinds of elements display in the diagrams, or even whether they show up at all. With the View Management page you can:

- Control the general level of detail shown in diagrams.
- Control whether members in Classes display with UML or Java format (products with Java language support).
- Show or hide subpackage contents in diagrams.
- Control how Java Bean classes/C++ properties display in Class diagrams.
- Show or hide referenced classes in diagrams.
- Control the display of dependencies. **Note**: Showing dependencies slows down the performance.
- Show object class names and message numbers in Sequence diagrams.
- Control display of messages in Sequence diagrams.
- Control banned destinations
- Specify maximum call stack depth for Sequence diagrams generated from operations.
- Show or hide aggregations of diagram elements (including 2 user-defined) and EJB elements.

Source Code

In this page you can control a number of default behaviors and appearances that apply to the formatting of source code during forward and reverse engineering operations. In Source Code options you can:

- Specify the relative position of Attribute declarations and Operation declarations within Class declarations (i.e. Attributes first or Operations first).
- Control how link-attributes are handled when destination is deleted.
- Specify exactly how source code is formatted (indents, line breaks, spacing, etc.); also specify when code is reformatted.
- Specify the type of line separator for your OS.
- Specify language specific code formatting, optimization, and name referencing, and import statement options for the programming language(s) supported under your Together license.
- 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, etc.) 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

In this node you can specify the paths to the sets of audits and metrics, and define the applicable scope of Quality Assurance.

Print

In this page you can set a number of defaults that apply to printing diagrams, files and generated documentation. In Print options you can:

- Set default paper size or define a custom one (e.g. 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 etc.

Text Editor

In this page you can control a number of default behaviors and appearances that apply to the display of text in the Editor pane. In Text Editor options you can:

- Define the number of spaces inserted in text with 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.
- Choose your favorite External Editor.

Note that source code formatting is not customized on this page. Use the Source Code page.

Generate HTML

In this page you can control the inclusion/exclusion of various content in the output of the standard HTML documentation generation facility (Tools | Generate HTML). In Generate HTML options you can:

- Include or exclude author and version tags in generated output.
- Specify all JavaDoc settings.
- Specify which visibility levels of classes to include in generated HTML output.

> **Tip:** You can set these options "on the fly" as you begin the doc-gen process. From the Generate HTML dialog, click Options to display the Options dialog with only the Generate HTML tab visible. If you change any settings, they are used for the current doc-gen operation only and then discarded. Your Options settings for your configuration are not changed.

Run/Debug page

In this page you can customize your runner/debugger. In Run/Debug page you can:

- Define root directory of JDK for running and debugging applications from Together.

  Note: The default compiler is the Sun SDK, which is specified in the field "JDK Home". By default it points to TGH/jdk. If JDK is not a part of installation, users should specify another folder. Otherwise compile/make/run commands will not work.

- Define location of the sources and current working directory
- Specify the project's Run Configurations
- Specify settings for Debugger
- Specify settings for JSP

Database

The Database options enable you to customize a number of common database properties. Here you can define:

- Timeout that Together will be waiting for connection with DBMS.
- Control whether Together will replace tables after generating DDL.
- Specify whether to use quoting symbols for identifiers in the resulting DDL.

WebServices

This node is destined for the list of available application servers, which can be used to register the web services.

Tools

Together's file-based architecture makes it possible to work in conjunction with other file-based development tools... compilers, debuggers, IDEs, editors, etc. This node contains the external tool definitions for your configuration. The *External Editor* tool definitions are already pre-defined for you.

Unlike previous versions, JavaVM is not specified any more in the Tools options.

Shell definitions for several other tools are included in the dialog, which you can use to set up interaction with an IDE and other tools. You can edit various fields to point these definitions to appropriate tools on your system.

These user-defined tools plus those above should be adequate for almost all needs. If you find you need more tools definitions, it is possible to add more options to the Tools page. This is an Advanced customization that involves editing the `tool.config` file.

Menu commands for launching or interacting with the various tools are displayed on appropriate menus in the menu system.

You can issue Compile/Make command from the project, package and class menu in the project explorer, from a package and class in the physical class diagram, from the current file in the editor, and by a button in the builder pane.

EJB

The EJB page options allow to edit properties for Enterprise Java Beans. In this page you can:

- Select Application Server.
- Edit EJB suffixes that are used for EJB recognition.

Builder

In the Builder node you can:

- Choose performing compilation prior to run/debug.
- Reflect compiling process in the Status bar.
- Control the format of compiler output.
- Choose target folder for the generated makefile.
- Set compiler options.
- Specify maximum permissible number of compilation errors.

Version Control

This node is where you enable version control and specify version control integration. Here you can:

- Enable or disable version control.
- Specify default interactions such as getting files on project open.
- Specify what type version control integration to use (CVS or SCC).
- Set properties for CVS LAN and CVS Client-Server.

By default, everything is set up to integrate with CVS (which is installed with Together). If you use a SCC-compliant version system you can choose 'SCC' in the *Use* option.

> **Note:** To use a SCC version control system, *Together* must be running under Windows (NT/98/95) and Coroutine classes must be installed (normally auto-installed and configured by the *Together* installer for Windows.)

Options for Activatable modules

Some additional nodes add to the Options tree when appropriate modules are activated, such as Ant Runner or XPTest.

See also

Using the Options Dialog
Common Configuration Tasks

Using the Options dialog

The **Options dialog** is *Together*'s main configuration control center. You can use the dialog in normal or Advanced mode. Normal mode enables setting options for a single configuration level. Advanced mode surfaces the *multi-level configuration feature* and enables you to set options at different levels without re-invoking the dialog (see Using Advanced Mode below). The title of the Options dialog and content can vary depending on how you invoke it. For convenience, some menu commands display the dialog with only a subset of the available options. (See Invoking the Options dialog below).

The Options dialog presents several tabbed pages of configuration options that organize the options into categories. For an overview of the categories and the types of configuration tasks you can do on each page, see *Guide to the Options pages*.

Invoking the Options dialog

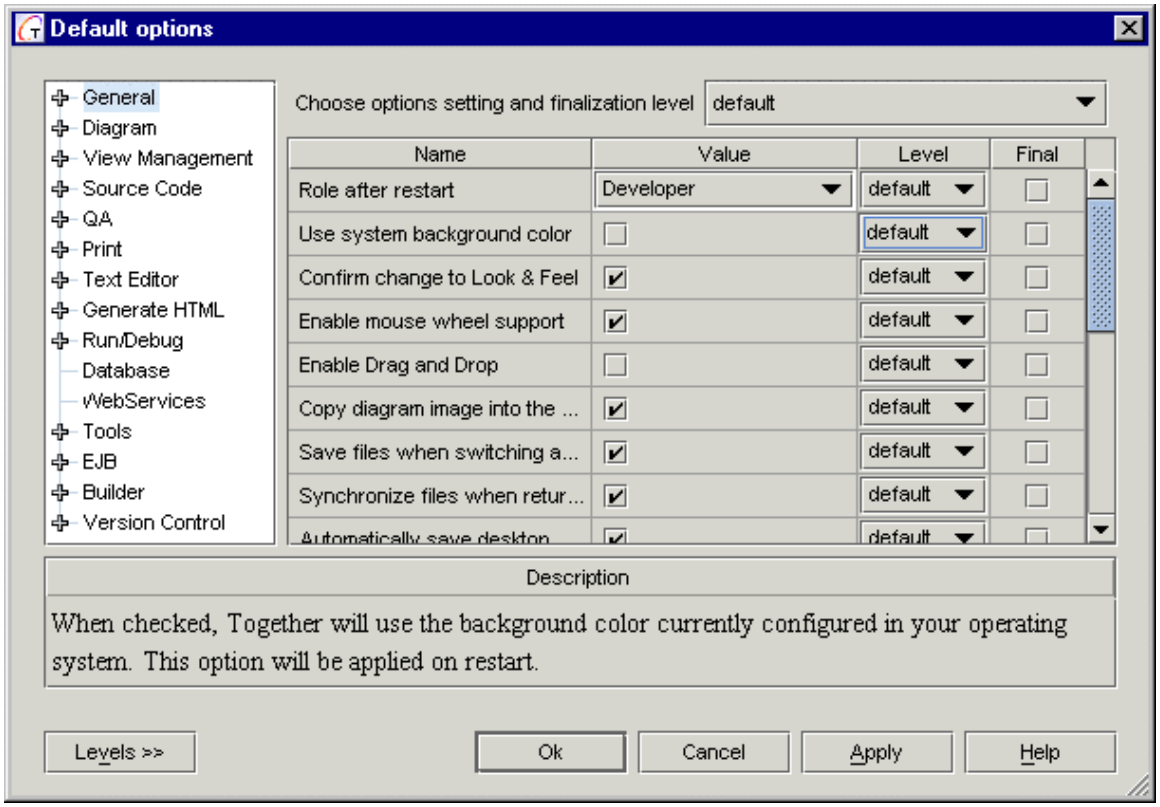
The Options menu enables you to invoke the full Options dialog for each of the configuration levels. It also has commands that present a subset of all available configuration options at a specified configuration level.

The following table shows the Options menu commands and function of each:

Command	Action
Default	Invokes the Options dialog and presents configuration settings at the *default* level. Command is always enabled.
Project	Invokes the Options dialog and presents configuration settings at the *project* level. Command is only enabled when a project is open.
Diagram	Invokes the Options dialog and presents the configuration settings available at the *diagram* level. Command is only enabled when a diagram is open and the Diagram pane is the active pane.
Default Tool Integration	Invokes the Options dialog and presents only the Tools page at the *default* level. Command is always enabled.
Diagram View Management	Invokes the Options dialog and presents only the View Management page at the *diagram* level. Command is only enabled when a diagram is open and the Diagram pane is the active pane.
Diagram Print Options	Invokes the Options dialog and presents only the Print page at the *diagram* level. Command is only enabled when a diagram is open and the Diagram pane is the active pane.
Text Editor Options	Invokes the Options dialog and presents only the Text Editor page at the *default* level. This command is always enabled.
Activatable Modules	This command presents a submenu of feature and integration Building Block modules that you can activate and deactivate by checking or unchecking. Keep in mind the more modules you activate, the longer Together takes to load. Some modules may impact performance as well.
Reload	Reloads the underlying configuration properties files. You only need to call reload if you do file-level customization of one or more of the properties files while *Together* is running.

Using Advanced mode

The *Levels* button of the *Options* dialog toggles Advanced Mode. In this mode, the drop-down list of finalization levels appears at the upper-right corner of the dialog, and the *Level* and *Final* columns add to the options pages. The Level column is an indicator/selector showing the currently defined configuration levels. The currently selected level is displayed.



The *Options* dialog in Advanced Mode at Default level

Level (in the page) 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

Final indicates that the option cannot be overridden at a level more local than the one indicated on the Level column.

Advanced mode enables you to set options at multiple configuration levels without re-invoking the Options dialog. You can change any settings that are not marked *final* at a "higher" (i.e. more global) level. In a local installation, you normally have complete control over your configuration and the settings at all levels. In a shared installation, you can change only those settings not marked as final by the system administrator in the shared configuration.

Setting options for multiple configuration levels

To access both the Default and Project levels:

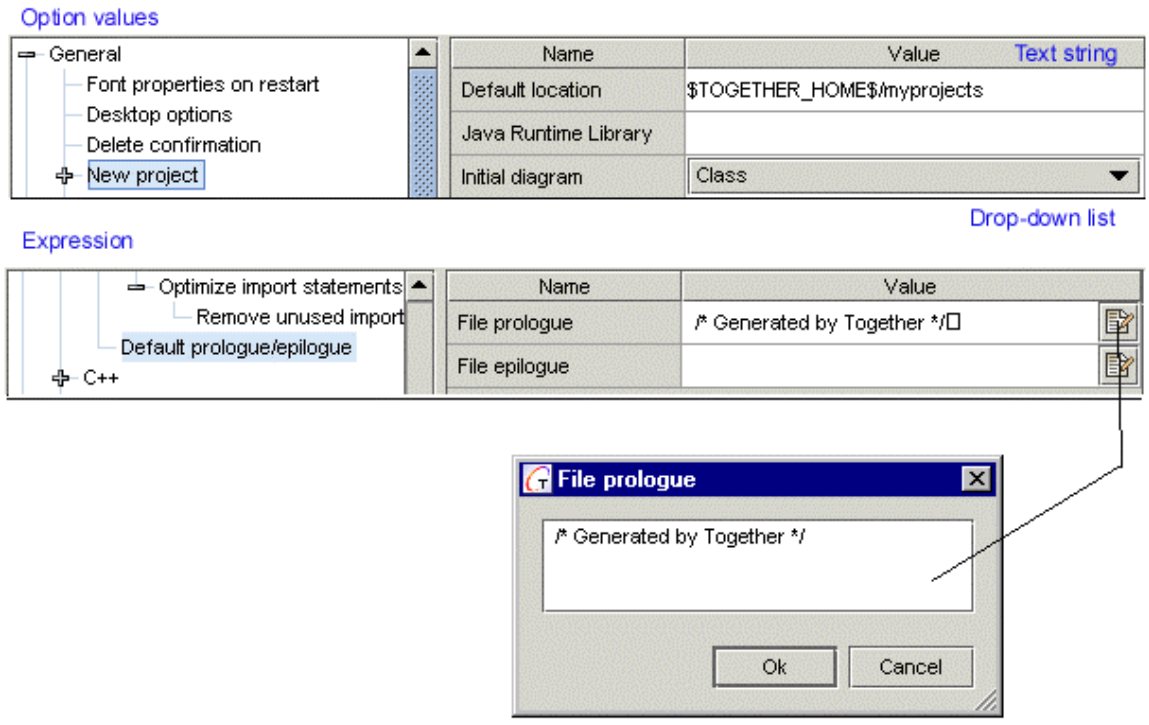
1. Open a project
2. From the Options menu, choose Project to display the Options dialog.
3. Click *Levels>>* to display the levels. Only *Default* and *Project* appear.
4. Set options at the *Default* level first, checking *Final* for all options that you do not allow to be overridden.
5. Set options at the *Project* level next, checking *Final* for all options that you do not allow to be overridden.
6. Click Apply to save changes as you work. Click OK to save changes and close the dialog.

To access all three levels:

1. Follow the steps 1-5 above.
2. Open the diagram(s) for which you want to set diagram-level configuration options. You can optionally open them one at a time, or several at once.
3. Select an open diagram in the Diagram pane and choose Diagram Options from the diagram speedmenu to launch the Options dialog.
4. Click *Levels>>*. The *Diagram* level is selected in the level selector/indicator.
5. Set *Diagram* level options as desired and click OK to close the Options dialog.
6. Repeat above for other diagrams as desired.

Using the Options editors

The Options dialog features new options editors displayed in a tree-like structure that logically groups the options. In cases where an option is a group of sub-options, the node expands to show all available sub-options. Different options have different editors depending on the type of its value. If the options are multi-valued, the values are comma - delimited. The illustrations below explains the basics of this interface.



Note the convention for check-boxes: checked = Boolean *true*, cleared = Boolean *false*. True means the option is active, engaged, or in effect. For example, checking the box for a Filter means the filter is active.

Tips and tricks

Individual options appear as nodes on an inspector tree in each tabbed page. Some options expand to show sub-options. If there are more options that can display in the page, a vertical scrollbar will appear.

Help for each page of the Options dialog, and for each of the options, displays directly in the dialog... no need for constant task switching. To see a general description of the page, click on its tab. To see Help for an individual option, click on the name.

Where the value of an option may be a **multi-line expression** (SourceCode | Code Templates, or View Management | Show | * | Expression for example), a browse button appears beside the edit field. This launches a multi-line editor.

For **checkbox options**, checked state is Boolean *true* or *yes*, cleared state is Boolean *false* or *no*. For example, in View Management's Show options, checking Show | All Members means *yes* (show).

You can **resize** the Options dialog. You can also change the width of the Name and Value columns of the dialog by dragging the separator between the column headings.

See also

Multi-level architecture overview
Guide to the Options pages
Common Configuration Tasks

Frequently Asked Questions on Common Configuration Tasks

Common configuration tasks

This section is intended to be a sort of FAQ for commonly needed customizations. Each topic in the section provides task-oriented specifics on how to do the specific configuration modification. Navigate through the table of contents to the desired task section. See Guide to the Options pages for additional information on the Options dialog customization facilities.

How to make Association links display a directional arrow

You can 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:

1. Launch the Options dialog.
2. Use *Levels* button to select the configuration level if necessary
3. Select the Diagram tab.
4. Expand the Associations node.
5. Set the value of *Draw directed* as desired.

   All - all links will be shown directed.
   No - all links will be shown undirected.
   Automatic - links represented by the attributes whose names start with "lnk", will be shown directed. All other links will be shown undirected.
6. Click 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" then the link will be always display according to that setting, regardless of the value set for this option.

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

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

1. Launch the Options dialog.
2. Use *Levels* button to select the configuration level if necessary
3. Select the Source Code tab.
4. Expand the appropriate language node.
5. Expand Default Prologue/Epilogue node.
6. Edit the default text to whatever you want. For example: "Generated for XYZ. Corp. Copyright (c)2001. Company confidential."

How to customize the default settings for C++

If you need to learn how to customize the default definitions for C++ source and header files, configure default library support, and more, refer to Languages support chapter that contains special notes related to configuration issues of using Together with C++.

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

This features allows to significantly speed up the coding process and avoid misspellings. See Defining Snippets in the Editor section of the User Guide.

How to configure Stereotypes

Customizing stereotypes is a low-level configuration task and requires Java programming. This allows to populate stereotype lists, modify default stereotype values, and specify RGB color values for stereotypes, etc. See Advanced Customization: Customizing Property Inspectors for more information.

How to customize Inspector properties

If you need to learn how to modify the property names and/or default values in properties Inspectors, add your own properties, or delete properties you don't want, refer to Advanced Customization: Customizing Property Inspectors for more information. Customizing inspector properties is a low-level configuration task and requires Java programming.

How to hide and show elements

To hide specific types of diagram elements from view:

1. Launch the Options dialog.
2. Click *Levels* to select the configuration level if necessary
3. Expand View Management node.
4. Expand *Show* node.

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.

For more information see View Management 'Show' options

Note that individual elements can be hidden in diagrams using the Diagram speedmenu. If you don't see an element in a diagram, and the element is not hidden by View Management options, choose Show Hidden from the Diagram speedmenu and check the hidden elements list.

You can define filtering expressions for the User-Defined options under the Show options of View Management page. Study the expressions in the pre-defined Show options to see how you can 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

To customize source code formatting options:

1. Launch the Options dialog.
2. Use Advanced to select the configuration level if necessary
3. Select the Source Code page.
4. Expand the node for Formatting Options and set sub-options as desired.

How to enable mouse-wheel support (Windows only)

Java VM 1.2, 1.3 doesn't 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 Options | [*level*] | General.
2. Expand General node and locate the *Enable mouse wheel support* option.

   If the option is checked, this feature is enabled.

   If unchecked, Together doesn't transform mouse wheel events.

   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:

1. Launch the Options dialog.
2. Use Advanced to select the configuration level if necessary
3. Select the Version control page.
4. Check or clear the *Enable version control* checkbox.

To enable version control for a project:

1. Click Advanced in the New Project dialog (if creating a new project) or the Project Properties dialog (for an existing project).
2. Check the *Version control project* box to enable version control.
3. Click Select. Select the source control project in the currently configured version control system (CVS or your SCC-compliant system).

For more information, including what Together files to place under version control, see Using Together with Version Control System.

Command line operations and macros

Command line parameters

Basic command-line syntax

```
Together_starter [Options] [ProjectFile]
```

where:

Together_starter	is a command that starts Together. On the Windows platform Together has the following launchers: `Together.exe` with disabled console output, `TogetherCon.exe`, that uses the existing console, or opens a new console for output. `Together.exe` provides dialogs for selecting parameters. Therefore, console version is preferable for automated doc generation. In some cases, when graphic output is advisable in the console version, you can use the option `-gui`. For other platforms, you might create `Together.sh, Together.cmd`, etc. depending on the operating system. Alternatively, this can be a complicated command that calls your Java VM, specifying parameters for it, followed by the Together main class name (`com.togethersoft.together.Main`) followed by any parameters for that class.

Options	are 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:
	
	`-script=com.togethersoft.modules.helloworld.HelloJava`
	
	or
	
	`-script:com.togethersoft.modules.helloworld.HelloJava`
	
	Note: Use colon (`:`) symbol under Windows, since "`=`" is incorrectly processed by the Windows command line interpreter.

ProjectFile	is the 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.

Using the Windows launcher

The installer for Windows platforms installs the Windows-specific launcher `Together.exe`. The launcher invokes the registered Java virtual machine, prepares a command line for it, and passes any parameters in the input command that don't pertain to itself on through to the main Together class.

For complete parameters of `Together.exe` see Together.exe parameters.

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 `$TOGETHER_HOME$/bin`. Use these files directly, configuring as necessary for your system, or use them as models to create your own launcher files.

Parameters of Together's main class

<Project.**tpr**>	specifies the fully qualified name of Together project file that will automatically open on start-up.
-config.path:<path>	Tells Together to search <path> for configuration properties instead of the default hard-coded location. For information on usage see Creating a shared multi-user configuration
-**script:** <module name>	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.
-**version**	Types Together version and build number, and exits. *
-help	Types parameters of Together's main class, and exits. *
***** This option is only used if console is enabled.	

Command-line examples

Standalone Formatter

You can format the source codes of your projects externally, using a standalone formatter. The command files for standalone formatter reside in the relevant folders under %TGH%\bin. For the Windows platform, `Formatter.bat` file resides in 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.

Running modules on 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 Hello_World module does not access project information. Note that Java files and Java classes should be specified with the proper case of letters (Java is the case-sensitive language).

```
cd %TOGETHER_HOME%\bin <Enter>
TogetherCon -script:com.togethersoft.modules.helloworld.HelloJava
samples\java\CashSales\CashSales.tpr
```

Generating doc from the command line

You can run HTML or RTF documentation generation module from the command line as part of an automated build process, using `Together.exe, TogetherCon.exe`, `umldoc.exe` (for Windows users only) or `Together.sh` (for Unix users only).

`Together.exe` launches Together shell and brings you through the entire documentation generation procedure. The main intention behind using `TogetherCon.exe` and `umldoc.exe` was to be able to produce documentation without opening Together windows.

When you run `Together.exe` or `TogetherCon.exe` in the command line mode, you can select between generating HTML or RTF documentation, specifying the appropriate module name. `umldoc.exe` utility allows to generate HTML documentation only.

Launching the documentation generation module:

```
<launcher> [module] [options] <project>
```

where:

<launcher> is `Together.exe, TogetherCon.exe or umldoc.exe`

[module] is a command for the specific utility.

`-script=com.togethersoft.modules.genhtml.GenerateHTML` for generating HTML documentation;
`-script=com.togethersoft.modules.gendoc.GenerateDocumentation` for generating RTF documentation;
Note: This parameter is not required for `umldoc.exe`, since it executes generateHTML module only.

[options] are the ones for the command-line GenerateHTML module launcher (`$TOGETHER_HOME$\bin\win32\umldoc.exe`) or for GenerateDocumentation. The options are listed below in the tables.

<project> is the path to the project on which to generate documentation;

Options for umldoc.exe

Option	Description
`-overview <file>`	Read overview documentation from HTML file
`-public`	Show only public classes and members
`-protected`	Show protected/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 Together.exe, since it initiates the Together documentation generation dialog.
`-use`	Create class and package usage pages

Option	Description
-version	Include @version paragraphs
-author	Include @author paragraphs
-splitindex	Split index into one file per letter
-windowtitle <text>	Browser window title for the documentation
-doctitle <html-code>	Include title for the package index(first) page
-header <html-code>	Include header text for each page
-footer <html-code>	Include footer text for each page
-bottom <html-code>	Include bottom text for each page
-nodeprecated	Do not include @deprecated information
-nodeprecatedlist	Do not generate deprecated list
-notree	Do not generate class hierarchy
-noindex	Do not generate index
-nohelp	Do not generate help link
-nonavbar	Do not generate navigation bar
-stylesheetfile <path>	Cascading stylesheet file to control appearance and formatting of the generated documentation
-togethersettings	Use settings from config/GenerateHTML.config file
-recurse	Create output for packages specified in [package names]and their subpackages
-javadoc	Create the same output as javadoc.exe produces
-browser	Launch HTML browser
-diagrams	Create diagrams' pictures
-navtree	Generate Navigation Tree
-nopackagedependencies	Do not show package dependencies in all class diagrams

Options for gendoc

Each GenDoc option can be either "switch option" or "parameter option". Switch options are represented as

-option_name

Parameter options should be followed by parameter values

-option_name parameter_value

There are two option types in GenDoc: the *Regular options* and the *Custom options*. Regular options allow to specify the required parameters for the Doc Generator, such as template file, output directory, etc. GenDoc recognizes a fixed number of regular options.

Regular options

Option	Description
`-template <path>`	template file. if omitted, the default 'ProjectReport' template will be used: `..modules\gendoc\templates\ProjectReport.tpl`
`-metamodel <path>`	meta-model file; if omitted, the default meta-model file will be used: `..modules\gendoc\templates\MetaModel.mm`
`-format <RTF | HTML | TXT>`	output format; RTF assumed by default.
`-styletemplate <path>`	style template file (depends on the output format, e.g. `*.dot` file for RTF output)
`-d <directory>`	output destination directory
`-f <path>`	output file path. This option allows to redirect all GenDoc output to the specified file. The option is compatible with TXT output format only.
`-diagrams`	include diagram charts
`-recurse`	create output for packages specified in [packagenames] and their subpackages

Custom options

Any options that are not recognized as the regular options, are considered custom options. There is no fixed list of custom options and they are not directly processed by the Doc Generator. A custom option should always have a parameter. *GenDoc* stores all passed custom options and their values. Subsequently, when a particular template is processed, the value of any custom option can be obtained by name within the template, using the function `getDGOption(String optionName)`.

This feature allows to pass parameters to a template from command line and adjust the template behavior dynamically.

See also

Generating Project Documentation

Parameters for 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`'s parameters by running it from the Windows Command prompt using the **-h** option.

Parameters for VM preferences

Note the presence of parameters **-builtin** and **-sun13**. These enable you to specify the preferred 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.
USAGE: `Together.exe [options] [-c[class_name]] [parameters]`
Valid options are:

Option	Description
-?,-h,-help	print this usage message and exit
-con,-nocon	how/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
-cp:p <path>	prepend <path> to classpath
-cp:a <path>	append <path> to classpath

-nojit	disable JIT compiler
-verify,-noverify	verify/don't verify classes when loaded
-verifyremote	verify network loaded classes (default by VM)
-builtin	prefer built-in Java VM (Sun JDK) to other Java VM
-sun13	prefer Sun JDK 1.3 VM to other Java VM. **Note:** Together supports JDK 1.3 only!
-nosystemcheck	don't check system memory size
-c<class_name>	class name to run (default com.togethersoft.together.Main)

The rest of command line after `-c` or unknown parameter prefix is `class_name` or its parameters. By default, JDK 1.3 (installed with Together on Windows platforms) is used.

EXAMPLE:

```
Together -sun13 -ms16m param1 param2 param3
```

See also

Command Line Parameters

System macros

System macros are shorthand notations for lengthy path specifications that you may need to use for configuration, scripting or other tasks. *Together* knows how to expand the shorthand and make the proper reference.

You will most likely need to use macro references when doing customization tasks in the Tools tab of the Options dialog. Specific references to usage there are noted in blue font.

Macro	Description
TGH	contains the full path to your Together's installation. For example (c:\Together5).
$TOGETHER_HOME$	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(Windows) or a colon(UNIX). For example: `$SYSTEMJVM$;c:\MyClassesDir (Windows)`
$CLASSPATH$	contains the value of the system's CLASSPATH environment variable.
$CLASSPATH_JVM$	Contains the path to the rt.jar file.
$CLASSPATH_PROJECT$	contains the paths to all the packages in the project.
$SOURCEPATH$	contains the paths to all the writable packages in the project.
$DESTINATION$	contains the value of the Destination option in the Tools tab of the Options dialog (this value is defined in the "build.destination" property located in the tool.config file)
$MAINCLASS$	contains the name of a class in the project, defining the "public static void main (String[])" method. If there is no such class, this macro contains an empty line ("").
$LINENUMBER$	contains the line number of the selection in the file containing the selected element. For example, for selected operation this macro will contain the line in the file with the operation's class.
$CLASS_NAME$	contains the name of the selected class (fully qualified name in Java).
$PROJECT_DIR$	contains the full path to the project's directory. For example, if the project is located in the `c:\together\myprojects\CoolProject` directory, then this macro will contain the value `"c:\together\myprojects\CoolProject"`
$PROJECT_NAME$	contains the name(without extension) of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain "myproj".

Macro	Description
$PROJECT_EXT$	contains the extension of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain "tpr".
$PROJECT_FULLNAME$	contains the name (with extension) of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain "myproj.tpr ".
$PROJECT_SPEC$	contains the full name of the project file (path, name and extension). For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain the value: "c:\together\myprojects\CoolProject\myproj.tpr".
$FILELIST$	contains the "@somefile", where `somefile` is the name of the automatically generated file with the list of the files (each file on a new line) contained in the selection. (If a user clicks on a class, `somefile` will contain only the file with the selected class, but if a diagram is clicked, `somefile` will contain all the files that represent classes in the diagram, and all the classes in subpackages shown on this diagram, etc.)
$FILE _DIR$	contains the full path to the selected file's directory. For example, if the selected file is located in the c:\together\myprojects\CoolProject directory, then this macro will contain "c:\together\myprojects\CoolProject"
$FILE_NAME$	contains the name (without extension) of the selected file. For example, if the selected file is `c:\together\myprojects\CoolProject\MyClass.java`, then this macro will contain "MyClass".
$FILE_EXT$	contains the extension of the selected file. For example, if the selected file is `c:\together\myprojects\CoolProject\MyClass.java`, then this macro will contain "java".
$FILE_FULLNAME$	contains the name(with extension) of the selected file. For example, if the selected file is `c:\together\myprojects\CoolProject\MyClass.java`, then this macro will contain "MyClass.java".
$FILE_SPEC$	contains the full name of the selected file (path, name and extension). For example, if the selected file is `c:\together\myprojects\CoolProject\MyClass.java`, then this macro will contain "c:\together\myprojects\CoolProject\MyClass.java".
DEF_FILE_DIR	contains the full path to the selected definition file's directory (C++ only). For example, if the selected definition file is located in the `c:\together\myprojects\CoolProject` directory, then this macro will contain "c:\together\myprojects\CoolProject". Definition files has .cpp extension
DEF_FILE_NAME	contains the name (without extension) of the selected definition file (C++ only). For example, if the selected definition file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain "guest".

Macro	Description
DEF_FILE_EXT	contains the extension of the selected definition file (C++ only). For example, if the selected definition file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain "cpp".
$DEF_FILE_FULLNAME$	contains the name (with extension) of the selected definition file. For example, if the selected definition file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain "guest.cpp".
DEF_FILE_SPEC	contains the full name of the selected definition file (path, name and extension) (C++ only). For example, if the selected file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain " c:\together\myprojects\CoolProject\guest.cpp".
$PROMPT$	Displays a dialog with a text input field and, when you press OK, returns the entered value. If you press CANCEL, the command is cancelled. **Variants:** $PROMPT:name=placeYourLabelHere$ Same as $PROMPT$, but you can specify the name (a string after the equal sign) of the label above the input field. $PROMPT:name=placeYourLabelHere,default=placeDefaultValueHere$ Same as above, but allows you specify the default value of the input field. If you press CANCEL, the default value will be returned. Note: If there are several $PROMPT$ macros, then only one dialog will be displayed, containing fields specified in these macros, for example: $PROMPT$$PROMPT:name=MyLabel$_someString_$PROMPT$ will display a dialog with three input fields, and when the user presses OK, will return a string containing : AB_someString_C, where A,B,C are the entered values.

Referencing configuration properties as macros

Besides using the predefined macros (described above), it is possible to use any of the properties defined in the *.config files. In this case you must 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$`. For all other properties you must specify their full name.

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 with % characters, are used in the template values for both Forward (code-gen) and Reverse (parser) engineering.

All languages

Macro	Definition
%Class_Name%	This macro doesn't show up in the Choose Pattern dialog. For the classes, it is substituted with the name of the class to which this template is applied (e.g. for class constructors). For the members, it is substituted with the name of the class, where a member is created by this template. **Note:** can't be used in prologue/epilogue properties
%FILE_NAME%	Name of the class source code file. **Note:** used for generating prologue/epilogue only.
%FILE_EXT%	Extension of the class source code file. **Note:** used for generating prologue/epilogue only.
%Name%	Name of a generated class / attribute / operation, editable in the Choose Pattern dialog
%Dst%	Name of the destination class of a generated link, editable in the Choose Pattern dialog
%Type%	Type of attribute or return type of operation, editable in the Choose Pattern dialog

C++ Only

Macro	Definition
%Header_File%	The header file path

In addition to the above macros, parser can use the **%Any%** macro that matches any token.

Notes:

1. Case of letters in the file macros (`%FILE_NAME%`, etc.) controls 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".

2. 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.

3. Templates replace blueprints of the pre-4.0 versions. Together recognizes Blueprints from earlier versions, and will for several more releases. However, it is strongly recommended to converse to Templates, as they are more efficient.

Keyboard shortcuts

Together provides a keyboard interface for most commonly-needed tasks. The standard keystrokes are documented in this topic. 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.

Main Menu

Explorer

Expand node	Right arrow, ENTER, double click
Collapse node	Left arrow, ENTER, double click

File

New project	CTRL+SHIFT+N
New diagram	CTRL+N
Open project	CTRL+SHIFT+O
Save	CTRL+S
Save all	CTRL+SHIFT+S
Close	CTRL+W
Close all	CTRL+SHIFT+W
Print Diagram	CTRL+P
Exit	ALT+X

Edit

Undo	CTRL+Z, ALT+BACK_SPACE
Redo	CTRL+Y, CTRL+SHIFT+Z
Cut	CTRL+X, SHIFT+DELETE
Copy	CTRL+C, CTRL+INSERT
Paste	CTRL+V, SHIFT+INSERT
Delete	DELETE
Go to line	CTRL+G

Search

Find	CTRL+F
Replace	CTRL+H

Search next / previous	F3 / SHIFT+F3
Search in files	CTRL+SHIFT+R
Search by query	CTRL+SHIFT+Q

View

Toggle explorer pane	F6
Toggle editor pane	F9
Toggle diagram pane	F10
Toggle message pane	F11
Full screen view	F12

Select

Next Pane	CTRL+F12
Previous Pane	CTRL+SHIFT+F12
Next Tab	ALT+RIGHT
Previous Tab	ALT+LEFT
Open & select explorer pane	CTRL+F6
Open & select editor pane	CTRL+F9
Open & select diagram pane	CTRL+F10
Open & select message pane	CTRL+F11

Help

Contents	SHIFT+F1

Diagram shortcuts

Deselect element	ESC
Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Invoke speedmenu of the selected element	SHIFT+Right Click
Auto layout all elements	CTRL+K
Update	F5
Paste reference	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

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	/

Class diagram

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Class

New attribute	CTRL+A
New operation	CTRL+O
New member by pattern	CTRL+T
New property	CTRL+B

Package

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Attribute, Operation, Property, Statechart, Activity diagrams

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

Compile and Run/Debug shortcuts

Make	F7
Rebuild	CTRL+SHIFT+F7
Run configuration dialog	SHIFT+F10
Run application with the current parameters	CTRL+F5
Run application with the parameters dialog	CTRL+SHIFT+F5
Debug application with the current parameters	SHIFT+F9
Debug application with the parameters dialog	CTRL+SHIFT+F9
Attach to remote process	SHIFT+F5

Editor shortcuts

Insert snippet	CTRL+J
Invoke Code Sense	CTRL+SPACE
Insert Bookmark	CTRL+M
Invoke Bookmark Dialog	CTRL+D
Insert Numeric Bookmark	CTRL+SHIFT+number
Navigate to Numeric Bookmark	CTRL+number
View method parameters	CTRL+F8

File Chooser

Invoke speedmenu in the file chooser dialog	SHIFT+F10

Version Control

System	CTRL+Q

Navigation shortcuts

Switching between open panes	CTRL+F12
Switching between open panes back	CTRL+SHIFT+F12
Switching between Explorer's tabs	ALT+ RIGHT/LEFT ARROW
Move to the splitter bar	Alt+F8
Open next openable message	ALT+F10
Open previous openable message	ALT+F9

Projects and Project Management

Project basics

To begin modeling with *Together* you need to create a *project*. At minimum, a project consists of:

- a *primary root* directory
- a *project file* (*.tpr* file extension and  icon)
- a default *package diagram*

Primary root directory

The primary root directory stores the project file and any initial diagram created along with the project. It also provides a storage location for project-level configuration properties files (see Multi-level Configuration).

When you create a project, the primary root can be either a new or an existing directory. By default, *Together* round-trip engineers any source code it finds in the primary root directory and any subdirectories below it. If you specify the top-level directory of an extremely large code base as the primary root, you may find the reverse engineering process too slow. In such cases, you can redefine how project resources are parsed. See the topics Creating and opening a project and Large projects.

Project file

The project file gets the name of the project as the filename and a **.tpr** extension. It displays the  icon in the Explorer.

The <default> diagram

When you create a new project, *Together* generates a diagram that presents a view of 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 etc. of any source code files found in the primary root.

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

Note that there is no *default* diagram that opens every time you open a project. The diagrams that open along with a project, if any, are controlled by the configuration settings under Desktop Options (General node of the Options dialog; see Configuring Together). If properly set, Together remembers what diagrams were open, when the project closed, and re-opens those diagrams when the project next opens.

Project content

The scope of a project is not limited to a single root directory – you can specify multiple directories as being project root directories, include or exclude subdirectories of any root, and exert some initial control over how these directories are treated during round-trip engineering. You can include individual Zip and Jar archive files in the project as well. You do this using the *Advanced mode* of the New Project dialog (File | New Project). For more information, see Creating and opening a project.

Creating and opening a project

This section explains how to open an existing Together project and create new projects. If you are new to Together, please read Project basics before you create or open a project.

Opening existing projects

To open an existing Together project:

1. Start Together if not already running.
2. From the Main menu, choose **File | Open Project** to display the file selection dialog for your operating system.
3. Navigate to the primary root directory containing the project (.tpr) file.
4. Select the project file (e.g. myProject.tpr) and click Open or Save (depending on OS)

Together opens the project and displays the contents in the Model tab of the Explorer. If the *Saved Desktop* option is enabled in your configuration options, any diagrams that were open, when the project was closed, re-open in the Diagram pane.

Tip: You can also navigate to a project file in the Directory tab of the Explorer and double-click to open it (or use the speedmenu).

New projects

You can create a new Together project "from scratch", from an existing code base, or some of both. This section describes the process of creating a project manually, so that you understand the various things that must be specified. You can optionally use the New Project Expert (File | New Project Expert) as a step-by-step guide to creating a project.

To create your very first project you might consider using the **New Project Expert.**

New projects "from scratch"

When there is no existing code to reverse engineer, project creation is quite simple. In many cases you need only specify a primary root directory for the project. You can do this in the default mode of the New Project dialog. If you decide you want the new project under version control, or if you want any external resources (headers, libraries, etc.) to be available to the project, you'll need to specify this in the dialog's Advanced mode (see Advanced Mode below).

New projects using existing code

When creating a Together project for an existing code base, think first about what you want as *modifiable* content, and what you want as *read-only content*, i.e., shown in diagrams but not modified from within Together. Also think about what content should be round-trip engineered.

For example, in a Java project where some of your classes extend Java classes or some component classes, you might want to show those dependencies in your visual model, but you would not modify the parent classes. You might also want to include some or all of the classes residing on your Java classpath in your diagrams, or you might want to see classes or diagrams from another Together project but not modify them in the new project. If you compile classes from inside 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, you can specify whether or not to allow modification. If you want classes form a physical directory but not including all of the subdirectories under it, you can exclude specific subdirectories.

Together provides this kind of flexibility when creating new projects, and when modifying existing projects. You can add or remove project resources as needed (without deleting any physical files), and you can control what resources can be modified within the context of the project. You will find this especially useful when dealing with extremely large code bases with dozens of directories and hundreds of classes. (See Project Management: Large Projects).

Basic and Advanced dialog modes

The New Project dialog offers two modes: **Basic** (the default mode) and **Advanced**. Use basic mode when...

- all the resources for the new project will reside in and under the directory where you will create the project, *and*...
- you want all resources included in the project, and...
- you want all resources to be modifiable and round-trip engineered.

Use Advanced mode to...

- specify multiple resource roots (directories or archives)
- add or remove resource roots (directories or archives)
- exclude subdirectory or file types in resource roots from parsing
- control modifiability of content in resource roots
- set up version control for the project

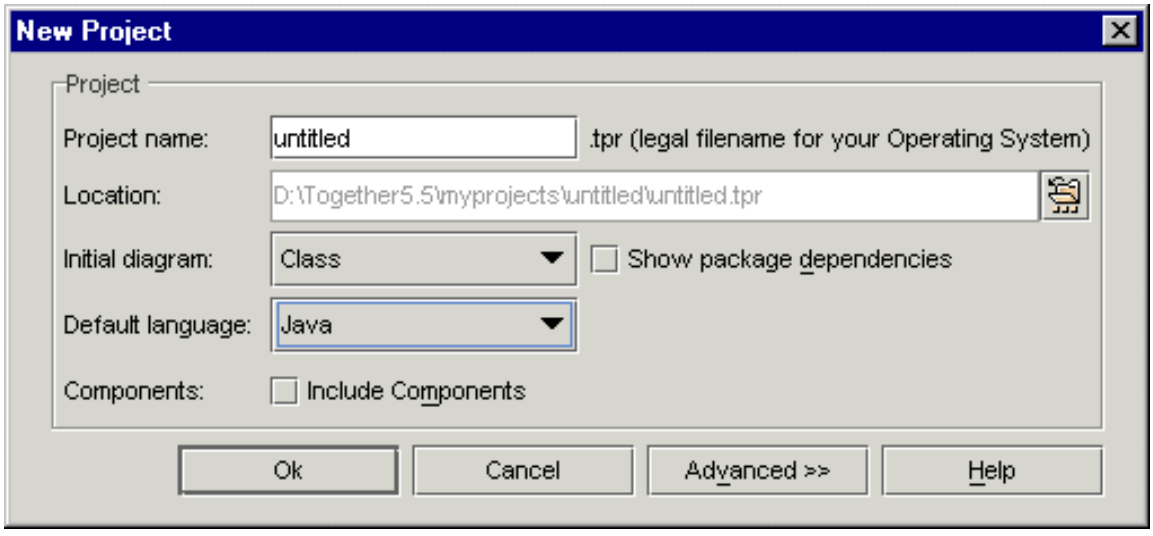
Creating a new project

To begin creating a new project:

1. From the Main menu choose **File | New Project** to display the New Project dialog. The dialog displays in Basic mode.

Using basic mode

The basic mode of the New Project dialog is fairly self-explanatory.



2. Enter a name for the project in the Project Name field. This should be a **legal filename** for your operating system. The Location field now displays a default path. This is where *Together* will create the project file and initial and default diagrams. If you want to change the location click Browse and navigate to an existing directory where you want to store the project file and default project diagram. You can also use the dialog to create a new directory for the project (on most OS platforms).

3. If necessary, use *Browse* to specify the location of the project directory. This can be any existing directory, or you can use the dialog to create a new one. Remember that basic mode's default means everything in and under the project directory is at least *reverse* engineered, and is modifiable (unless it's read-only at the OS level).

4. If you want to create an initial diagram (other than the default <class> diagram) along with the new project, select the type in the *Initial diagram* list. Otherwise, select *None*.

5. If you have a multi-language product, specify the target programming language for the project in *Default Language*. When you choose a language, any language-specific options available display next to the language selection.

6. The project now contains one root directory- the primary root as specified in steps 1 and 2. If you don't need to include any other directories as part of the project, click OK to create the project. If you want to specify other directories as part of the project or remove some added with language options, proceed to Advanced mode.

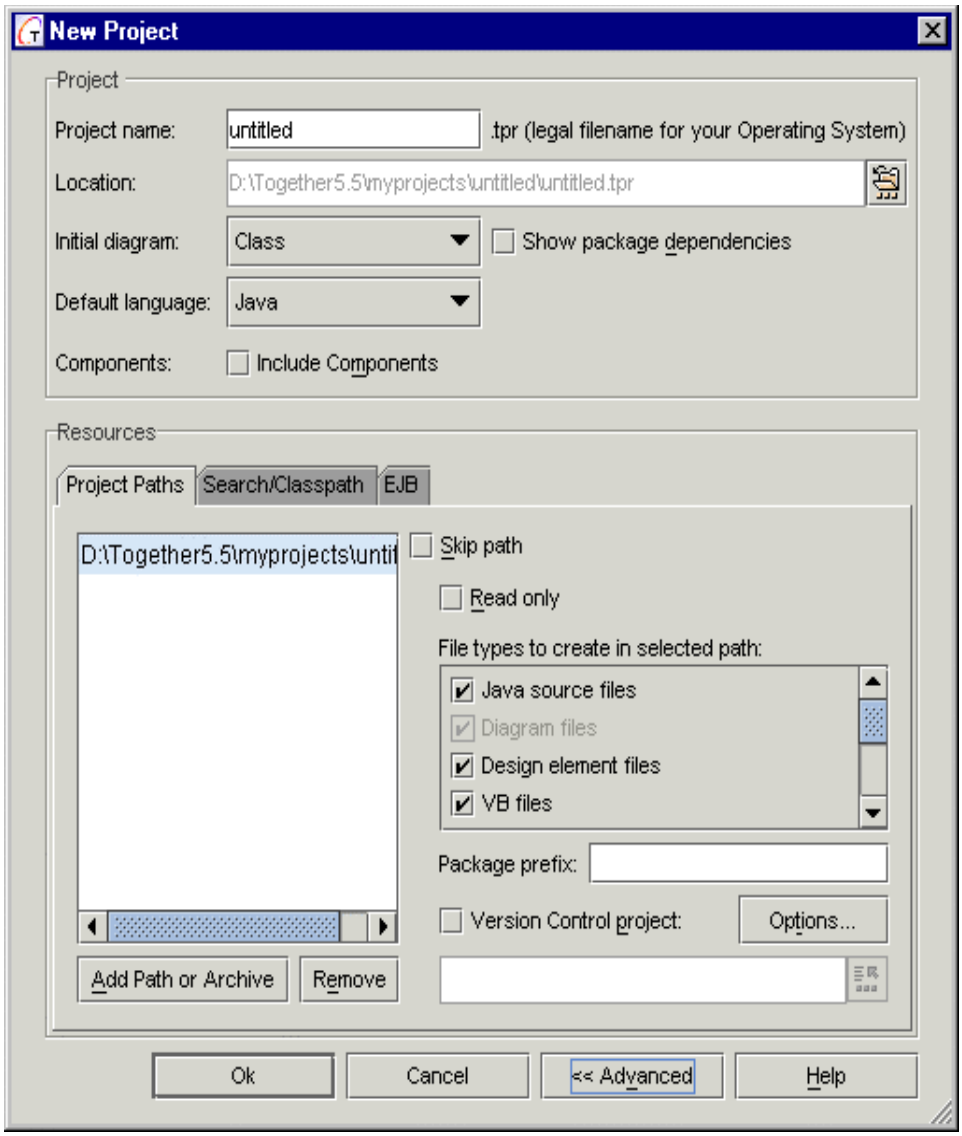
Checking the Components checkbox will include Coad Components into the project. Fully qualified path to the Coad Components appears in the Search/Classpath tab of the Resources pane (advanced mode).

Show package dependencies checkbox enables to automatically draw dependency links between packages. However, this can slow down performance for the large projects. You can rescan the project at suitable time using *Update Package Dependencies* command of the diagram speedmenu.

If you clicked OK at this point, the project file with the filename you specified and the initial diagram (with the same name) are created in the specified directory (called the *project directory*). The project file has a .tpr extension and displays with the  icon in the Explorer. At a later time, you can modify the project, adding additional resource roots, etc. from the Project Properties dialog (File | Project Properties).

Using Advanced mode

The New Project dialog's Advanced mode lets you control what resources are available to, and included as part of, the new project and how these resources are treated during round-trip engineering. You can also specify which project in your configured version control system to use for source files that are part of the new Together project.



To use Advanced mode:

1. Follow steps 1-4 in Basic Mode above.
2. Click the Advanced button in the New Project dialog to toggle Advanced mode.

Adding or removing resources

The *Project Paths* and *Search/Classpath* tabs display the lists of the directories and/or archive files currently included as resources available to the project. By default, the path specified in *Location* (near the top of the dialog) is present in the *Project Paths*, and the paths to the standard libraries are present in the *Search/Classpath* list. Checking "Include Classpath" adds your classpath directories to the list. For editions that include components, the Include Components box is enabled and checking it adds the component directories to the list.

Items in the *Project Paths* list are considered as *project roots*. Roots contain compiled or source classes and/or Together diagrams. **Project roots are parsed** during reverse engineering and treated as modifiable unless you specify otherwise in the other controls (see *Resource options* below).

The *Search/Classpath* list 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:*

1. Click the Add Path button to display the Select Path dialog.
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 them to be skipped over by the parsing engine, but still tracked with the project by Together, you can add the individual subdirectories with the Add Path button, as described above, and then choose the radio button **Skip path** (see *Skip path* section below).

- If you want some subdirectories to be excluded completely, use Ignore Files and Folders option on the General page of the Project Options dialog (Options | Project) to specify the ignored directories.

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

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

To **add** *an* **archive file** *as a project resource:*

1. Select either the Project Paths or Search/Classpath tab of the Resources section of New Project (or Project Properties) dialog
2. Click the Add Path or Archive button to display the Select Path dialog.
3. Navigate to the directory containing the archive file.
4. Select the desired file.
5. Click Open or Select, depending on the OS you are running on.

Removing resources

You can remove any resource from the Project Paths list or the Search/Classpath list by selecting it and clicking the Remove button. This is useful if, for example, you included your full Java classpath but that 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, you can add them later using File | Project Properties (Advanced mode).

Setting Resource options

After you have added resources to the project you can set some 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. If that default behavior is not what you want, you can 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:

Access: Selecting this option gives the parsing engine access to the selected resource.

> **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 are disabled.
>
> **File types:** 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 have a multi-language product version and 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 new/current 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.

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.

Project management

Together is designed to minimize or even eliminate the need for extensive project management. Once you have defined the project structure, created the necessary project file(s), customized your configuration options at all necessary levels, and set up your tools and version control integration, there's really very little else you need to do on an ongoing basis. There's no need to continually update a repository before code is generated, because *Together* generates code and synchronizes with the model as your team works.

As the project administrator, you will want to make sure that all your modeling files and all newly created files get placed under source control. (For information, see *Common Customizations: Version Control Integration:* Together files to include in version control).

You might also consider developing modules to run *Together*'s documentation generation (via the command-line interface) and integrate this as part of a periodic automated build process. For more information see Reference: Command Line parameters and Together API.

Setting up large projects

What constitutes a "large" project? A good thumbnail definition might be "any project that takes too long to reverse engineer". You just can't get around the basic laws of computer science: even *Together*'s legendary parsing engine needs instruction cycles and system resources to parse your code. The more code you have, the more resources it takes, and eventually you hit a point of diminishing returns. The good news is that with *Together* you can employ some simple project management techniques that will help you avoid such problems.

When it comes to extremely large code bases, remember the old adage that says, "*How do you eat an elephant? One bite at a time.*" If you have a huge project, you don't deal with it all in one chunk. You divide it up into subsystems, and perhaps modules or components within those subsystems. In *Together*, that translates to **projects and subprojects.** Don't attempt to create a single project that encompasses your entire code base. Instead, identify the subsystems and the modularity within them and create a number of *Together* projects in key directories of particular interest or significance. For an example of how this might be done, look at the `/component/CoadModelingComponent` directories under your installation. Although this is not a particularly large code base, it is illustrative of the project-subprojects technique you can employ for massive projects.

You should also consider automating documentation generation for very large projects as described above. You can write a script or module to regenerate all documentation, or a set of scripts to update different parts, which run automatically on different nights.

Creating views with referenced content

With the project definition features of Together you can create projects whose content is purely logical and contains only things you need to see for a particular purpose. For example, you could create a directory that is strictly for views of your code base. Under it you can create a series of project folders that contain *Together* projects that bring in different parts of your code... only abstract classes in your problem domain for example.

When creating such projects, you can specify as resource roots only those specific subdirectories you are interested in. Other directories with classes etc. you might want to show (but not modify) can be specified in the Source/Classpath list. (For more information, see Creating and Opening a Project.)

Performance tuning

Here are several things you can check before tackling large code bases.

- Check the resource paths in Project Properties (File | Project Properties). Deep path names like "C:" will slow things down.

- Look at the memory used while parsing; if it goes close to the maximum, then try to increase the swapping file size (set the minimum swap file size to something like 100-150M).

- Try running *Together* with jre.exe (Sun JVM). It provides options to specify the Java heap size. Change the maximum heap size (`-mx32m`) to some larger amount (e.g. `-mx60m` or `-mx100m`, depending upon how much virtual memory is available). Make sure you have computer hardware with sufficient power and system resources to handle the values you specify; otherwise the effect may be the opposite of what you want.

Integrating a project with Version Control

Even when version control is enabled in *Together* configuration options, you must still specify a version control system project for each *Together* project.

- For new *Together* projects, use the New Project dialog, Advanced mode.

- For existing projects use the Project Properties dialog (File | Project Properties), Advanced mode.

In both cases, check the Version Control option and specify the version control system repository.

For more information consult these topics

Version Control Integration
Guide to the Options pages: Version Control

Information Import-Export

Import-Export operations

This topic describes the information import and export features of Together. You must obtain and install a Together product with support for the various types of import-export operations in order to use the features. (See Where to Get Help for on-line resources for current product information.)

You can export information from your Together model in several ways:

Export Class or ER diagrams, or EJBs to DDL files for import into JDBC compliant databases
Generate Interface Definition Language (IDL) for a project.
Export model information to XMI

You can also import several types of information into Together:

Import Rational Rose model files (.mdl)
Import from JDBC databases
Import from XMI
DTD / XSD Import-Export
DTD interchange of class diagrams and database structures

Rational Rose Import

This section explains how to import files created with Rational Rose™. You can import Rose's .mdl files to create Together projects. Model files must be from Rose 98 version 4.2 or later. If you have older Rose models in a format previous to Rose98 version 4.2, you must convert them to the later format using an evaluation or regular version of Rose.

Importing a Rose model to Together

Import must take place within the context of a Together project. For maximum efficiency, especially when importing large models, close other applications to free up memory.

To import a Rose model:

1. Open or create a Together project
2. On the Main menu, choose Tools | Import | Import from Rational Rose. The Open File dialog for your system displays.
3. Select the Rose model file you wish to import and click OK.

Import processing begins and progress is indicated on the Together status bar.

Notes:

If the converter finds more than one diagram with the same name in one logical package, the diagram file names are modified by adding an incremented index to the end of the file name. For example, if there is already a diagram file named Main.vfUseCase and the converter needs to save another Main.vfUseCase file, then the latter one is saved as Main1.vfUseCase and the diagram names reflect the changed filename).

Note links ("Note anchors" in Rose terminology) are not imported.

Exporting a Together model to Rose

Direct conversion of a Together project into Rose's .mdl format is not supported. However, you can export model information to an XMI-compliant format that can be imported into Rose (see XMI Import-Export).

Database Import-Export

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

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

Generating DDL

With this feature you can generate Data Definition Language (DDL) for a data table based on a Class diagram, ER diagram, or EJB. You can opt to generate DDL files only, or to generate DDL and run in the same operation.

To run Generate DDL:

1. Open a project containing the diagram to use as the source for information export.
2. Open the Class diagram, ER diagram, or diagram modeling an EJB, from which you want to generate a database schema.
3. Choose Tools | Database Import/Export | Generate DDL Expert on the main menu.
4. Use the Generate DDL expert dialog to choose the supported DBMS, set access parameters, etc.

> **TIP:** If you are exporting from a Class diagram, all classes in the source diagram must have their *Persistent* property set to True.

Import database

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

To run Import Database:

1. Use your DBMS administration utility to create a JDBC/ODBC data source for the source database, if such a data source doesn't already exist. (You will need to specify the data source during the import procedure.)
2. Open a Together project in which to create diagram(s) from imported schematic information (or create a new project).
3. Navigate to the desired package.
4. Choose Tools | Database Import/Export | Import Database Expert on the main menu.
5. Use the Import Database expert dialog to choose the supported DBMS, choose tables, etc.

Generating and using IDL

If a Together project contains class diagrams with models appropriately structured to support IDL, you can generate and then round-trip engineer IDL for specific diagrams, selected classes, a specific package structure, or for the entire Together project.

Activating and deactivating IDL support

In products with IDL support, the support is deactivated by default for performance reasons. You can activate and deactivate IDL support as needed.

To activate IDL support:

1. Open a project.
2. On the Modules tab of the Explorer, expand the Early Access node to show the *IDL Export Support* node.
3. Expand the *IDL Export Support node* to show the *IDL Export Support* module.
4. On the speedmenu, choose *Activate*.
5. After activation, the *Generate IDL* command is added to the Tools menu.

Generating

To generate IDL:

1. Open the project containing the class diagrams that will be the source for IDL generation. If you want IDL for the whole project, skip to step 5.
2. If you want to generate IDL for a specific package, navigate to it and open the <default> diagram in the Diagram pane.
3. If you want IDL for a particular package structure, navigate into the top-level package of that structure and select it in the Model tab of the Explorer.
4. If you want to generate IDL for one specific package, select the package in the Model tab of the Explorer.
5. Choose Tools | Generate IDL on the main menu to launch the Generate IDL dialog.
6. In the dialog, choose the desired *Package* option based on your preferences in 1-4.
7. Select a target directory for the generated IDL file(s).
8. Optionally review and change IDL Options. Click Options and use on-screen help text to change any option settings. Click OK to effect any changes.
9. Click OK in the Generate IDL dialog to launch IDL generation.

TIPS:

- Open the message pane before generating IDL so you can see the messages generated by the process.
- Deactivate IDL Export Support in the Modules tab when you are finished to free up system resources.

Round-trip IDL support

Together supports round-trip IDL engineering. You don't need to import IDL into a project. You can simply create a project around existing IDL code and continue working with it in Together. The process is the same as creating any new project, except that you choose IDL as the default programming language during the creation process.

XMI Import-Export

You can import a model described in XMI into a Together project. This generates source code in the programming language(s) supported by your Together product.

You can export model information from a Together project to an XML file containing the modeling information described in XMI.

Note that XMI import-Export requires Sun JRE version 1.2x or above

XMI Import

XMI import is available on the main Tools menu. If you are importing a large model, close other applications to free up memory.

To import a model described in XML:

1. Create a Together project for the model, or open an existing project to which you want to add the XML model and create or navigate to the desired package.
2. On the Main menu, choose Tools | Import | Import from XMI.
3. Navigate to the XML file containing the XMI information you wish to import into Together.
4. Wait while the XMI code is processed. This may take some time depending on the size of the model.

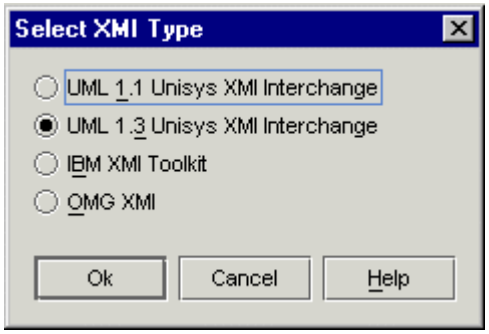
When processing is finished, a package diagram is created in the diagram pane. The Explorer's Model tab displays the packages and the default class diagram for each package. When you open diagrams and select classes, the generated source code displays in the Editor.

> **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.
>
> **Note:** Together supports only XMI 1.0.

Information about import operations is written into a log file located in the source folder of the imported xml file.

XMI Export

You can export a Together model to an XML file with the model described in XMI. You can subsequently import the XMI model into other systems that support XMI. Together supports UML 1.1 and UML 1.3 Unisys XMI interchange for 8 types of 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 Tools | Export | Export to XMI.
3. Select the desired XMI type from the dialog window and press OK.
4. In the Select Directory dialog, choose destination directory to store the resulting XML file. If such directory doesn't exist, right click on the desired location and choose Create New Folder on the speedmenu.
5. Wait while the XMI code is generated. This may take some time depending on the size of the model. Progress displays on the Together status bar.

User configurable translations

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

```
Tag_Name = RationalRose$MDC/:Tag Name
```

If you wish to import or export some specific properties of your project from/to XMI, you can edit the file `%TGHOME%/modules/com/togethersoft/modules/interchange/InterchangeTags.config`.

DTD Import-Export

You can import a Document Type Definition (DTD) file into a Together project. The import process creates a new XML structure diagram in the current package.

You can modify the DTD elements visually in the XML structure diagram. You can also load the DTD file in the Editor and edit it as text. However, if you modify the XML structure diagram visually, the original DTD file is not updated. To update the file you should export the XML structure diagram.

To import a DTD:

1. On the Main menu choose: Tools | Import | Import from DTD
2. In the resulting file chooser dialog, navigate to and select the DTD file you want to import.

A new XML structure diagram file is created in the current package. The diagram opens in a new tab in the Diagram pane. A shortcut to the new diagram is added to the diagram that was current before you began the import process.

To export a DTD file from a modified XML structure diagram:

1. Make sure the XML structure diagram is open, and is the current diagram in the Diagram pane.
2. On the Main menu choose: Tools | Export | Export to DTD.
3. In the resulting file chooser dialog, specify the location and filename of the DTD file you want to export.

To open a DTD file in the Editor:

1. Right-click on the Editor pane.
2. On the speedmenu choose *Open*.
3. In the resulting file chooser dialog, navigate to and select the DTD file you want to open.

> **Important:** *Together does not presently provide round-trip engineering for XML. Thus, modifications to an XML structure diagram imported from a DTD are not reflected in the source file. Likewise, changes to the source file are not reflected in any XML structure diagram.*

Exporting model information

Together provides two system modules for exporting model information:

Generate DDL: Generates Data Definition Language for a data table based on a class diagram.

Generate IDL: Generates Interface Definition Language for the current project.

Both modules are displayed in the System folder on the Modules tab of the Explorer.

To run either module:

1. Select the Modules tab.
2. Navigate to the desired module.
3. Choose Run from the speedmenu.

Generating DDL

The system module *Generate DDL from Class Diagram* extracts model information from a class diagram and creates a DDL file that a compliant DBMS system can read to generate a table schema based on the classes and their attributes and relationships. You can run this same module using the Generate DDL command on the main Tools menu.

To generate DDL:

1. Open a project containing one or more Class diagrams that will serve as the information source for the module.
2. Open the diagram(s) for editing in the Diagram pane.
3. If you open multiple diagrams, click the tab of the first one for which you want to generate DDL to make it the current diagram.
4. Run the Generate DDL module from either the Modules tab or the Tools menu (Database Import/Export | Generate DDL Expert). The Generate DDL Expert shows up.
5. Choose the target database from the list of supported DBMS systems. Generated DDL output will be compatible with the system you choose.
6. Optionally choose a different output location from the default location indicated.
7. Click OK to generate the DDL file.
8. Repeat 3-7 for other class diagrams.

Important: All classes in the source Class diagram must have their *Persistent* property set to True.

Generating IDL

The system module *Generate IDL* extracts model information from a class diagram and creates a IDL file.

To generate IDL:

1. Open a project you will use to generate IDL.
2. Make sure that IDL Export Support module is activated (Options | Activatable modules).
3. Select Generate IDL item in Tools main menu item.
4. Select packages you want to use for IDL generation.
5. Optionally choose a different output location from the default location indicated.
6. You can use Options button to setup IDL generation options.
7. Click OK to generate IDL file.
8. Repeat 3-6 for other class packages.

Version Control

Multi-user Team Support

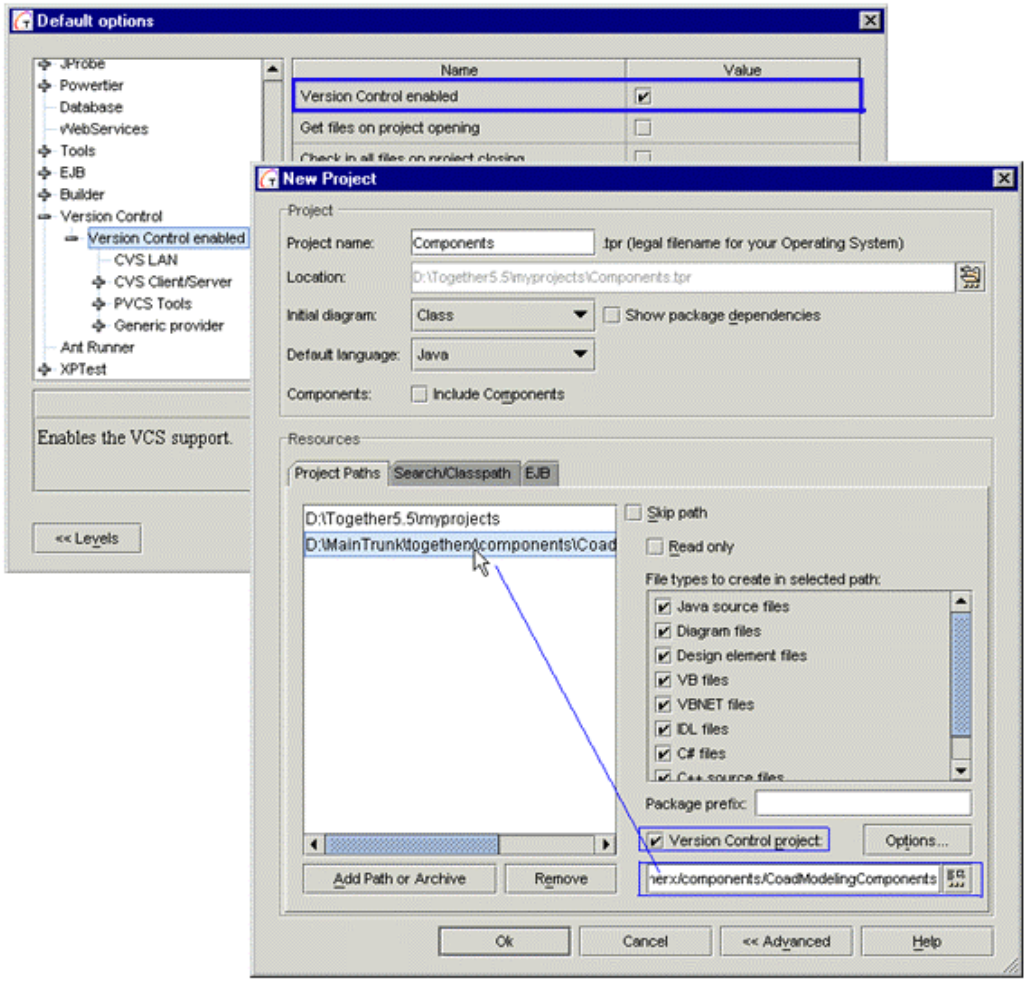
Together delivers support for true multi-user development.* You can use these features to...

- Protect your company's software assets
- Help team members work together
- Unobtrusively impose team-wide or enterprise-wide standards

Multi-user Version Control System

Together delivers seamless integration with your version control system 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 spending your time in the care and feeding of a proprietary internal repository like those found in some other design and development products.

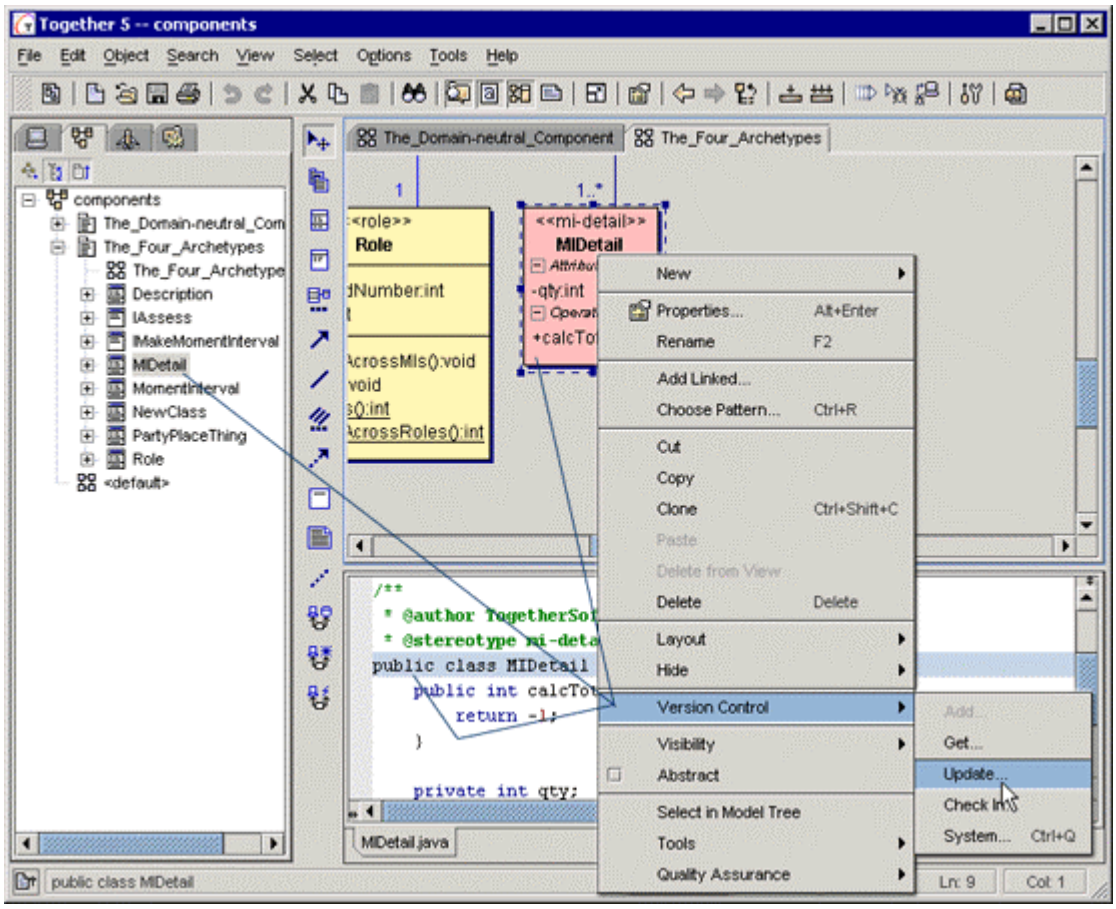
Together provides built-in integrations for leading multi-user version control systems, including those that support the SCC standard for Windows systems. **CVS**, a leading multi-user version control system, is bundled with Together pre-configured for immediate use.



Set Version Control options, enable version control for the project, and you can interact with version control right from the speedmenu system.

Interacting with the configured VCS

Once you have configured your version control options and associated your project with a version control project, it's a snap to interact with you VCS from Together. Version control commands appear 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.

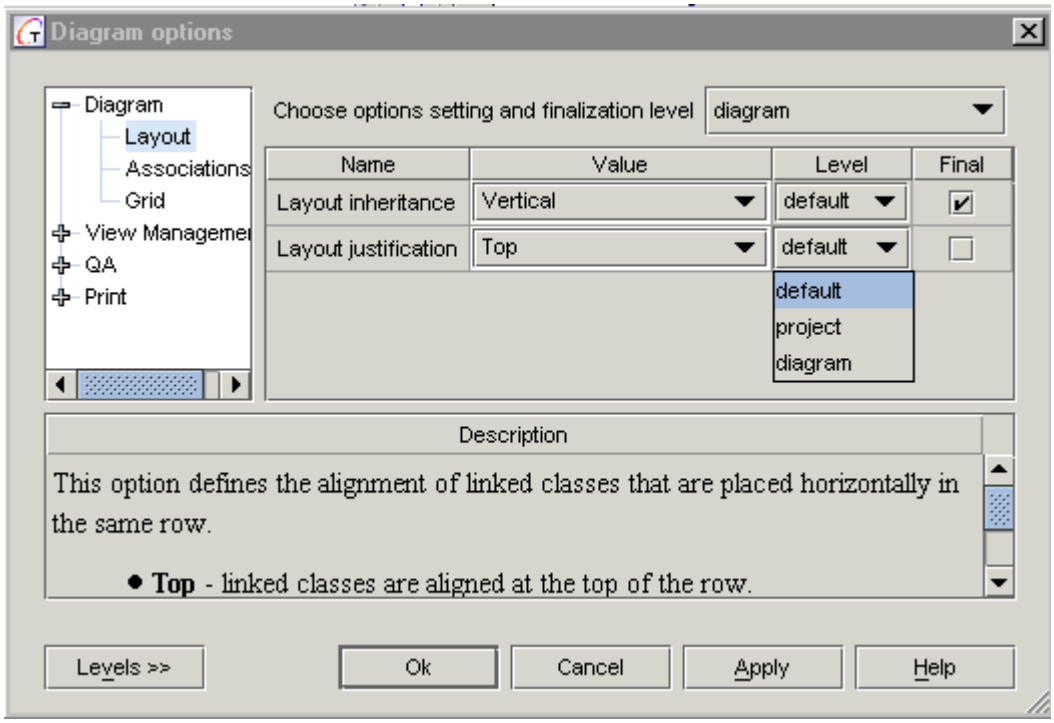


For more information on using Together with a Version control system, see Using Together with a Version Control System.

Hierarchical Configuration Options

Using the Advanced mode* of the Default Options dialog you can set configuration options at three levels- globally for your installation, project-specific, and diagram-local. The settings of all levels are merged together. If an option is set in multiple levels, the more local setting overrides the more global one. For example, diagram level overrides project level, which in turn overrides default level. However, options marked *Final* at a global level (by a project administrator on a shared installation, for example) cannot be overridden at a more local level. This is how you can enforce code formatting or blueprint options installation-wide on a server-based installation.

The number of levels is extensible by modifying the underlying properties file.



Set options at any of three default levels. Mark as Final any that you don't want overridden at another level.

*(Only available in products supporting this feature. For more information visit *www.togethersoft.com/together/* or contact TogetherSoft sales.)

See also

Advanced customization
Common customizations
Configuring Together

Using Together with a Version Control System

Together delivers seamless integration with leading version control systems. From the Together environment, you can get, add, check in, and check out source code using handy right-click menu commands on visual source elements in diagrams and the *Model* tab of the Explorer.

Overview of version control support

Together supports two main version control platforms: CVS and SCC. The SCC standard is supported by many leading version control products, one of which you should already have installed and working before configuring Together to work with it. CVS 1.11 comes bundled with the Together installation. Both CVS-LAN and CVS Client-Server are included. Together comes pre-configured for CVS LAN (local mode). Compiled CVS binaries for the different supported OS platforms can be found in the following locations in the Together installation:

Windows: %TGHOME%\bin\win32\cvs.exe
Linux: $TGHOME/bin/linux/i686-unknown/cvs
SunOs: $TGHOME/bin/sunos/sun4u-sparc/cvs
HP-UX: $TGHOME/bin/hp-ux/cvs
Compaq Tru64: $TGHOME/bin/osf1/alpha-alpha/cvs

Getting started with the Version Control

To work with your version control system (VCS) from Together, you need to do several things:

1. Enable Together's version control integration support in your configuration.
2. Set Version Control configuration options for the VCS you will use with Together.
3. Enable version control for your Together projects, specifying the VCS project or repository to use for each.
4. Use the various right-click menu commands and dialogs for specific CVS operations (add, get, etc.), 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 page of the Options dialog provides configuration settings that enable Together to work with your version control system. You should already have your VCS installed and operational before configuring Together.

Remember that Together's configuration system is multi-level. In the case of version control, this means you can set up versioning options to apply globally (Default level), or to a specific project (Project level). Most users generally set version control at the Default level, so the step-by-step procedures here will cite that level. For more information on multiple configuration levels, see Configuring Together.

Enabling version control support

Version control support is enabled in Together by default.

To enable or disable Together's version control support:

1. On the main menu, choose Options | Default to launch the Options dialog.
2. Select the Version control page of the dialog.
3. Use Advanced to select the configuration level if you want settings to apply at the Project level (project must be open).
4. Check the *Version Control Enabled* checkbox to enable version control support in Together. (Clear it to disable versioning support if you ever need to do that.)
5. Click *Apply* to immediately effect the change, or wait until you set the options for your version control system as described in the next sections.

At this point, you have made sure that support for version control integration is "turned on". The default configuration is in effect, which means Together uses CVS LAN in *Local/LAN* mode with default settings. If you are just getting started using version control for the first time and only need local access, these settings may be OK. But chances are that you will need to change VCS-specific settings so that Together can work with some existing version control system.

Choosing which VCS to use

If the CVS-LAN setting is what you want, you can skip this step and go on to Setting Up CVS LAN. Otherwise, you need to choose which system you want to work with. You should still be in the Version Control node of the *Options* dialog.

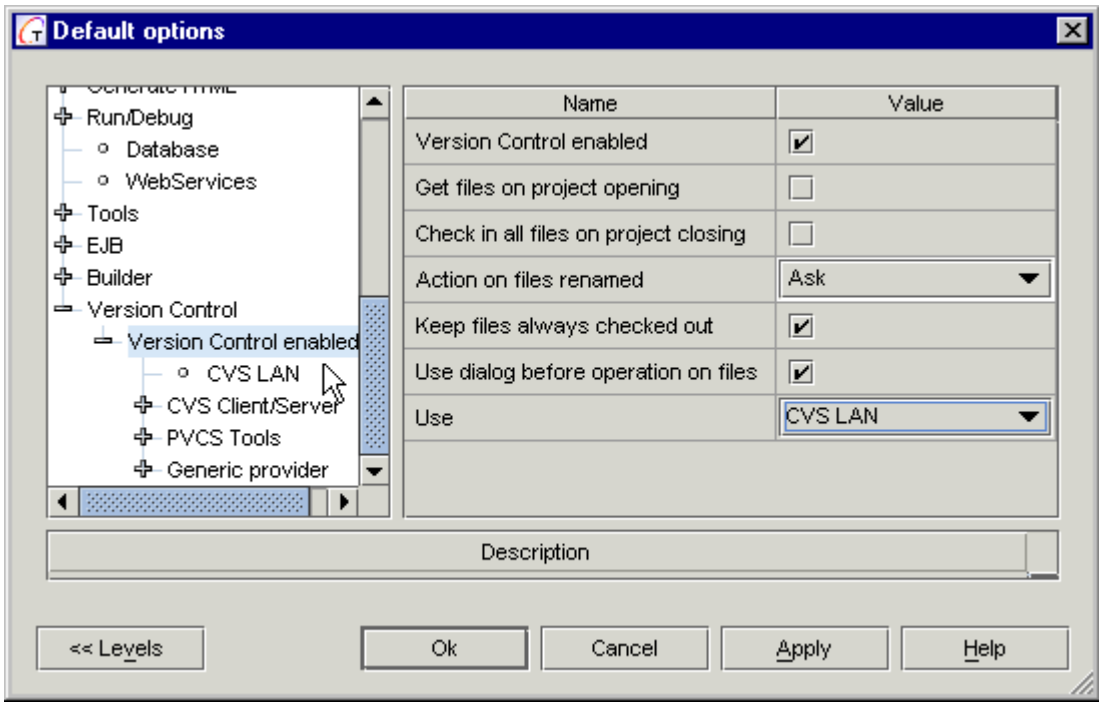
To choose a version control system:

1. Expand the Version Control enabled node to expose the general and system-specific configuration options.
2. Locate the Use node and choose the desired VCS from the drop-down list. The choices include:

CVS LAN (the default, accesses local or LAN-based CVS repository or server)
CVS Client/Server (for accessing a remote CVS server)
SCC: (for accessing an SCC-compliant version control product)
PVCS Tools (for configuring PVCS tool commands)
Generic Provider (for custom VCS interaction)

Configuring system-specific version control options

After choosing the VCS, you need to set system-specific options to that Together can work with the selected versioning system. The Version Control page of the Options dialog presents several option nodes corresponding to the different choices in the *Use* option. You need to set options only for the system you are using and ignore the others. Thus, if you chose CVS Client/Server in the Use option, you need to set the options under that node, ignoring CVS LAN, etc.



After selecting the VCS you want to use, go to the node that contains further configuration options

Setting up CVS Client-Server

1. In the *Version Control* node of the *Options* dialog, expand the node for CVS Client/Server.
2. If you connect to the repository via dial-up or similar non-persistent connection, check *Go offline by default*. Otherwise, you can leave this option unchecked.
3. In the Host field specify the host name or a valid alias for your CVS server. This is the same host you use in your CVS login command. For example, if your login command begins with: *cvs -d :pserver:/cuthbert@ourCVS-host* you would enter *ourCVS-host* in this field.
4. In the Repository filed, 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 you would enter /repository_alias in this field. Important: be sure to include the initial forward slash character.
5. In the Connection method drop-down list, choose the method by which you connect to CVS: pserver, or server.
6. If you chose pserver in #5, expand the pserver option and specify the correct port for your CVS server. Consult your CVS administrator for the correct port number if you are not sure.

7. If you chose server in #5 above, expand the server option and specify the correct port and command for your CVS server. Consult your CVS administrator for the correct port number if you are not sure.

8. 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 (1 minimum, 9 maximum) in the Compression level field.

9. If necessary, specify an appropriate vendor tag in the Vendor tag field. If not using such tag, clear this field. If you're not sure, consult your project manager or CVS administrator.

10. If necessary, specify the Release (Rev) tag for your CVS repository. If not using such tag, clear this field. If you're not sure, consult your project manager or CVS administrator.

11. Optionally specify the message you want to display when creating a new repository in the Default import message field.

If you are using CVS Client-Server and have completed the steps above, you can skip to *Enabling Version Control for Projects*.

Setting up CVS LAN

Configuring for CVS-LAN is similar to CVS-CS, especially if you're using *pserver* protocol... you specify host, repository, ports, and tags as described in the previous section, only you do so under the *CVS LAN* option node.

1. In the Version Control page of the Options dialog, expand the node for CVS LAN.

2. Specify the path the CVS LAN shared folder in the *Shared Folder field*. For a local or LAN-based repository, this is the local or LAN path to the repository. For a server-based repository (using pserver protocol), this is the path to the repository on the server.

3. Choose the connection mode in the *Mode* field's drop-down list. How you set the remaining options depends on your choice in this field.

4. Specify the name of the CVS executable file. The default is *TGH/bin/win32/cvs.exe* which is applicable to Windows, but appears as the default no matter what your OS. If you are not running under Windows, specify the path to the CVS executable for your OS (see Overview of Version Control Support earlier in this chapter.)

5. If you are using *pserver* protocol for a server-based repository, set *Server Name, Port, Vendor Tag,* and *Release Tag* as described in Setting up CVS Client-Server (see steps 3, 5, 9, and 10).

6. Optionally specify the message you want to display when creating a new repository in the Default import message field.

Setting up a 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 PVCS, StarTeam, Perforce, and Continuus version control systems. For specifics about these systems, see Product-specific VCS notes (or the readme files provided in *$TOGETHER_HOME$/modules/com/togethersoft/modules/vcs*). Other SCC versioning systems may be used with Together, but only those mentioned in these readme files are currently tested.

Use the *Generic provider* options to configure other version control products.

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.

Make sure you log on to your Windows NT or Windows 2000 computer with full Administrator rights before installing Together.

After Together installation, you can check to confirm that Coroutine classes have been installed. Look for them in the directory: $TOGETHER_HOME$/lib/coroutine/com/neva/. Also check to be sure your classpath includes this path.

If, when using SCC version control feature, you get an error message that Coroutine can't 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: `$TOGETHER_HOME$/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

If you want to set global SCC options, you can do so with or without opening a project. To set SCC options for a project, you must first open it.

1. Choose Options | Default to display the Default Options dialog.
2. Click *Advanced* to display levels.
3. As you work, if you will be setting some options to apply at different levels, click the desired level before setting the option.
4. Expand the node for the option *Version Control enabled* (this option should be checked at Default level).
5. Click on the *Use* option and select *SCC* from the drop-down list.

To set SCC options:

1. Make sure SCC is selected in the *Use* option.
2. If using one of the tested versioning systems, expand the option node for your product.
3. If using another SCC system, expand the Generic option node.
4. Set the options of the selected node to conform to your version control system. Context-sensitive on-screen Help texts are provided.

Switching among different SCC providers

If you have several SCC providers installed on your system, for example `SourceSafe` and `PVCS VM`, you probably have to change registry entries in order to be able to work with specific source control system.

All SCC providers are listed in values of 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 the other tools, such as `DevStudio`

Tuning SCC 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 you have VSS explorer installed. If you are using several SourceSafe databases you may 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 rather use the ones stored in this file. If you ever want to switch the default database used by the source safe SCC interface, you should modify values 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 have to type the path to your preferred database in each of these keys in order to work with it.

Examining differences

Having selected the type of version control to be used, you can set up the desired tool to view differences. Together provides a built-in difference viewer, which is used by default. For CVS LAN and CSV Client-Server, however, checking the option *Using External Diff tool by default* allows to invoke any other external tool, whose fully qualified path is specified in the field *Name of External Diff executable*.

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.

New projects

Whether or not you immediately enable version control for a new project depends on your development plan for the project. If you think you will spend quite some time brainstorming, modeling, and designing, and you don't 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 thought out, or if you are creating a new project for an existing code base, you'll want to enable the new project for version control as you create it.

To enable version control in a new project:

1. Make sure your version control system is already installed and operations, and that it is configured in the Options dialog as described earlier.
2. Launch the New Project dialog (File | New Project).
3. Select Advanced mode.
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. Depending on which VCS you are using, you may be presented with a login dialog when you click the browse button. One you log in, a chooser dialog relevant to the configured VCS project or repository is presented. Choose the folder or repository that is relevant for the project you are creating.

For more information on creating projects, see Creating and opening a project.

Existing projects

To enable version control in an existing project:

1. Make sure your version control system is already installed and operations, and that it is configured in the Options dialog as described earlier.
2. Launch the Project Properties dialog (File | Project Properties).
3. Use Advanced mode.
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. Depending on which VCS you are using, you may be presented with a login dialog when you click the browse button. One you log in, a chooser dialog relevant to the configured VCS project or repository is presented. Choose the folder or repository that is relevant for the project you are creating.

Other version control information

Together files to include in version control

To protect your visual modeling information you should 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 look for and add to source control:

File(s)	Description	Where located
`*.tpr`	*Together* project file	Project primary root folder
`*.df*`	UML diagram files. **.dfPackage** is the default package view diagram generated by Together. All other diagrams are user-created.	There will be one .dfPackage file in every folder in the project that has been parsed. Other diagram files are wherever users have created them.
`*.tws`	User-specific desktop settings. Probably not a source-control item, especially in multi-user environments, but you can back it up in VCS if you want to.	Project primary root folder

CVS documentation

You can find documentation on CVS in the file $TOGETHER_HOME$/bin/win32/cvs.html, or refer to the **http://www.gnu.org/manual/cvs/html_chapter/cvs_20.html** .

Peculiarity of jCVS in Windows environment

Together uses jCVS, a Java based CVS Client When working in Windows environment, jCVS can throw exception

```
java.lang.StringIndexOutOfBoundsException: String index out of range: -1.
```

This situation takes place if the CVS repository contains folders with the same names presented in varying case, for example:

```
MyRepository

MySourceFolder
<...>
mysourcefolder
<...>
```

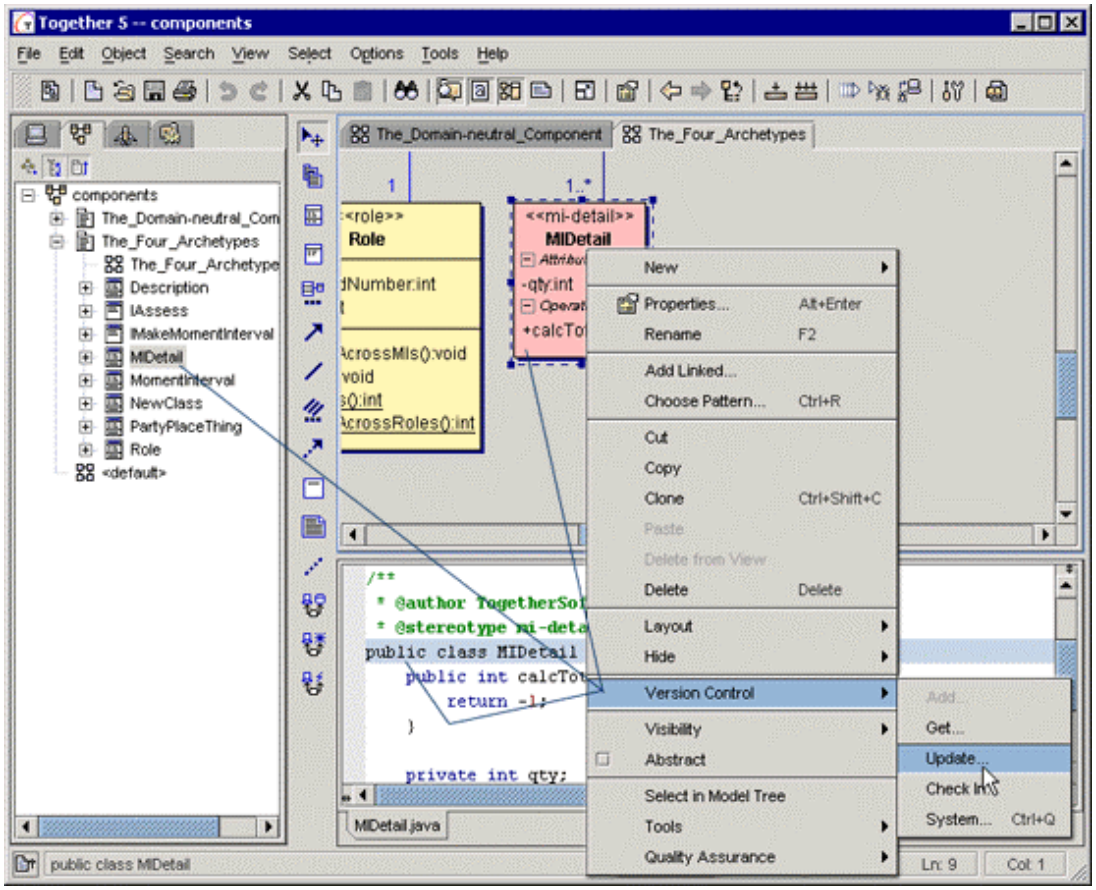
Make sure you don't confuse folder names to avoid this problem.

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, etc. To use version control in your project:

1. Make sure that version control is enabled in the Options and Project properties.
2. Select a source-generating diagram element in a diagram (Class or Interface for example), or...
3. Select a source generating element in the Explorer, or...
4. Focus the Editor with a class source file loaded.
5. Right-click and choose *Version Control* from the speedmenu. This displays a submenu of the commands available for the currently configured VCS.
6. 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're experienced with your VCS, the dialog should be fairly self-explanatory. For information about the various dialogs for different systems, see Version control dialogs.



Access the most needed VCS commands right where you work

See also

Product-specific VCS notes
Project basics
Project management

Product-specific VCS notes

Together® has been tuned and tested with PVCS (VM and Dimensions), StarTeam, Perforce, and Continuus version control systems. This topic covers specifics related to these systems. Other versioning systems may be used with Together, but only those mentioned here are currently tested.

PVCS command line tools

This section covers support for PVCS command line tools in Together®. Support for PVCS tools was introduced in Together build 822 and is currently available only on the Win32 platform.

The latest version of the module implementing support with PVCS tools was tested with PVCS VM 6.5 running on Windows NT 4.0 SP 5.

Known problems

- PVCS locks are not recognized. Together figures out if the file is checked in or out ONLY in function of its read-only status. That is, if a writable file is recognized as being under version control, it is assumed to be checked out. On the other hand, a read-only file under version control is always assumed to be checked in. Recognition of locks will be added in further versions.

- At startup time, the path to the PVCS database is checked. If the directory does not exist, the module goes into an uninitialized state and you must restart Together in order to make it function. This is a known problem and will be fixed soon.

- Advanced options of the PVCS tools must be set using the Options dialog. For convenience, later versions will add the ability to specify additional options to most of the version control commands.

Setting up Together to use PVCS tools

1. In the Options dialog, go to the Version Control tab and select PVCS Tools from the drop-down list in the Use field.

2. Set the path to your PVCS database directory in PVCS Database (visible after expanding the PVCS Tools option). By default, the value points to: `c:\Program files\PVCS\VM\SampleDb,` which is also the default for PVCS.

The default archive suffix is set to "-arc", which may be changed, for example, if you are using "v" as default suffix.

Below, we provide specifications of the PVCS commands used by Together to operate with the repository. You may want to add some options or specify the full path to all commands if your PVCS tools are not on PATH. By default, it is assumed that all tools are on system PATH.

These commands are used by Together for version control tasks:

PUT - used for Add and CheckIn

GET - used for Get and CheckOut (with -l parameter)

VLOG - used to get History and Details

VDIFF - used to get the difference with the latest revision in repository

VDEL - used to remove a file from repository

VCS - used to test if PVCS tools are on PATH. It is executed at start-up time. If the command can not be executed, the module does not function.

Specifying a PVCS project in the Together Project Properties dialog

1. When in the Project Properties dialog, either with a new project or having already opened one, click on the Version Control Project checkbox to have the PVCS project be associated with the currently selected root. By default, the short name of the root directory is used as the proposed PVCS project.
2. Click *Select* to see the list of available projects in the repository. Note that Together internally adds the archive suffix to the path entered in the *Options* dialog, so you do not have to specify it explicitly.

Using PVCS within a Together project

After Together has opened a project that contains roots associated with Version Control projects, the *Version Control* command is added to each element's speedmenu. This leads to a submenu with VCS-specific commands.

Short summary of commands:

Add: adds file(s) to repository

Get: get latest version of file(s) from the repository

CheckIn: put local file(s) into the repository and optionally unlock

CheckOut: get latest version from the repository and lock

System: displays dialog with further information about files

Within the System dialog, additional operations are available:

Diff: shows the result of VDIFF command, including the local file and latest version in repository

History: shows the history of revisions, result of VLOG with -br parameter

Details: detailed information about the file: locks, revisions etc. The information is the output of command VLOG with -b parameter.

PVCS Dimensions

This section covers issues concerning support of PVCS Dimensions with SCC interface.

The SCC integration with PVCS Dimensions version 5.0 was tested with Together® 4.2 under SUN JDK 1.2.2, Windows NT 4.0 SP 5 and Windows 2000.

Known issues

- Before starting to use Dimensions with Together, Dimensions must be configured to support Together as IDE. See the corresponding section in this document for details.

- The directory path set in the default Work Set must be the parent directory of the Together project root that is 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

- The SCC integration does not work under JDK 1.3. There is no workaround for this problem at time of writing. You must use SUN JDK 1.2.2.

- When running Version Control commands from Together, Dimensions may display a modal window with command execution status, which is inactive and can't 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, no matter what was the state of Keep Checked Out checkbox, Dimensions deletes the file(s) from the local directory. To get these files back, execute "Get" or "Checkout" on the directory where the file(s) 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.

In product $GENERIC, 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 lifecycle SOURCE to the object type.

Define file formats and MIME types for file types you wish Together to upload to Dimensions, if not yet defined. These types are:

Name	Format	MIME type
JAVA	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/gif

In IDE Setup tool, define item types used by Together. These items are:

Pattern	Format	Type
%.java	JAVA	SRC
%.cpp	C++	SRC
%.cc	C++	SRC
%.hpp	C++	SRC
%.h	C++	SRC
%.idl	IDL	SRC
%.pas	PASCAL	SRC
%.dfActivity	DIAGRAM	SRC
%.dfBusinessProcess	DIAGRAM	SRC
%.dfClass	DIAGRAM	SRC
%.dfComponent	DIAGRAM	SRC
%.dfDeployment	DIAGRAM	SRC
%.dfEJBAssembly	DIAGRAM	SRC
%.dfER	DIAGRAM	SRC
%.dfPackage	DIAGRAM	SRC
%.dfSequence	DIAGRAM	SRC
%.dfUseCase	DIAGRAM	SRC
%.dfXMLType	DIAGRAM	SRC
%.tpr	PROJECT	SRC
%.tws	PROJECT	SRC
%.wmf	WMF	DAT
%.gif	GIF	DAT

View Management

Together's view management features take into account the fact that not all of the stakeholders in a project need to see everything contained in a model all the time. Domain experts don't need to see implementation details, for example. With Together, you see what you want to see, when you want to see it.

View management features

These features are:

Creating work views

You can toggle Main window panes to create the desired view.

Control the general level of detail shown in diagrams

Choose the desired level of details in diagrams. Possible levels are: Default, Analysis, Design, Implementation. Default level is assigned according to the current role.

Control whether members in Classes display with UML or Java format

This options controls the way the elements are displayed in diagrams. Java format is only available for the products with Java language support.

Show or hide subpackage contents in diagrams

Check this option to display subpackages' contents as package icons with the lists of classes, interfaces and underlying subpackages.

Control how Java Bean classes/C++ properties display in Class diagrams

If the option *Recognize JavaBeans* is on, the classes on the Class diagram are recognized as JavaBeans. Bean tab adds to the Object Inspector of the classes, where you can add bean properties, getters and setters, event sets. *Show attributes and accessors* option controls whether bean properties and events show up on the class icon.

You can also opt to impose constraints on the classes with or without public/default constructors.

The option *Recognize C++ properties* allows to display C++ properties in classes.

Recognize JavaBeans/C++ properties options are also available from the Options menu and correspond to the Project level of the Options dialog.

Show or hide referenced classes in diagrams

When checked, displays names of the referenced classes in diagrams.

Control the display of dependencies

You can opt to show dependencies between classes and interfaces. If this option is on, it is possible to choose the scope of analysis: declarations only, or all usages of the dependencies, including the method bodies. Besides that, it is possible to recognize @see tags as hyperlinks.

Caveat

Recognizing dependencies option can result in slow performance and too large diagrams.

Control display of the Sequence diagrams

The options under this node provide full control of Sequence diagrams display. You can opt to show object class names and message numbers, control display of messages in Sequence diagrams, assign depth of call nesting, choose generating and showing multiple diagrams.

Control banned destinations

Use this group of options to keep association links under control. If the links to standard classes are not filtered from the view, your diagrams will soon become crowded with links that only impede comprehension. Some standard java classes are banned by default. Three additional slots allow to define custom banned destinations.

It is also possible to add more fields, but this requires editing the `viewManagement.config` file. This procedure is described step by step in the Description field of the Options dialog.

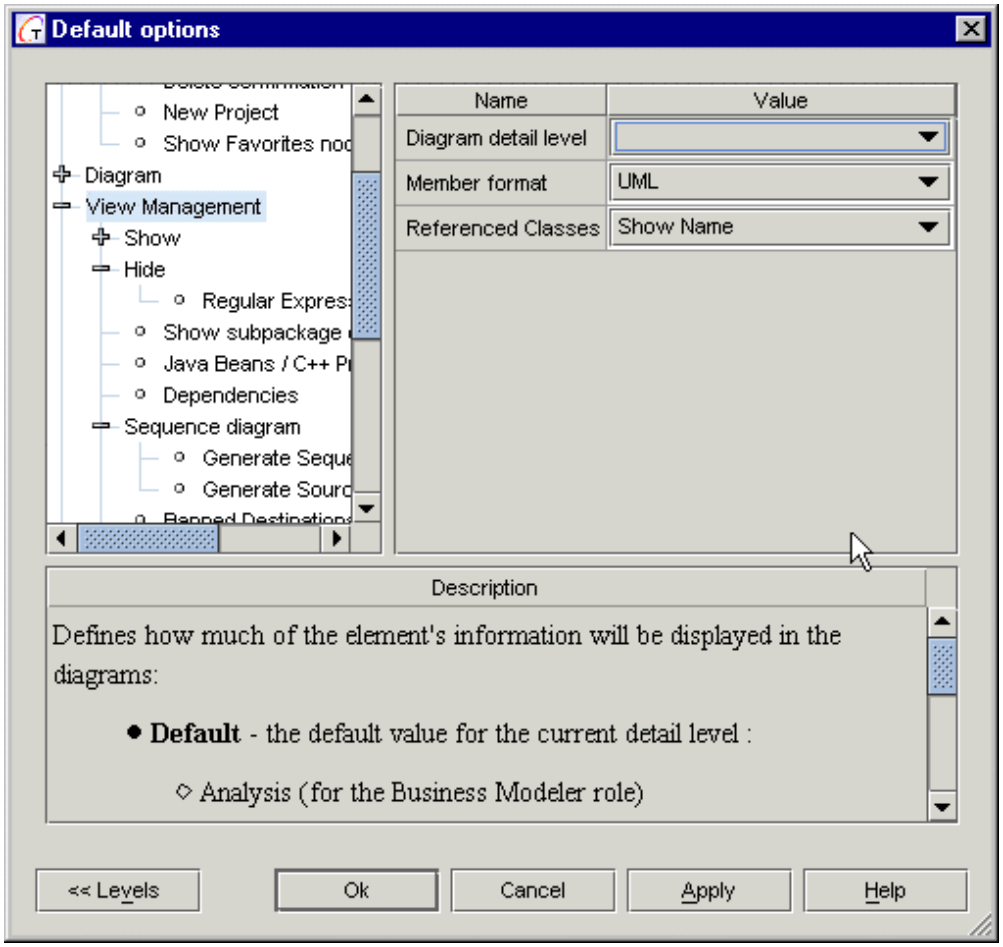
Sequence diagram options

In this node you can control display of the Sequence diagram: specify maximum call stack depth for Sequence diagrams generated from operations, choose generating multiple diagrams, control display of messages etc.

Show or hide aggregations of diagram elements and EJB elements

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

You can find detailed description of each option on the View Management page of the Options dialog.



View Management page of the Options dialog

See also

Working with View Management

Working with View Management

This topic discusses the different ways to show or hide model content to create different views of the project for different purposes or users- analysts, domain experts, managers, etc. Some technical books on UML refer to this practice as *elision* or *eliding* of model content. Together's **view management** features enables you to apply this concept at several levels.

View management mechanisms

Together's view management features provide several ways to control what you see in your visual model at any given time:

View Management "Show" Options: these configuration options enable you to:

- control how much detail is shown in classes,
- include or exclude different types or groups of elements from view

Element Show/Hide: optionally hide and restore individual elements or selected groups
Detail levels: control how much detail is shown in classes
Work views: toggle main window panes to facilitate your personal work style

"Show" Options

Together provides a number of configuration options that control which elements in a Class diagram are visible and which elements are hidden. For example, you might choose to hide such things as abstract Classes, private Members, or some types of relationship links. These are called simply *Show Options*... you use them to specify what you want to *show* in the diagram.

The Show options are expression-based and user-modifiable. Show Options are in one node of the View Management page of the Default Options, Project Options, and Diagram Options dialogs. Use Default Options to apply the settings globally at the level of your configuration. Use Project Options to apply the settings to a currently open project. Use Diagram Options to set them individually for a specific diagram, overriding the more global settings. **Tip:** The Diagram View Management button on the Main Toolbar provides a shortcut to view management options for the current diagram.

Setting Show Options

Show Options are binary in the sense that the elements they affect are either Shown (box checked) or Not shown (unchecked). The default state is Shown.

To set default Show Options:

1. On the Main menu choose Options | Default
2. Select the View Management page and expand the Show node.
3. Set options as desired and click OK to finish or Apply to set changes and continue setting other options.

To set Show Options for a diagram:

1. Make sure that the diagram you want to affect is the current diagram in the Diagram pane.
2. On the Main Toolbar click the Diagram View Management button to display the Diagram Options dialog. Select the View Management page.
3. Set options as desired and click OK to finish or Apply to set changes and continue setting other options.

To set Show Options for a project:

1. Open the project
2. On the Main Menu, choose Options | Project
3. Click the View Management tab.
4. Expand the Show node and set options as desired.

To turn Show Options on or off:

1. Clear checkboxes to hide the designated element(s).
2. Check the checkboxes to show the designated element(s).

Show/Hide for individual diagram elements

You can hide Node elements in all diagrams, either singly or as a group, using the *Hide* command on the speedmenus of such elements. Any relationship links between elements you hide this way become hidden as well. It is also possible to hide links and members. You can restore hidden elements and their links to view at any time.

To hide elements:

1. Select one or more elements.
2. Right-click on any of the selected elements.
3. Choose *Hide* from the speedmenu.

To restore hidden elements:

1. Right-click on the diagram background to display the Diagram speedmenu.
2. Choose *Show Hidden* to display the Show Hidden dialog.
3. The "Hidden" list on the left displays all hidden elements.

   To restore all elements click Show All

   To restore some of the elements, select one or more elements (the "Hidden" list is multi-select) and click Show.
4. Click *OK* to close the dialog.

The previously hidden elements are restored to view along with any links between them to elements that were not hidden to begin with.

> **Tip:** In large diagrams you may find it more convenient to use the Show Hidden dialog to *hide* elements. Just follow the procedure to restore elements, except that you select elements from the "Shown" list and use the Hide or Hide All buttons.

Detail levels for Classes

You can control how much of the information in Class diagram elements (e.g. Classes, Interfaces) displays in the diagrams. The different detail levels are:

Analysis: names only (no visibility signs)
Design: names and types (visibility signs are shown)
Implementation: names and types, parameters for operations, initial values of attributes (visibility signs are shown). This is the default detail level.

Set the Diagram Detail Level option on the View Management page of the Default, Project, and Diagram Options dialogs (Main menu |Options).

To change the detail level in a Class diagram:

1. Right-click on the background of the Class diagram.
2. Select Diagram Options... to display the Diagram Options... dialog.
3. On the View Management tab, in the Diagram Detail Level node, select the desired level of detail for the diagram and click OK to write the changes to your system configuration.

Managing Working Views with Panes

Together main window is comprised of several panes. Depending on what you do, you may or may not need to see some of the panes. For example, a designer might not need to see the Editor pane. And regardless of which pane you use most, it can be handy to get the Explorer pane out of the way. You can do pane management in 3 ways: with the main View menu, the Main Toolbar, or using keyboard shortcuts. Have a look at the View menu and try out the different toggles to see which view and toggle method you prefer.

Role-Based Workspace

Together introduces the concept of **role-based workspace**. The role-based workspace is a framework designed to enhance team communication, by providing each team member with an individual set of facilities and enabling team members with different roles to exercise their own approach to development.

Role is a pre-defined configuration of the user interface for specific groups of Together customers. Choose your role - and Together automatically optimizes the user interface for working in that role, by adjusting tool-bar content, menu content and workspace layout accordingly.

Roles are only available in ControlCenter.

The four roles are:

Business Modeler:

If you are a domain expert or analyst, choose this role. In brief: diagram editor central, text editor upon demand, tool bars and menus focused on business modeling.

Explorer displays only the items that pertain to business modeling: The Modules tab of the Explorer is hidden, and the Directories tab shows only the Current Project, Samples, and Users projects nodes. New Project and Project Properties dialogs are removed from the File menu (Project Expert can be used instead), Activatable modules are removed from the Options menu, Quality Assurance and Compile are removed from the Tools menu, and EJB buttons do not appear on the Class diagram toolbar. Speedmenus are also streamlined for the business modeler's tasks: source formatting, make/rebuild QA features, and the Choose Pattern command for members and attributes are all removed.

Designer:

If you are an analyst and designer - or a designer - choose this role. In brief: both diagram and text editor central, everything up to the point of compilation, yet not further.

Developer:

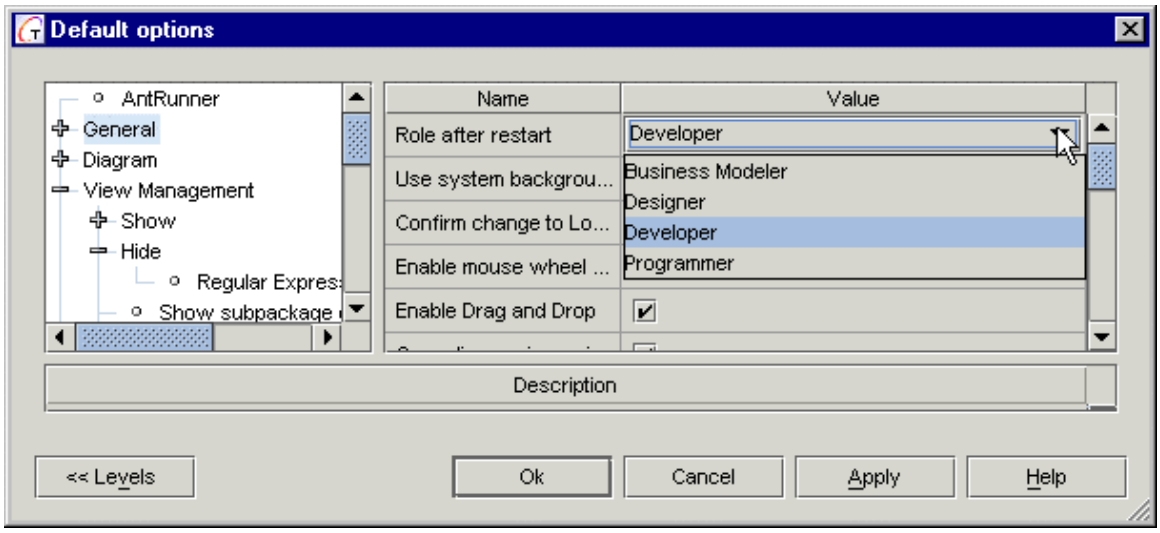
If you do a little bit of everything, if you are an analyst, designer and programmer, - or a designer and programmer, choose this role. In brief: both diagram editor and text editor central; compile, debug, assemble, deploy and run.

Programmer:

If you are a programmer, choose this role. In brief: text editor central; compile, debug, assemble, deploy and run. With this role selected, Together opens with the closed Diagram pane, but it is still possible to open it if required.

Changing the configured role

You select a role during installation. However, it is possible to change roles any time in the General options of the Options dialog (Options | Default). Select the new role from the drop-down list in the *Role after restart* option. The change takes effect next time you start Together. Advanced users can also modify the default role-based configurations by editing the `workspace.config` file.



Complete descriptions of the default settings and detail levels can be found in the Description field for the General node in the Options dialog (Options | Default), and in Diagram Options | View Management of the Diagram speedmenu.

Part 2. Working with Features

Modeling with Together

Introduction to modeling

The main reason for modeling is to organize and visualize the structure and components of software intensive systems. With models, you capture requirements, identify and specify subsystems, and visualize and document logical and physical elements, and structural and behavioral patterns.

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

- Identify different architectural viewpoints, i.e., use case, design, process, implementation, and deployment views
- Identify the system context and the actors involved
- Decompose a large, complex system into its most granular subsystems.

In addition, the authors outline a number of activities that apply to both the overall system and each of the subsystems:

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

Design view: model a design view specifying classes, interfaces, and collaborations. These provide a working vocabulary for the system in terms of both problem and solution. Create Class diagrams (including Objects as necessary) to model static aspects, and again, appropriate combinations of Activity, Collaboration, Sequence, and State diagrams to show dynamic aspects.

Process view: model a process view to describe threads and processes of various synchronization and concurrency mechanisms. Same diagrams as above are recommended, except focusing on active classes and objects representing threads and processes.

Implementation view: model the components used to build and release the system. Use Component diagrams for the static aspects, and again, appropriate combinations of Activity, Collaboration, Sequence, and State diagrams to show dynamic aspects.

Deployment view: model the nodes, components, and interfaces forming the hardware topology for the runtime system. Use Deployment diagrams for static aspects, and appropriate combinations of Activity, Collaboration, Sequence, and State diagrams to show the dynamic aspects.

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

The authors go on to point out that creating of a system architecture isn't a single event; rather it is a process of successive refinement, in a manner that is "*use case-driven, architecture centric, and iterative and incremental.*"*

Choosing the right set of models is important. There are no hard and fast rules, but the wrong models give you an inaccurate view of the system and jeopardize the overall success of a project. A few points to remember about *good* models are:

- Good models are a simplification of reality from a specific point of view
- Good models can stand alone semantically
- Good models are loosely coupled to other models
- Good models in aggregate provide a complete blueprint for a system.

UML and Together Diagrams

Together provides support for the most frequently needed diagrams and notations defined by the UML. As the UML specification evolves, you can count on Together to keep pace with developments and supply new builds that deliver updated UML support.

UML Diagrams and support

Together now supports the major UML 1.3 diagrams:

- Class (includes Object diagrams and Packages)
- Use Case
- Sequence
- Collaboration
- Activity
- StateChart
- Component
- Deployment

Together Diagrams

Together provides several other custom diagrams in addition to the UML types:

- Robustness diagram to provide robustness analysis of the use cases
- Business Process diagrams for modeling business operations
- Entity Relationship diagrams for data modeling
- Diagrams for visual assembling of distributed applications: EJB Assembler diagram, Web Application diagram and Enterprise Application diagram.
- XML Structure diagrams for visually creating DTDs. Import existing DTDs to get a quick visual picture of the elements and their relationships
- TagLib diagram to create tag libraries.

Notation

UML diagrams are rendered on-screen and in print using UML-compliant notation. In Class diagrams, you have full control over code generation for the various notational elements through either global or diagram level configuration options.

Stereotypes

Together supports the use of stereotypes. You can use stereotypes to adhere strictly to UML-defined stereotypes, or you can customize them to suit your requirements. You can even add color to stereotypes... a new dimension in communication that exceeds what the UML presently specifies.

Additional resources

You can find detailed information on UML modeling techniques for Class and Package diagrams in Chapters 8-12 of the *UML User's Guide*, and detailed information about component modeling in color in *Java Modeling in Color with UML: Enterprise Components and Process*.

Working with Diagrams

Creating Diagrams in Projects

This topic explains how to create new diagrams, and clone or rename existing diagrams.

Diagrams exist within the context of a *project*. You must create or open a project before you can create any new diagrams. If you create a Together project around an existing code base, *Together* automatically creates Class diagrams showing the contents of each package when it parses your code. Before creating a project you may want to customize forward and reverse engineering and/or source code formatting (see Configuring Together*)*.

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

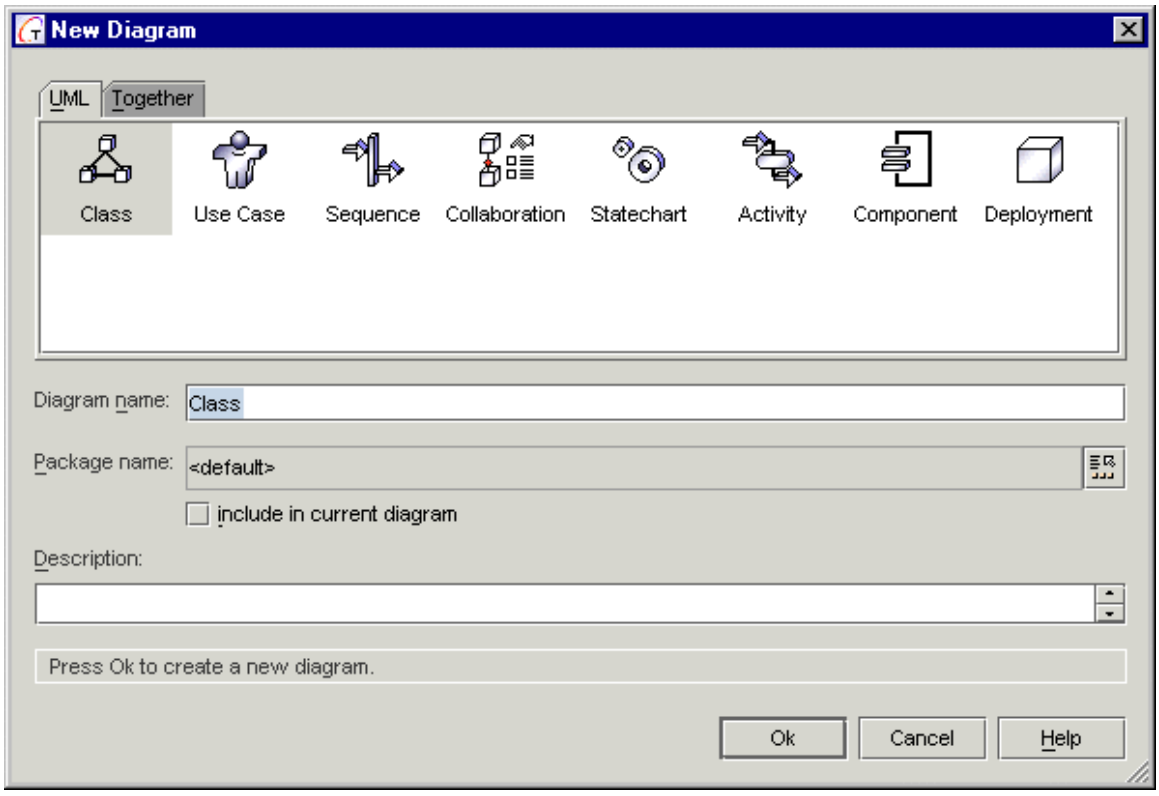
- UML diagrams (Class, Use Case, Sequence, etc.)
- Special Together diagrams (Entity Relationship, EJB Assembler, XML Structure, etc.)

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

- the Main menu
- speed menus in the Explorer
- the Hyperlinking feature

Using the Main Menu or toolbar

You can use the **File | New Diagram** menu command, or the New Diagram icon of the toolbar to create a new diagram in any directory/package in your project.



1. Select the destination directory or package in the Project Explorer's *Directory* or *Model* tab.
2. Choose File | New Diagram from the menu bar, or click on New Diagram icon of the toolbar to display the New Diagram dialog (see figure above).
3. Click the icon for the type of diagram you want to create (only those diagrams enabled by your license are enabled).
4. Enter the diagram name and optionally its description.
5. Choose a destination for the new diagram file. Default is the package currently selected in the Explorer.
6. Check *Include in current diagram* if you want to place an element with a logical link icon to the new diagram in the current diagram.

Using Explorer speedmenus

When a package node is selected in the Explorer's Model tab, you can use the New | Diagram command on its speedmenu to display the New Diagram dialog box. Follow the same steps as above.

Using the Hyperlinking feature

The Hyperlink feature enables you to create a new diagram that is automatically linked to an existing diagram or diagram element. You can do this in the Hyperlinks tab of the class Inspector (class speedmenu | Properties). For more information see Hyperlinking diagrams.

Cloning diagrams

The Clone command on the speedmenus of existing diagram nodes in the Explorer Model tab lets you quickly create a new diagram with the same content as the existing one. The new diagram is created with a unique default name in the same package as the first diagram. You can rename the clone diagram as described below.

Renaming diagrams

To rename an existing diagram:

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

Configuring diagram options

The *Diagram* page of the Options dialog provides a number of customization settings that affect diagrams in general and Class diagrams in particular. You can set these options either globally for all diagrams, project-wide for just the current project, or locally to the current diagram. For more information, see Creating a shared multi-user configuration.

See also

Opening diagrams for editing
Creating and opening a project
Configuring New Diagram dialog

Drawing diagram elements

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

When you create a new diagram, the Diagram pane presents an empty background. You place the various Node elements (e.g., Class, UseCase, etc.) on the background and draw relationship *links* between them (Association, Communicates, etc.) What elements and relationships you draw is up to you... whatever meets the requirements of your model.

The main tools you use for constructing diagrams are:

Diagram Toolbars: Place icons for Node elements and draw links on the diagram background.

Diagram speedmenu: Show hidden objects, manage the layout, control Zoom, configure diagram-level Options, update diagrams and hyperlinks.

Element speedmenus: The right-click menus of the various nodes and links provide functions specific to each. For example, you can add or delete members (or delete the element itself), cut-copy-paste, hide and show elements, route links, and more. Explore the speedmenus of the different elements as you encounter them to see what's available for each one.

Inspectors: (Speedmenus | Properties) Edit diagram or element properties, create hyperlinks between diagrams, elements of diagrams and other diagrams, and between diagrams or elements and files or URLs. Edit annotations and source code comments.

Using the Grid

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

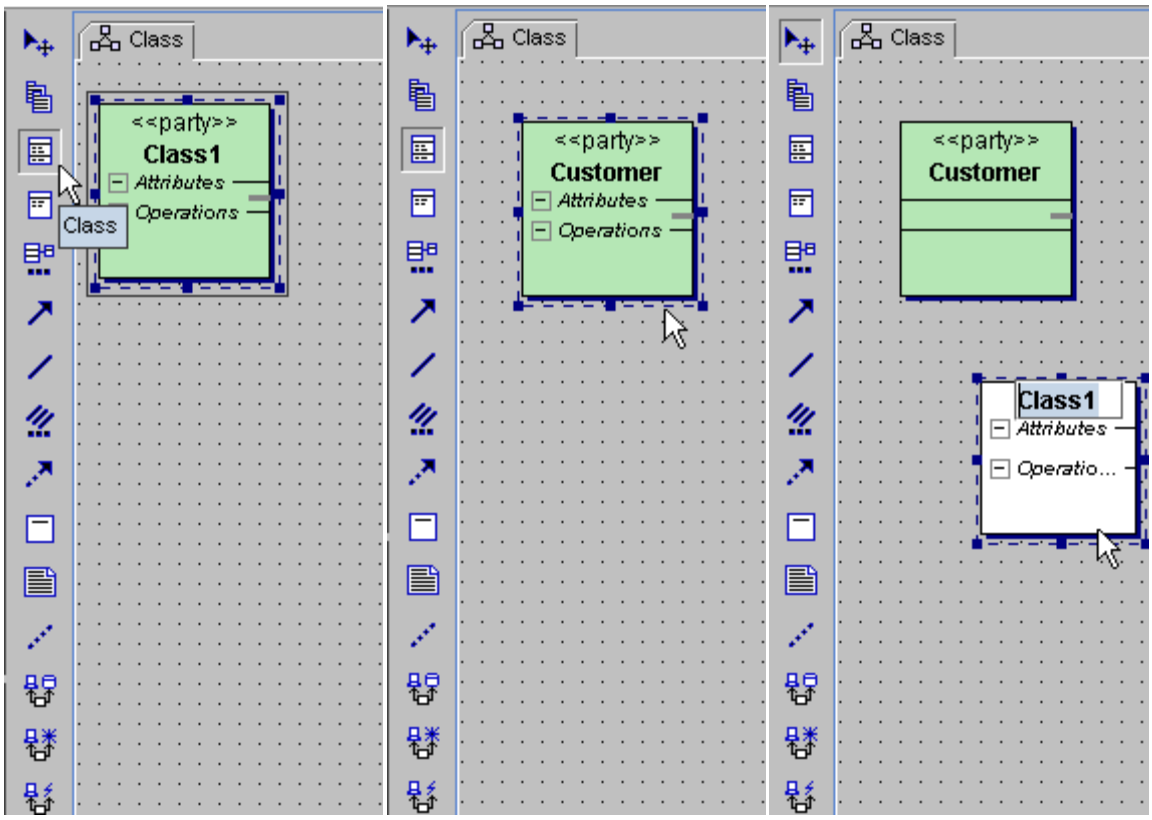
To control grid parameters:

1. Go the Main menu and choose Options | [scope] (where *scope* is *default, project,* or *diagram* depending on how broadly you want the settings to apply).
2. On the *Diagram* page of the Options dialog, expand the *Grid* node and set the *Show grid* and *Snap To Grid* options as desired.
3. Specify granularity of the Grid in the fields *Grid width* and *Grid height* in pixels.

Placing Nodes

To place a Node element:

1. Create or open a diagram
2. On the Diagram Toolbar click the icon for the element you want to place in the diagram. (Icons are identified with tool-tips.)
3. Move the pointer over the Diagram pane to the place you want to create the new element and click. This creates the new element and activates the in-place editor for its name. (*Note:* You can place multiple elements of the same type in one operation. See *Tips and Tricks* below.)



1. Click the desired element icon. Icon stays depressed.

2. Move pointer over diagram and click

3. New element created, in-place name editor activated

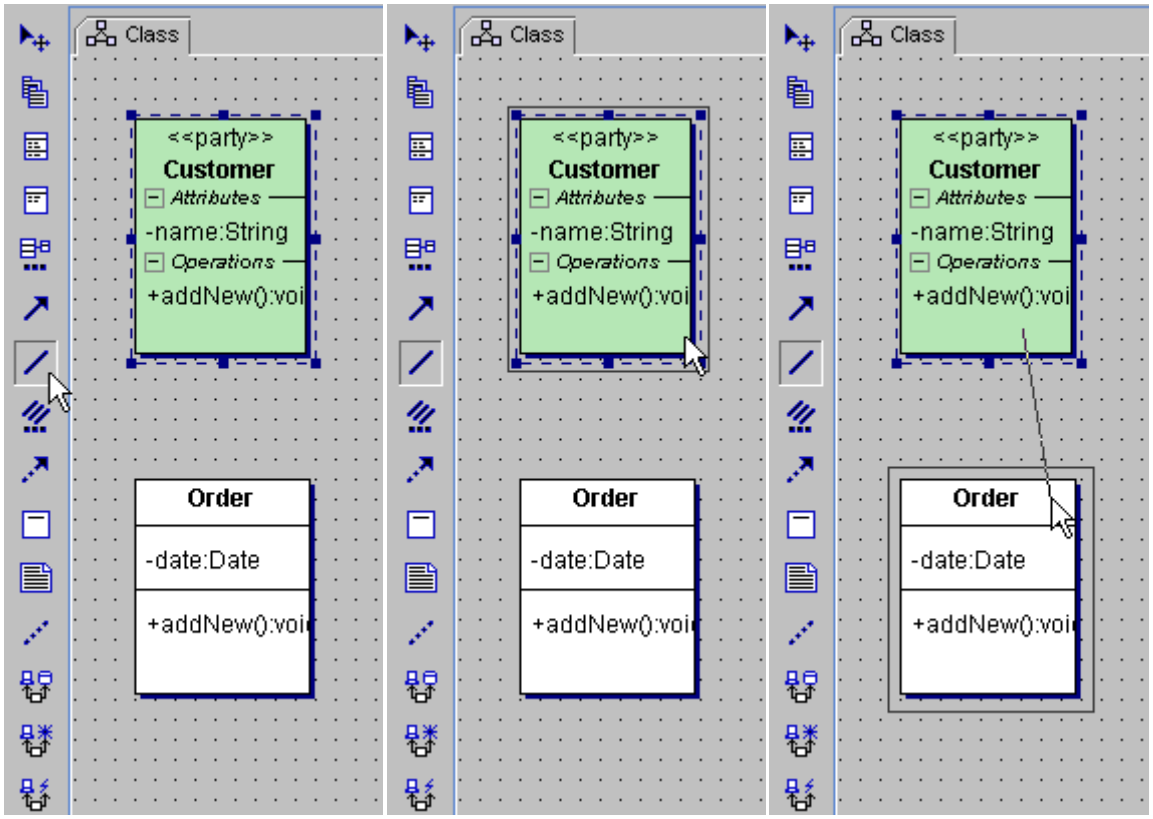
4. Alternatively, you can invoke the diagram speedmenu and choose the *New* node. Its drop-down list displays all basic elements that can be added to the diagram, and *Shortcut* option.

Drawing relationship links

To draw a relationship link between Nodes:

1. On the Diagram Toolbar click the icon for the type of link you want to draw in the diagram. (Icons are identified with tool-tips.)
2. To draw the link...

   - Move your pointer over the Node element where the link should originate.
   - Drag from the originating element to the destination element and drop the link.
   - If the destination element is out of reach, drop the link on the background. The *Choose Destination* dialog displays. Choose the destination element in the list and click OK to create the link. (**Note***:* You can place multiple links of the same type in one operation. See *Tips and Tricks* below.)



1. Click the desired link icon. Icon stays depressed.

2. Move pointer over first element, drag to second one

3. Drop when second element is highlighted

Bending points

If your diagram is densely populated, you can draw the links between the source and target elements bypassing the other elements on the way. This is how it's done... choose link icon on the toolbar and click on the source element. Then, drag the link line and click on the diagram background to create sections of the link, and finally click on the destination element.

Link to self

Choose Association link icon on the Diagram toolbar and double click on the element. This draws the link to self.

Bidirectional links

An association link between two classes can be converted to a bidirectional link. To do this, select the link and invoke *Choose Pattern* command on the speedmenu (or press CTRL+R). In the *Choose Pattern* dialog select Bidirectional pattern, adjust members if necessary and press *Finish* to complete operation. This adds *Split up* checkbox to the list of pattern parameters.

If there are two classes on a diagram associated with each other, both associations can be coupled into a single bidirectional link. To do this, select both associations and apply Bidirectional pattern to them.

All usual operations are applicable to the bidirectional links: you can move, reroute or delete them as required.

To split one bidirectional link into two separate associations, select the bidirectional link and invoke Choose Pattern dialog. Set the flag *Split up* and press *Finish*. The link falls apart into two associations.

Link labels

For better lucidity of a diagram, add textual labels to association links. To do that, invoke the link's speedmenu and choose *Label* command. This brings in an in-place editor where you can enter the link description. If the description is too long , it is possible to wrap it. This facility is controlled by the *Wrap link labels text* flag on the *Diagram* page of the *Options* dialog. When this option is selected, and the label exceeds a pre-defined length in pixels (also specified in the same place of the *Options* dialog), the label text wraps and displays in multiple lines.

Labels can be oriented along the links. This behavior is controlled by the option *Show labels oriented along the links* in the *Diagram* page of the *Options* dialog. However, oriented labels and multi-line labels are alternative. If oriented labels are selected, word wrapping is disabled.

See also

Manipulating diagram elements
Editing properties
Managing diagram layout

Tips and tricks

Constructing diagrams with *Together* is straightforward and easy to master. For newcomers, this section details some techniques that you may find helpful as you work.

Adding multiple elements: You can place a number of nodes of the same type in the diagram without returning to the toolbar. These will all have default names which you can then edit in-place or in the properties Inspector for each one.

- With the depressed CTRL key, click on the Toolbar button for the node you want to create (the button remains depressed).
- Release CTRL key.
- Click the desired location in the Diagram pane. The new element is placed on the diagram at the point you click (the button remains depressed)
- Click the next location in the Diagram pane. The next new element icon is placed on the diagram
- Repeat previous step until you have one less than the desired number of elements of that type.
- To place the last element in the series, click once more on the background and close the inplace editor for the element name.
- Click "Select" toolbar button or just press ESC.

Anti-aliasing: When zooming in a diagram, you can observe piecewise-linear lines. To dither the lines being drawn, set *Antialias graphics* flag in the *Diagram* page of the *Options* dialog.

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

In-place editing: You can edit the *name* property of elements, members, and links in place. Click once to select the name label, then again to activate the in-place editor. For Attributes and Operations you can type in either the full declaration statement or the element name only. The declaration must be full correct syntax for the project's target language.

Properties: Clicking on the diagram's background displays the diagram's properties in the diagram Inspector. Selecting an element in the diagram displays its properties in the element's Inspector. View Inspectors using Properties on the speedmenu of a selected element or the diagram background.

Selecting: Click on any element in the diagram to select it.

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

Speedmenus: Many things that you will want to do with elements or the diagram as a whole can be done from the different speedmenus. Some of the operations include:

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

Right-click on diagram elements, including class members, for access to element-specific operations on the respective speedmenu. Right-click the background to access the diagram's speedmenu. (**Note:** The speedmenu of a selected element is duplicated in the main Object menu.)

Note that if you select an element on diagram and wish to invoke its speedmenu, you can by chance loose the focus and get the speedmenu of some internal element. To avoid this situation, use SHIFT+Right Click. This invokes the speedmenu of the desired element and preserves the current selection.

Undo/Redo: Undo and Redo work with changes that affect the visual diagram, layout of elements, adding of Attributes and Operations, etc. These features do not work when the change to the diagram involves the creating or renaming a file or directory-- creating and renaming classes, interfaces or packages for example.

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

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

Viewing: Scroll the Diagram pane horizontally and vertically using its scrollbars or the Overview tab.

- Resize the pane vertically by dragging its lower edge up or down.
- Resize the pane horizontally by dragging the separator located between the Toolbar and the Explorer pane, or by resizing the window.
- Enlarge your work area in the Diagram pane by hiding any or all of the other panes using the View menu or Main Toolbar.
- Modify the Zoom level (magnification) using the Diagram speedmenu or by dragging a corner of the shadow in the Overview tab.

Zooming: you can obtain the required magnification on the Diagram pane. This is how it's done: select Zoom lens button and click on the Diagram pane to zoom in, depress ALT key and click on the Diagram pane to zoom out. Alternatively, choose Zoom command on the diagram speedmenu, or use keyboard shortcuts. More sophisticated zooming capabilities are available on the Zoom node of the diagram speedmenu: *Zoom to Selection, Fit in window* , *Fit 1:1*.

Manipulating Diagram Elements

Once you have placed elements into a diagram you may find you need to manipulate them visually. If you experiment a little you should find this easy and intuitive to do. Most manipulations involve dragging with your pointing device or executing speedmenu commands of selected elements. If you want to know more about some of the common visual manipulations, check the sections below.

Moving elements and drag-drop copying

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

You can use the same technique to copy members to different classes and interfaces by pressing CTRL before initiating the drag. When you move an element, it is *deleted* from the source package or element and moved into the destination package or element. *Copied* elements are *not deleted* from their source.

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

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

Full drag-and-drop support

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

Copying and "cloning" elements

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

> **Copy** an element by selecting it and using the Copy command on the speedmenu, or Edit | Copy command of the main menu
> **Paste** an element by selecting it's destination (element or diagram background) and using Paste on the destination's speedmenu, or Edit | Paste command of the main menu.

Note that Cut works the same way except the source item is deleted once the Paste operation is complete.

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

Note that you can perform cut, copy, paste, and clone operations from the Explorer using the appropriate speedmenu command of an element selected there.

Resizing node elements

Drag the corner of a selected diagram element to the desired size. The element grows or decreases in the direction of the cursor.

If you hold Shift key depressed while dragging the corner of a selected element, the element grows or decreases uniformly in all directions.

Manually resized Node icons shift to an automatically optimized size when their contents change, when members are added or deleted, for example. Restore the default size with the Layout | Actual Size command on the element's speedmenu.

Changing link routing

You can alter the routing of a relationship link's line by selecting the line and dragging any point on it in any direction. To remove bending points created this way, select Layout | Route Links from the speedmenu of either connected Node.

Changing link destination

To change the source or destination node of a link:

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

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

Related topic

Managing diagram layout

Standalone Design Elements

Together allows users to work with diagrams and diagram elements in a really flexible way. To keep the projects nice and clean, the design elements are stored inside the diagram packages. However, this impedes sharing of the elements between team members and storing the elements in a source code control system.

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

Using Standalone Design Elements simplifies visual analysis and facilitates exchange of design elements among the team members through the version control system, email etc.

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

Creating SDE's

There are two possible ways to create SDE's:

1. In the General tab of the Options dialog, check the flag *Create new design element as standalone*. With this option selected, any design element created on a diagram, is added as a shortcut, and appropriate `.ef<shapetype>` file is written to the physical package of this diagram.

2. Alternatively, when this option is unselected, you can copy a design element on the diagram and add it to the model as an SDE. To do that, right click on a package where you want this element to be added, and choose *Paste as standalone* on the package speedmenu.

Integrating SDE into a diagram

To integrate a standalone design element into a diagram, select this element in the Explorer's Model tab and paste to the diagram pane. The element adds to the diagram like a regular design element, and appropriate node is removed from the Explorer.

Using SDE's

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 in the Directory tab of the Explorer: for example, `Actor1.efActor, Actor2.efActor, Actor3.efActor`.

If this option is unselected, and you still want to make use of an existing or imported standalone design element, select this element in the Model treeview and choose *Copy* command on the node speedmenu. Next, right click on the target diagram and choose *Paste/Paste shortcut* on the diagram speedmenu.

See also

Using Together with a Version Control System

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.

Using the automated layout features

The Diagram speedmenu 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 a diagram element:

Layout | Selected, Layout | Selected for Printing repositions only selected element as just described.

Creating your own 'manual' layout

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

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

If you need to know how to do these tasks, see Drawing Diagram Elements: Tips and Tricks.

Diagram layout tips

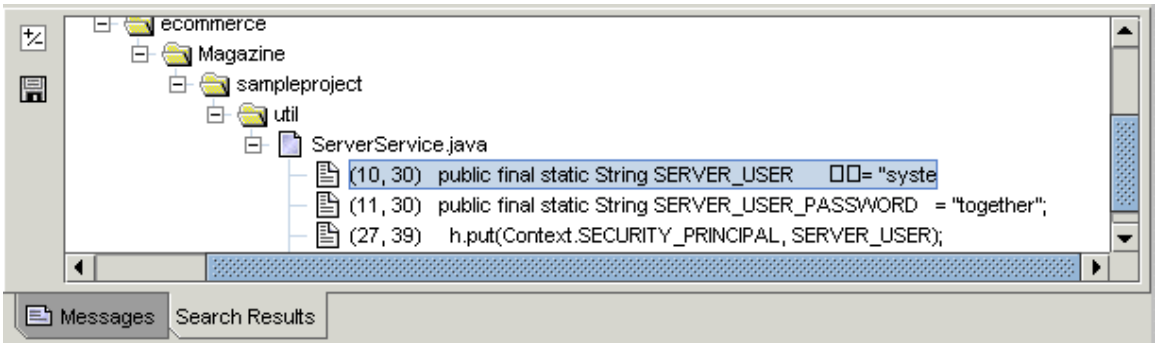
Manual layouts are saved when you close a diagram or project and restored when you next open it.

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.

Searching on Diagrams

Together allows to search through the diagrams and projects. Commands are available on the Search item of the main menu. Search adds new tabs to the Message pane to display the results. Icons at the left allow to expand / collapse the treeview nodes  , and to save the search results  .



This section gives just a brief overview of search and replace facilities. You can find the detailed description of controls and options in the appropriate topics of the Context Help.

Search on diagrams

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

Search by query

This feature provides advanced search tool. Selected element is sought for in the specified scope, according to the user-defined query or filter.

Search / Replace

These commands provide a flexible and powerful facility of searching and replacement in the specified range. Search results display in the special tab of the Message pane as a treeview of found elements.

Search for Usages Dialog

This command performs search for the occurrences of a language construct in the specified scope. Search results display in a special tab on the Message pane as an expandable treeview.

Update Dependencies

Diagram Editor draws links between the classes and interfaces within the same diagram - for example, a class extends the other class, or implements an interface... However, if the diagram contains packages, the dependencies between those packages are not drawn automatically.

Update Package Dependencies option of the Diagram speedmenu builds links between the packages. A link between two packages means that there is some sort of dependency: implementation, inheritance, usage - say, real dependencies are only taken into account. For example, if a certain class is imported but never actually used, this is not considered a dependency.

This is how it works... Select one or more packages in the Diagram Editor and choose **Update Package Dependencies** on the diagram speedmenu. Together will build all the links between the selected package(s) and the rest of the diagram. If none is selected, all packages are considered selected. In this case Update Package Dependencies provides two options: *Current Diagram* (quite self explanatory) and *All* (arm yourself with patience - this builds links for the entire project).

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

The process of building dependency links is quite time consuming... However, its speed is configurable. 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` causes generating the links between subpackages.

Opening diagrams for editing

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 Diagram pane. Tabs display the diagram name and diagram type icon. (The icons correspond to those shown in the Model tab of the Explorer.) Switch between open diagrams by clicking the desired tab.

To open a diagram for editing:

- Select the Model tab of the Explorer if it is not currently selected.
- In the Model tab, find the diagram you want to open.
- Right-click on the diagram in the treeview and choose Open or Open in New Tab from the speedmenu.

Tips

- If you want to work with several diagrams, right click on each diagram icon and select Open in New Tab.

- Icons for UML diagram types look like the UML symbol for the main element of each type. For example, the icon for a Component diagram looks like the UML icon for a component.

- You can also open diagrams from the Main menu with **File | Open** .

- Speedmenu "Open" replaces the contents of the current diagram (if any) with the diagram you are opening.

- Double-clicking on a diagram node in the Explorer opens the diagram in the currently focused diagram page replacing any open diagram.

Closing open diagrams

You can close the current diagram by choosing **File | Close** from the main menu, or by right-clicking on the diagram tab and selecting the **Close** speedmenu.

You can close all open diagrams at once by choosing **File | Close All** from the main menu, or by right-clicking on the diagram tab and selecting the **Close All** speedmenu.

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 *properties Inspector* (further referred to as Inspector). The content of Inspectors varies depending on what is selected. Generally an inspector presents several tabbed pages representing different categories of properties. For example, the **class** Inspector has the following tabs:

Properties: General properties of the class (name, stereotype, abstract, etc.) Displays two columns:

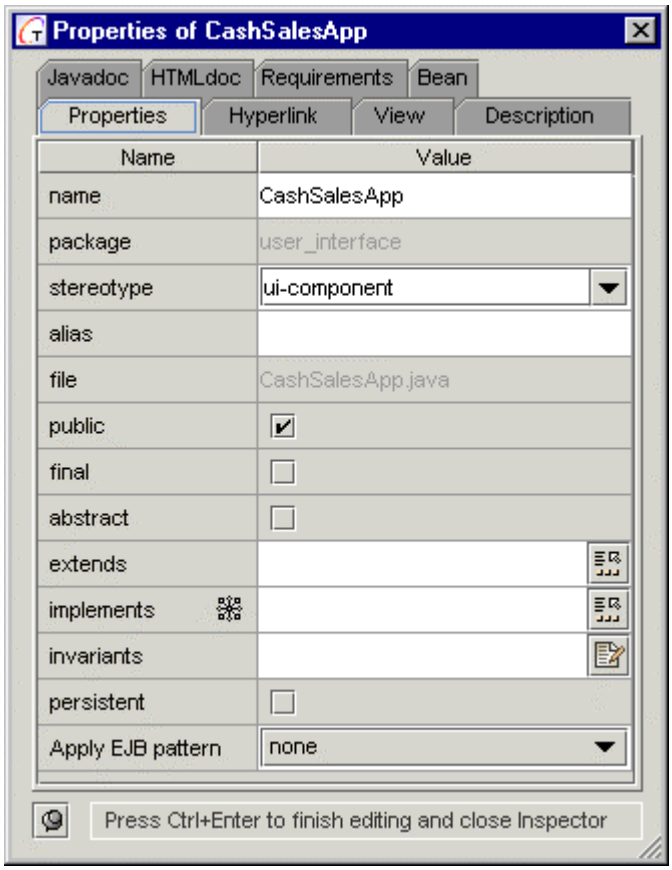
> **Name** shows the name of each property
> **Value** displays the value or setting of the property named in the same row of the Name column.

Hyperlinks: Intra-project and URL hyperlinks to related information (see Hyperlinking diagrams)
View: Visual properties of the diagram icon (color, etc.)
Description : Source code comments
Javadoc: Properties used for Javadoc generation (author, version, etc.)
HTMLdoc: source code comments in HTML format
Req: Information tracking requirements (number, priority, difficulty, etc.)
Beans: Optional JavaBean properties (if class is a JavaBean)

Property editors

In the general Properties of elements and diagrams, you can enter a value for string type properties by typing a value in the edit field in the Value column. Some string type properties display a "browse" button beside 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
- 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.



Tips for Editing Properties

Applying changes: Changes to property fields are applied when you:

press *Enter* (inspector dialog remains open)
press *Ctrl+Enter* (inspector dialog closes)
exit the edited field, or...
when the open inspector dialog loses focus.

Important: Note that if you click the dialog's close button while changes are pending to a field, *the changes are not saved.*

Inspector contents: In the Diagram pane, select a diagram element to load its properties in the appropriate Inspector. (If you select nothing, the background is selected by default and diagram properties are loaded.)

Inspector invocation: Use the element or diagram speedmenu (choose Properties) to display the Inspector. Also accessible from the Object menu. Keyboard shortcut is Alt + Enter.

File chooser buttons: Use keyboard shortcut ALT+INSERT to edit the fields with file chooser buttons.

Modifying properties: To modify a property value, click on 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 properties are disabled for editing.

Complete operation: Press *Enter* to complete editing in a text field. *Ctrl-Enter* applies modifications and closes the inspector.

In-place editing: You can edit the name property of nodes in place on the diagram icon itself.

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

Exit: Press *Esc* to quit the Inspector.

See also

Property Inspectors

Editing Properties in-place

Working with diagrams encourages inplace editing of the properties, in addition to the inspector edit. Each diagram element has an initial string with a certain set of properties. However, some of these properties are not displayed, and become available only in course of inplace editing. Users may modify all properties, or some of them, so that the underlying engine completes changes.

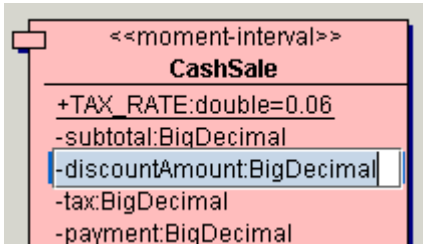
To edit in-place, double click on the selected element, choose *Rename* on the element speedmenu, or just press F2 for the selected element. This brings in a highlighted text string with a cursor that allows modification, unless a diagram is read-only. The changes are applied by pressing <Enter>, or clicking on another element, which causes relevant code generation.

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

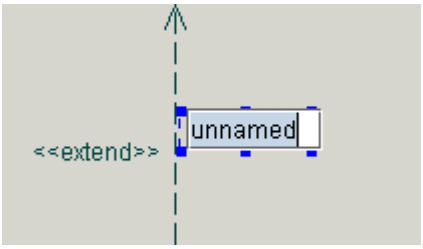
In case of incomplete entry, the omitted visibility modifiers, attribute types, return types of the operations are replaced with their existing values. For example, if an attribute was private and the visibility is not entered, then it will be kept private. If no value was specified before, then default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

Almost all strings that show up on the diagrams can be modified on the fly.

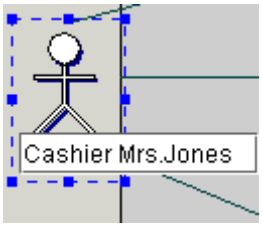
Note: shapetypes and link types are not editable.



Inplace edit of a member



Inplace edit of a link comment



Inplace edit of an actor

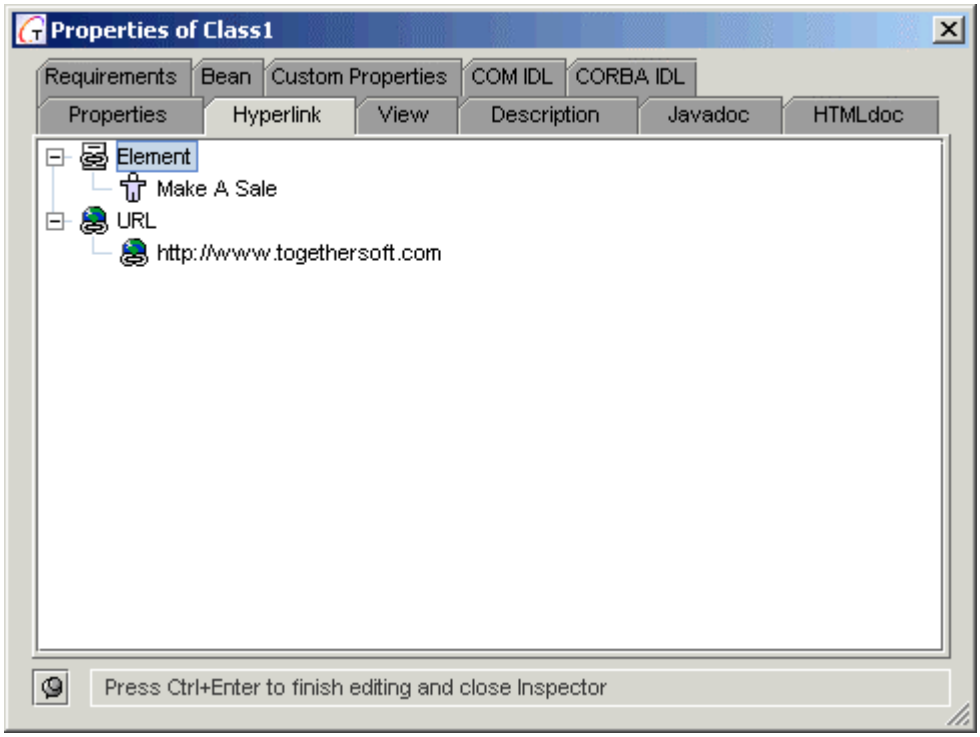
Hyperlinking diagrams

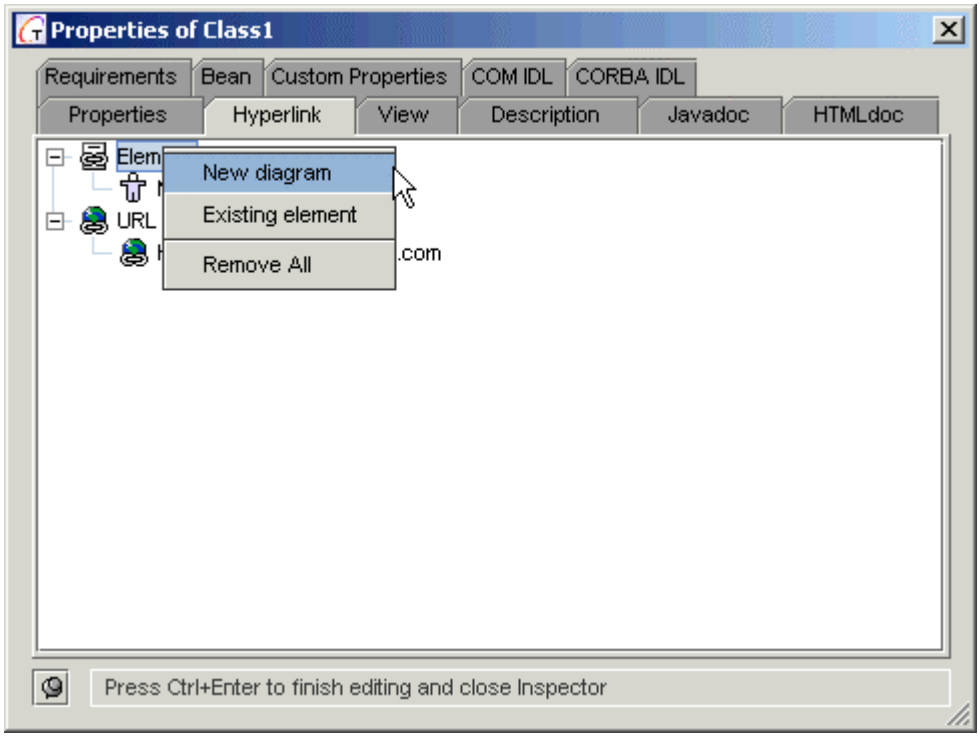
This topic explains the types of hyperlinks you can create with Together and why you might want to create them. There are also step-by-step instructions for **creating**, **viewing, removing**, and **browsing** hyperlinks.

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

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

You create, view, remove, and browse hyperlinks with the Hyperlink Tab speedmenu.





Add or remove hyperlinks using the speedmenus of the link-type nodes of the Hyperlinks tab

Why use hyperlinking?

- Link things that are generalities or overviews to specifics and details.

- Create browse sequences leading through different but related views in a specific order; create hierarchical browse sequences.

- Link descendant classes to ancestors; browse hierarchies

- Link diagrams or elements to standards or reference documents or generated documentation.

- Facilitate collaboration among team members

How to create hyperlinks

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

Hyperlinking to a new diagram

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. When creating a new diagram this way, the main difference in procedure is launching the New Diagram dialog from the Hyperlink tab instead of the Main menu. Of course, the new diagram is then hyperlinked to your originating element by default.

To create a new hyperlinked diagram:

1. Open an existing diagram from which to create the hyperlink (or create a new diagram).

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

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

Hyperlinking to an existing diagram or diagram element

You can 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. If you want to link to the diagram as a whole, click on the diagram background to deselect all elements.
3. Choose Properties from the speedmenu and select the Hyperlinks tab in the Inspector.
4. Right-click on the node named *Element* to display the speedmenu.
5. If you want to link to a diagram or a diagram element, choose **Existing element**.
6. Select the desired diagram or element in the dialog that is now displayed. For element selection, you can expand diagram nodes in the selection dialog's treeview.
7. Click OK to close the dialog and create the link.

Hyperlinking to a URL or file

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

To create a hyperlink to a file or URL:

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. If you want to link to the diagram as a whole, click on the diagram background to deselect all elements.
3. Choose Properties from the speedmenu and select the Hyperlinks tab in the Inspector.
4. Right-click on the node named *URL* to display the URL dialog.
5. If linking to a file, enter the complete path, or browse for it. If linking to a URL, enter the URL.
6. Click OK to create the link.

Viewing hyperlinks

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

To check if anything is hyperlinked 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 to something else display in blue font. If you want to know 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 have found the defined hyperlink(s) for a selected diagram or element, you can use the Hyperlink tab's speedmenus to browse to the linked resource(s).

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

To browse **any** *hyperlink:*

1. Right-click on the desired hyperlink in the Hyperlink tab.
2. Choose Open from the speedmenu.

Removing hyperlinks

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

To remove any hyperlink:

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

Annotating diagrams

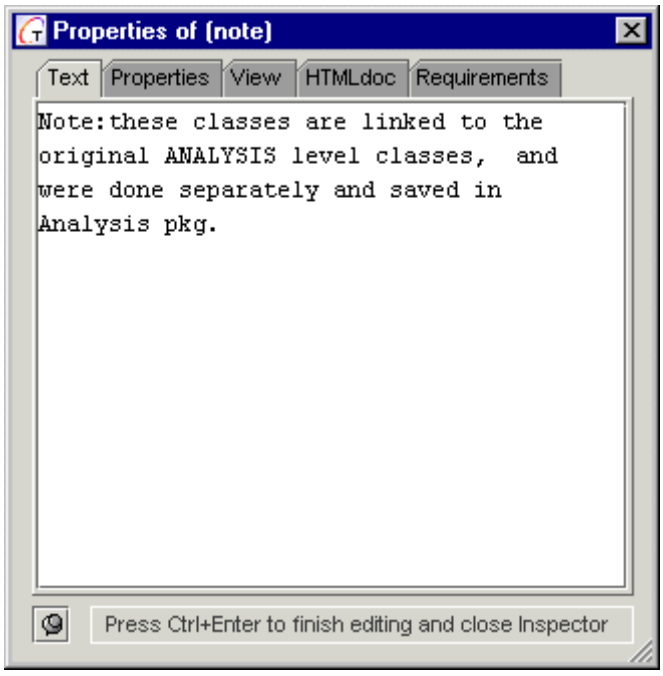
The Diagram Elements toolbar always displays *Note* and *Note Link* buttons that place Notes and Note Links on the diagram. Notes can be free floating, or you can draw a Note Link to some other element to show that a Note pertains specifically to it.

Using Notes

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

In the Note Inspector you can:

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



The Note Link

Notes are automatically included when you generate HTML documentation. The text of Notes linked to Class diagram elements does not appear in the source code. Use the Description tab of the Class Inspector to enter class description, and enter source code comments directly in your code using the Editor pane.

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

Inspector Documentation tabs

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

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

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

Besides these tabs that allow view and edit comments as plain text, there is the *HTMLdoc* tab that displays comments in the Browser mode.

Saving and Copying Diagram Images

You can save or copy entire diagrams or their selected parts for further re-use. There are three possible behaviors in Together:

Copy - Paste within Together

Select the required part of a diagram and copy it by choosing *Copy* command on the Edit menu or selection speedmenu (you can also use CTRL+C hotkey). Next, *Paste* the selection to the target location. This feature works in Together only.

> **Note**: If link labels don't display after pasting, press F5 with the Diagram pane having the focus, to refresh the diagram.

Copy image

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

You can opt to copy images in bitmap or WMF format. To choose the desired format, set/clear the flag *Copy diagram image into clipboard as bitmap* in the *Options | General* page. Bitmap format cures such problems of WMF format as distorted fonts, wrong conversion of the national fonts, lack of Java 2D functions support. Besides that, some rasterized graphics applications don't accept clipboard metafiles, but recognize 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 aviod this, use Paste Special command, rather than Paste. However, if you copy a diagram in WMF format, it can be reproduced in PowerPoint by the usual Paste command.

Save image

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

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

Printing Diagrams and Source Code

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

Setting Print Options

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

To check Together's default Print options:

1. From the main menu bar, choose Options | Default to display the Default Options dialog.
2. Select the Print tab and check the settings.

To check Print options for the current project:

1. From the main menu bar, choose Options | Project to display the Project Options dialog.
2. Select the Print tab and configure settings as desired.

To check Print options for a specific diagram:

1. Open the diagram for editing.
2. Choose Diagram Options from the Diagram speedmenu (or select Options | Diagram from the main menu) to display the Diagram Options dialog.
3. Select the Print tab and configure settings as desired.

For further information, consult the topics under Configuring Together chapter.

How to print diagrams

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

To print a single diagram:

1. Open the diagram in the Diagram pane (project must be open).
2. If there are several open diagrams, click the tab of the one you want to print
3. Choose **File | Print Diagram** on the Main menu to launch Print Diagram dialog.
4. Click OK

To print several diagrams at once:

1. Open the diagrams in the Diagram pane (project must be open).
2. Click on the workspace of any open diagram.
3. From the Main Menu, choose **File | Print Diagram** to launch *Print Diagram* dialog.
4. Select the All Open option and click OK.

How to print text

To print source code:

1. Make sure the Editor pane shows up (check Editor pane checkbox in the View menu, or press F9).
2. Click on the desired element of a diagram to display its source code in the Editor pane, or just type your text in the active tab of the Editor pane.
3. With the Editor pane having the focus, choose **File | Print File** from the Main Menu, to start *Print File* dialog.
4. Make necessary settings and click OK.

Tips and tricks

- The Options dialogs provide context-relevant Help for each option directly in the dialog.
- If you are running *Together* under Windows, set your printer's True Type Fonts property to *Print As Graphics* in the Font tab of the Windows Printers dialog before printing diagrams.
- To make sure that all diagram elements fall within page boundaries, run the auto-layout command Layout All for Printing from the Diagram speedmenu before printing a diagram. If you have done a manual layout, restore it using *Undo* after printing. Be sure to run Undo *before closing the diagram* or your manual layout will be lost.

Troubleshooting

Printing problems can stem from a number of things unrelated to a particular application program such as *Together*.

Some printer drivers on the Windows platform can cause printing problems. The diagram image is scaled up or down on the paper and doesn't match the visual boundaries. This problem was reported with the Sun JVM versions 1.1.5 and 1.1.6 under Windows (both 95 and NT). They consistently occur with the MS JVM build 2829 (SDK 3.0) and build 2925 (SDK 3.1).

On the other hand, Sun JVM 1.1.7B does not cause this problem. Upgrade to this version or a later if you use Java from Sun.

For other VM versions, *Together* provides a workaround for printing in Windows 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 as described above, try adjusting the value or commenting out this property. Then try printing again.

For each test, modify and save the `misc.config` file, call Tools | Reload Options (assuming Together is running), and finally invoke File | Print diagram.

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 Auto-layout | All for Printing.

Tip: If you want to preserve a diagram layout that you have done manually, you can invoke Undo after you 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.

See also

Print Diagram dialog

Print File dialog

UML Diagrams

Creating UML Diagrams

Using the main menu or toolbar

You can use the **File | New Diagram** menu command or the **New Diagram** icon of the toolbar to create a new diagram in the directory/package currently selected in the Project Explorer.

1. Select the destination package in the Project Explorer *Model* tab.

2. Choose File | New Diagram from the main menu bar, or click on New Diagram icon of the toolbar to display the *New Diagram* dialog box.

3. Click the icon for the type of diagram you want to create (only those diagrams enabled by your license are available). Use edit control to edit the default diagram file name, and use *Browse* button to specify *Package name*. Optionally you can uncheck *include in current diagram* checkbox and/or write comments in the *Description* edit box.

If you only specify a filename, the diagram file is created in the Directory or Package currently selected in the Project Explorer. If you use Browse button, the destination path you specify overrides the Project Explorer selection.

4. Click OK to create the new diagram file and open it for editing in the Diagram pane.

5. Use in-place editing in the Project Explorer to rename the diagram if desired.

Using Project Explorer speedmenu

1. Click on the destination package in the Project Explorer *Model* tab.

2. In the *New* group on the speedmenu select Diagram.

This displays the *New Diagram* dialog box.

Using the Hyperlink feature

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

1. Right-click on the background of the Diagram pane of an open diagram and select Properties speedmenu.

2. In the opened Inspector dialog click on the Hyperlink tab.

3. Right-click on the Element item and select New Diagram speedmenu.

Use Case Diagrams

Use Case diagrams provide a way of describing the external view of the system and its interactions with the outside world. Actors, are a representation of the outside world, and they could be people or computer systems. A use case is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors), together with actions performed by the system. A use case diagram shows the relationship among actors and use cases within a system.

Creating and drawing Use Case diagrams

If you need to learn how to create new diagrams in a project, or the techniques for placing elements and drawing links, consult the User's Guide topics found under " Working with Diagrams: Basic Diagram Techniques" in the Table of Contents. See Related Topics below.

Key elements and properties

Content of Use Case diagrams

Use Case diagrams usually contain:

	Use cases
	Actors
	Relationship links:
	Communicates
	Extends
	Includes
	Generalization
	System boundary
	Notes and note links

Elements of Use Case diagrams are defined by the UML and Together provides these on the Use Case Diagram Toolbar. Use Tool-tips to identify the different elements on the toolbar.

Actor

An Actor element characterizes the role played by an outside object; one physical object may play several roles and therefore several actors may model it. You can select the type of class the actor will belong by setting the *stereotype* property. Select the stereotype from the drop-drown list or enter a new one.

Use Case

A Use Case is a descriptor for a set of action sequences performed by a system (including variations thereof) that produces an observable result of value to one or more actors. You can view Use Case properties in the Properties Inspector. Properties you can define include stereotype, explanation, pre and post conditions, and flow.

Links

There are several standard relationships among use cases or between actors and use cases. These are:

Communicates: The participation of an actor in a use case. This is the only relationship between actors and use cases. It can also be used between Actors to indicate necessary communication between them.

Extends: An extends relationship from Use Case A to Use Case B indicates that an instance of Use Case B may include (subject to specific conditions specified in the extension) the behavior specified by A. Behavior specified by several extenders of a single target use case may occur within a single use case instance.

Includes: An *includes* relationship from Use Case A to Use Case B indicates that an instance of the use case A will also include the behavior as specified by B.

Generalization: denotes a relationship in which objects of a specialized element can be substituted for the objects of a more general (parent) element.

Working with Use Case diagrams

Outside of the basic mechanics of drawing the diagram to construct your use case views, there are several conceptual things you should understand about Use Case diagrams:

How to create browse-through sequences of Use Case diagrams using Hyperlinks
How to show another diagram in a Use Case diagram

Creating browse-through sequences of Use Case diagrams

Typical uses of Use Case diagrams include modeling system context and modeling system requirements. Often you begin at a high level and specify the main use cases of the system itself: "Conduct Business" for example. Then you break the main system use cases down further. For example, the "Conduct Business" use case might have another level of detail that includes the use cases: "Enter Customers" and "Enter Sales". Once you have broken things down to the desired level of granularity, it's useful to have a convenient way of "drilling down" or "rolling up" to grasp the scope and relationships among the system's use case views.

Together's Hyperlinking feature makes it easy to create browse-through sequences comprised of any number of Use Case (or any other) diagrams. You can link entire diagram 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. You can browse the hyperlink sequence to follow the relationships between the Use Case diagrams.

You aren't confined to such sequences, however. Hyperlinking is completely flexible and you can use it to link diagrams and elements in the ways most meaningful to you. For example, you might create a hierarchical browse-through sequence of Use Case diagrams, and within the diagrams themselves create hyperlinks that follow a specific Actor through all use cases that involve it.

The mechanics of creating and browsing hyperlinks are covered in Working with Diagrams: Hyperlinking diagrams.

Showing other diagrams in a Use Case diagram

You may find it more useful to show relationship of a Use Case diagram to some other diagram as part of the Use Case diagram itself, that is, *not* using the Hyperlinking feature. You can do this by placing a *shortcut* to another diagram into the current diagram.

An diagram shortcut is represented with a Package that displays the icon of the diagram type it represents and the diagram name. The Package also lists the elements of the diagram.

To create a shortcut to the current diagram:

1. In the Model tab, select the node for the diagram you want to import and choose Copy from its speedmenu.
2. Right-click on the diagram background and choose Paste from the speedmenu.

You can open the diagram represented by the shortcut using the speedmenu of its Package icon.

Tips and Tricks

- Using the *Clone* command on the context 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.

- Using *Add Shortcut* command on the diagram's context menu, you can reuse any already created elements in other Use Case diagrams.

Related Topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Customizing property inspectors

Class Diagrams

Class diagrams are the most common type of diagram in many object models. They show the static structure of the system, in particular, the things that exist (such as classes and types) and their internal structure. Class diagrams also depict collaborations and relationships between classes, and inheritance structures. A Class diagram may also show instances (or objects) and links between objects,

- Class diagrams define a **vocabulary** for the system, and may also be the basis for Component and Deployment diagrams.
- Class diagrams are the basis for visualization, specification, and documentation of the structure of the system and also for the **implementation** (Class diagrams that are round-trip engineered by *Together).*
- Class diagrams that show only **Packages** are referred to as *Package diagrams*.

If you need to learn how to create new diagrams in a project, or the techniques for placing elements and drawing links, consult the User's Guide topics found under " Working with Diagrams: Basic Diagram Techniques" in the Table of Contents. See Related Topics below.

Content of Class diagrams

Content of the Class diagrams is language-dependent. Elements of Class diagrams are defined by the UML and UML extension. *Together* provides these on the **Class Diagram Toolbar**. Use **Tool-tips** to identify the various elements on the toolbar.

Class diagrams usually contain:

	Packages
	Classes
	Interfaces
	Classes by pattern
	Objects
	Notes and note links
	Relationship links
	Dependencies
	Generalizations / Implementations
	Associations
	Links by pattern

Depending on the current project language, Class diagram toolbar displays icons specific for the selected language:

Java		Entity EJBs
		Session EJBs
		MessageDriven EJBs
IDL		Struct
		Valuetype
		Exception
C++		Aggregation

Packages represent underlying physical packages that are part of your project, subsystems of a larger system, or logical groupings of information from any resource available to your project. Since *Together* diagrams can contain any kind of elements, diagrams can also be used as general grouping mechanism for creating different views.

A diagram imported into another diagram displays as a Package with appropriate stereotype-icon. For more information, see Creating Views.

An EJB Application diagram or a Web Application diagram imported into an Enterprise Application diagram displays also as a Package with appropriate stereotype-icon, see Creating EJB Shortcuts.

Key elements and properties

Class

A class is the descriptor for a set of objects that have the same attributes, operations, relationships, and semantics. Classes usually implement one or more *interfaces*. Classes capture the vocabulary of the system.

In *Together*, classes have various properties in addition to the usual members. Most of these are defined by the UML, but some such as *alias* are provided as enhancements to communication. Some properties of interest are listed below:

Key Class properties

Property	Description
Name	Every class has a name to distinguish it from other classes. Name is a text string. You need only enter the *simple name* for classes (as opposed to qualified name). Because classes are coupled to round-trip engineering, **name must be a valid identifier for the target language**.
Stereotype	Provides problem-specific extension of the basic UML vocabulary for a class. You can enter a new stereotype in the property Inspector (Speedmenu | Properties) as you model, or choose from a list of the stereotypes defined in your configuration.
Alias	Optional alias replaces the Name in the diagram. Because spaces and other characters aren't allowed as class names by e.g., Java, using aliases can make your class diagrams more readable.

Showing Java Beans

To show classes as Java Beans on the Class diagram you have to turn on the bean support option:

1. Invoke the *Options* dialog,
2. Expand *View Management* node
3. Select the page *Java Beans / C++ Properties*
4. Set *Recognize Java Beans* option.

Java Beans, including Enterprise JavaBeans (EJB), are displayed as classes with two additional sections, one for properties, and the other for events to which the Java Bean can respond. When you select the Beans support option, all the classes on the diagram are rescanned. Those classes with get and set methods are displayed as Java Bean nodes on the diagram denoted by a Bean symbol by the class name. Display support for Beans is controlled in Default and Diagram Options.

Interface

An interface is a device that unrelated objects use to interact with one another. It defines a set of methods but does not implement them. A class that implements the interface agrees to implement all of the methods defined in the interface. Interfaces have the same properties as classes except for *Implements*.

Object

An Object represents an instance of a class. Objects are displayed on diagrams to facilitate communication about the model; they have no representation in source code.

Key object properties

Property	Description
Name	Name is any text string since Objects are not coupled to round-trip engineering.
Stereotype	You can assign Actor or Database as stereotype or enter any stereotype name you want.
Persistence	You can choose transient, static, or persistent or enter any stereotype name you want.

Entity EJB

Entity EJB creates elements in the visual model and generates the underlying source code for a default implementation of a persistent entity EJB with skeleton declarations.

Session EJB

Session EJB creates elements in the visual model and generates the underlying source code for a default implementation of a nonpersistent session EJB with skeleton declarations.

Message Driven EJB

Message Driven EJB creates elements in the visual model and generates the underlying source code for a default implementation of a message-driven bean with skeleton declarations.

Class by pattern

Class by pattern creates elements in the visual model using one of the pre-defined patterns or templates.
In C++ project it is possible to create a template class according to UML 1.3 specification, as a small dashed rectangle superimposed on the upper right-hand corner of the class rectangle.

Package

A Package is a group of items of the physical model. Packages may also show logical groupings of model content.
Packages often represent physical directories. If the default diagram for the project (specified in your configuration) is a Class diagram, it will contain Package icons for each physical subpackage under the project directory... essentially it is a Package diagram.
Other diagrams of any type may be represented in a class diagram. These appear as Packages with a visual stereotype, i.e., the icon representing the diagram type. Packages are represented in the Explorer, Diagram toolbar, and element selectors in dialogs with a smaller icon of the same shape:

Tips on packages:

Viewing Packages and content: The Model tab of the Explorer displays the packages and subpackages in the project. You can use the Explorer to navigate into them and view their contents.
Opening packages: Choose one of the Open commands on the speedmenu.
Renaming a Package: You can rename a package in place, or using the properties Inspector (Speedmenu | Properties). This change is propagated to all the source files.
Moving elements into Packages: Classes and interfaces can be visually moved to other packages by dragging from the diagram to the target package. For more information see Manipulating elements. Other packages, imported or compiled classes or interfaces cannot be moved in this way however. Pressing the CTRL key and dragging a class into a package imports the class into that package. The class name in italic appears in the package icon.
By default the package icon on the diagram displays its updated contents. You can use the class/interface context menu and add attributes and operations directly. The diagram option Subpackages controls how the packages are displayed in the diagram.
Moving nodes from Packages: You can move that class or interface to the current diagram, deleting it from its current package and moving it to its new package by dragging the class or interface name from the package element to the diagram background.
Dependency links: When you choose Update Package Links on the diagram speedmenu, *Together* scans each package on the diagram. If it finds an element within the package that is linked to an element of another package, then it creates a Dependency link between those package icons on the diagram.
This covers only source-code based elements. Inter-package dependencies between such elements as Objects, Notes, search-path imported classes, Use Cases, etc. are not recognized.
Deleting a Package: You can only delete a package using its context menu if all of its contents have been removed first. A physical package can't be deleted if it contains source or diagram files, files of non-Together types, or if other applications share the corresponding directory.

Links

There are several standard relationships defined by the UML and applicable in class diagrams. Together's class diagrams support:

Association: a structural relationship describing a set of object links. There are two specialized forms of Association:

- **Aggregation:** a special type of Association that denotes a whole-part relationship
- **Composition:** a form of Aggregation with strong ownership in which the parts may be created after the composite but once created, live or die with it.

Generalization: denotes a relationship in which objects of a specialized element can be substituted for the objects of a more general (parent) element.

Implementation: visually the same as Generalization and drawn with the same toolbar icon. Use to show the inheritance of the implementation of a more specific element

Dependency: indicates a semantic relationship between two (or more) model elements in which a change to the independent element may affect the dependent element.

Link by pattern: the link is chosen from the tree of links that already exists.

Note link: the relationship between the note and the diagram element that defines the content of the note.

Working with Class diagrams

Outside of the basic mechanics of drawing the diagram to construct your model, there are several conceptual things you should understand about Class diagrams:

How to show classes from packages that are part of your project
How to show classes from search path packages outside your project
How to define inner classes
How to import another diagram into a Class diagram
Showing Association, Aggregation, and Composition

Showing project content from different packages

Packages, classes and interfaces that belong to your project display in the Model tab of the Explorer. Their paths are defined in your project properties (File | Project Properties). If, when creating the project, the Class diagram was specified as default, the project package contains a Class diagram that maps out the packages belonging to the project. Depending on your specifications in the New Project dialog (and later Project Properties), the classes of these packages may be reverse-engineered into Class diagrams.

Assuming this is the case, the Class diagram having the same name as its containing package, shows the classes and interfaces in the package, along with relationships and dependencies. You can optionally add classes from any package in the project in these diagrams, or you may choose to create new Class diagrams in a package to show different views of the package contents and relationships to other packages. For example, you might create different Class diagrams to show classes for components or subsystems contained in multiple packages. Assume for the moment that everything in such views resides in a package that belongs to the project and thus displays in the Model tab.

To show classes etc. from another project package in a Class diagram:

1. In the Model tab, navigate to the package containing the element(s) you want to show.
2. Select the desired element in the Explore and copy it to the Clipboard.
3. Paste the element in the desired location in the diagram.
4. After copying all elements, choose *Update Package Dependencies* on the diagram speedmenu.

Showing classes on the search paths

When you create a project you can define directories any number of *search paths* whose content you may want to show in diagrams. For example, you might want to show things that reside on the standard for full Java classpath. Such resources "exist" for the project but are not included in generated HTML documentation as belonging to the project.

To show classes etc. from one of the search paths in a Class diagram:

1. Open or create a Class diagram.
2. Right-click on the background and choose *Shortcut* from the New node of the diagram speedmenu to launch the Add Shortcut dialog. This dialog shows all the content available for the diagram on the left-hand side, and all content residing outside the current package on the right.
3. Navigate to the resource you want to add and click the Add button. Repeat until you have added all the resources you want.
4. Click OK to close the dialog.

> **Tip:** If the resource you are looking for is not shown, it is probably not in the search paths defined in Project Properties. You can add resources to the search paths at any time by choosing File | Project Properties | Advanced and selecting the tab for search paths. For more information, see User's Guide: Creating and opening a project- Advanced mode.

Defining inner classes

The easiest way to create an inner class is to drag it over another class and drop it. This works well when the inner class already exists. If the inner class doesn't exist you can create it and define it within its parent at the same time.

To create and define a new inner class:

- Select the parent class.
- Choose New | Inner Class from the speedmenu.

Tips:

- You can define new inner classes from the speedmenus of classes in the Explorer.
- You can use drag-drop to remove inner classes from classes in the diagram.

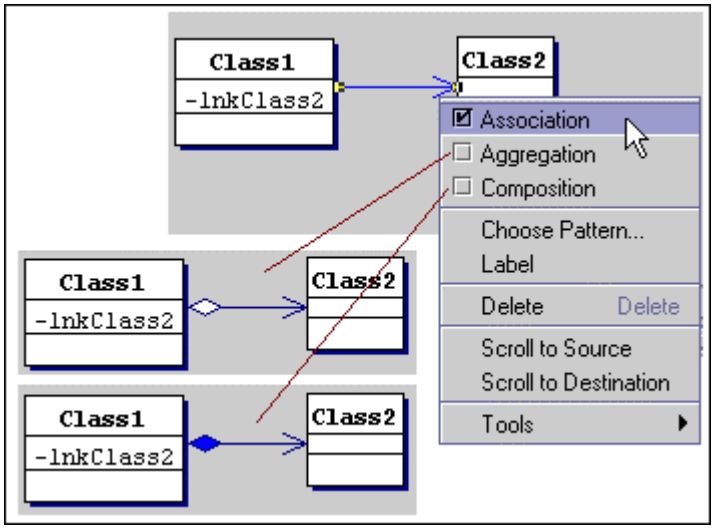
Importing other diagrams

You can import another diagram of any type into a class diagram creating an automatic link to that diagram. A diagram that is imported into another diagram is shown as a Package with a corresponding stereotype-icon. You can open the diagram from the context menu of the package icon. (See also: Packages below).

Showing Association, Aggregation, and Composition

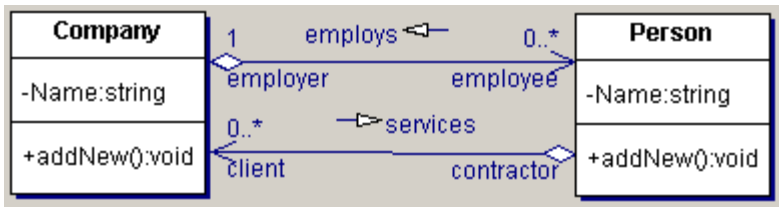
Use the Association icon  on the toolbar to draw all types of Association links. After drawing the Association link, select it and use the **link's speedmenu** to specify which type of association to show in the diagram. (You can also change the type by applying the appropriate pattern to the link.)

You can show directionality of Association, Aggregation, and Composition links by setting the *directed* property in the link's properties Inspector (Speedmenu | Properties).



How to show different Association types. The directional arrow is specified in the *directed* property of the link.

You can also show semantic directionality in the link label of these relationship links using the *label direction* property of the link. Semantic direction may be independent of the link navigation direction, as shown in the following figure.



The semantic direction of the link label vs. the link direction

Here there are two possible relationships between *Company* and *Person*. As you can see, the label direction creates semantics *Company employs Person* in the first instance, and *Person services Company* in the second case.

For more information see Common Customizations: Association direction.

See also

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Introduction to modeling

Creating and editing members and properties

Adding and editing members

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

- Right-click on the class or interface
- Select New | Attribute or New | Operation
- The member is inserted after the last attribute or operation, respectively

If a class or interface already has attributes or operations, you can right-click on these directly to create the additional member using the existing member's speedmenu. The new member is inserted before the selected member.

You can also add members form the Explorer. Use the speedmenu of class or interface nodes.

To edit an attribute or an operation:

Use inplace editing or editing in the properties inspector.

Adding and editing properties

To add a property:

- Right-click on the class
- Select New | Property

Location of the newly created property in the class depends on whether Recognize Java Beans options is on or off. If this option is selected, the property adds to the properties' compartment, and to the list of properties in the *Beans* tab of the Inspector. The class displays as a bean icon. If this options is off, the property adds to the attributes' section, and its accessor methods add to the operations' section.

In the latter case, you have to take special care when editing or deleting properties. When a new element is created, in-place editor is immediately available. However, if you edit property type in place, the relevant types in the accessor methods will not be synchronized. Same happens when a property is deleted: accessor methods stay in place and should be deleted individually.

Thus the only safe way to edit properties is to use the Choose Pattern dialog. This is how it's done:

- Right-click on the property to be edited
- Select Choose Pattern
- In the pattern treeview select *Property* pattern, make all the necessary changes and click *Finish*.

When this technique is applied, the changes propagate to all components of a property.

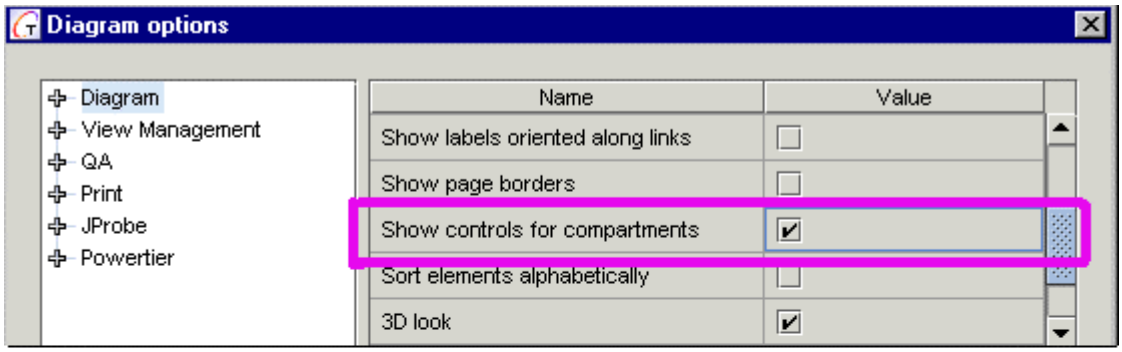
Rearranging the order of Attributes and Operations

Use drag and drop to reorder members within a class icon. The members are reordered in source code simultaneously.

Dropping one member on another member name positions the dropped member before the target member. Dropping a member on the class name moves it to the last position in the attribute or operation list respectively.

Compartment controls

You can optionally show an expansion/contraction control in the Attributes and Operations compartments of Class nodes. This is handy if you have large classes whose content you don't need to see all of the time. You can set this option at any configuration level... default, project, or diagram. On the Main menu choose Options | [scope], then select the Diagram page of the dialog and check Show controls for compartments.



Unselected

Selected, collapsed

Selected, expanded

You can use  icon on the main toolbar to control this behavior.
Class node in various selection states when compartment controls are enabled.

Sequence and Collaboration diagrams

Sequence and Collaboration diagrams, sometimes called collectively *interaction diagrams,* are two of several diagrams you can use to model the dynamic aspects of a system or subsystem. Both depict interactions consisting of a set of objects, their relationships, and messages exchanged among them.

Collaboration diagrams emphasize the *structural organization* of objects, while Sequence diagrams emphasize the *time ordering of messages*. Collaboration diagrams are essentially graphs; Sequence diagrams are essentially tables with different objects and messages depicted across the X axis and increasing time down the Y axis. However, the two diagrams are *semantically* equivalent: one type can convert to the other type with no loss of information. Together enables you to do this conversion with a simple mouse click.

Though semantically equivalent, the two diagrams do not necessarily **show** the same information. For example, Collaboration diagrams explicitly show how objects are linked, while in Sequence diagrams the links are implied. Message return values show in Sequence diagrams but not in Collaboration diagrams.

This feature only works in products that support Java, in projects where Java is the target programming language.

Creating and drawing Collaboration and Sequence diagrams

To create a Sequence diagram, invoke Generate Sequence Diagram Expert on the element speedmenu.

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. Choose Generate Sequence Diagram from the Operations' speedmenu to display the Generate Sequence Diagram Expert. View a list of the packages and classes involved in the operation for which you are generating a Sequence diagram.
4. In the Package/class list, select the packages and classes you want to display in the generated diagram. All packages and classes are selected by default. However, some Java things might not be relevant... `java.lang.integer`, for example. You can increase the meaningfulness of the generated diagram by removing anything that doesn't really help explain the sequence of operations.
5. In the Package/class list, for those elements you decide to show in the diagram, check whether or not to show implementation detail in the generated diagram.
6. Click *OK* to generate the diagram and open it in the Diagram pane.

Together generates a new Sequence diagram in the same package as the source class, with the same name as the source operation, and opens it in the Diagram pane.

Using Options dialog to control Sequence diagrams' generation

You may use Options dialog to check options for the generated diagrams or control their behavior. To do this, select *Diagram Options* on the diagram's speedmenu. This command displays *View Management, Diagrams* and *Print* tabs of the Options dialog on the project level. Alternatively, invoke the Options dialog from the main menu.

It is possible to generate sequence diagrams for several methods. You can display sequence diagram for each method in a separate tab, or to show sequence of calls for all selected methods in a single sequence diagram. To control this behavior, use the options *Create multiple diagrams* and *Show multiple diagrams* of the Sequence diagram node in *View Management* tab of the Options dialog.

When the option *Create multiple diagrams* is checked, a separate diagram is generated for each selected method. Otherwise, sequence of calls for all methods is generated on a single diagram.
When the option *Show multiple diagrams* is checked, all generated diagrams should show up automatically after generation completed.
It is also possible to control nesting level. *Depth of call nesting* option allows to change nesting value according to the required degree of complexity.

Converting to a different interaction diagram

As mentioned in the introduction above, Collaboration and Sequence diagrams are just different ways of viewing the same information. *Together* enables you to view either type of interaction diagram as the other type. However, when you create a new diagram, you must specify that it is either a Sequence diagram or a Collaboration diagram, and Together then tracks it as such. The diagram displays in the Explorer as the type of origin, and opens in that view. That is to say, if you create a Sequence diagram, it will always show in the Explorer, and open in the Diagram pane as a Sequence diagram. But you can view it as a Collaboration diagram.

To view an interaction diagram as the other type:

1. Right-click on the diagram background.
2. If the diagram is a Sequence diagram, choose **Show as Collaboration** from the diagram speedmenu. If viewing a Collaboration diagram, the menu command is **Show as Sequence.**
3. Repeat this process to switch back and forth.

Note: You can also do this switch from the speedmenus of interaction diagrams in the Explorer.

Key elements and properties

Content

Collaboration and Sequence diagrams both typically contain:

	**Objects**: to encapsulate states and behavior
	**Actors**: to send and receive messages
	**Statement Blocks**: to generate statements for messages or self messages
	Relationship links
	**Messages:** to convey information from one object to one or more other objects
	**Messages with delivery time**:
	**Self Messages**
	**Aggregation**
	Notes and note links

Key elements

Object

An object role in a Sequence diagram is shown as a vertical dashed line called the “lifeline”. The lifeline represents the existence of the object at a particular time. If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at an appropriate point; otherwise it goes from top to bottom of the diagram. An object symbol is drawn at the head of the lifeline.

Message

A *message* is a communication between objects that conveys information with the expectation that action will ensue. Receipt of a message is a kind of event. A message is shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another object. The message may also be drawn from and to the same object, representing a message from an object to itself. The arrow is labeled with the name of the message (operation or signal) and its argument values. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are useful on the diagrams for identifying concurrent threads of control.

> **Note:** The appearance of messages is defined in the View Management page of the Options dialog, under the Sequence diagram node. In the field "Maximum message labels length" you can specify the number of pixels allocated for the message label. If the label is too long, it can be either truncated and displayed with trailing dots, or wrapped into multiple lines, depending on the flag "Wrap message labels text".

Actor

The Actor enables you to create sequence diagrams that model how business workers interact with and handle business objects while performing the workflow of a realization of a business use-case.

The Actor in a Sequence diagram displays a lifeline and can exchange messages with other Actors and/or objects.

Key Object properties

Name

Every object has a name to distinguish it from other objects. Name is a text string. You need only enter the *simple name* for an Object. The qualified name is used when you specify the Object's classifier (see *instantiates).*

Instantiates

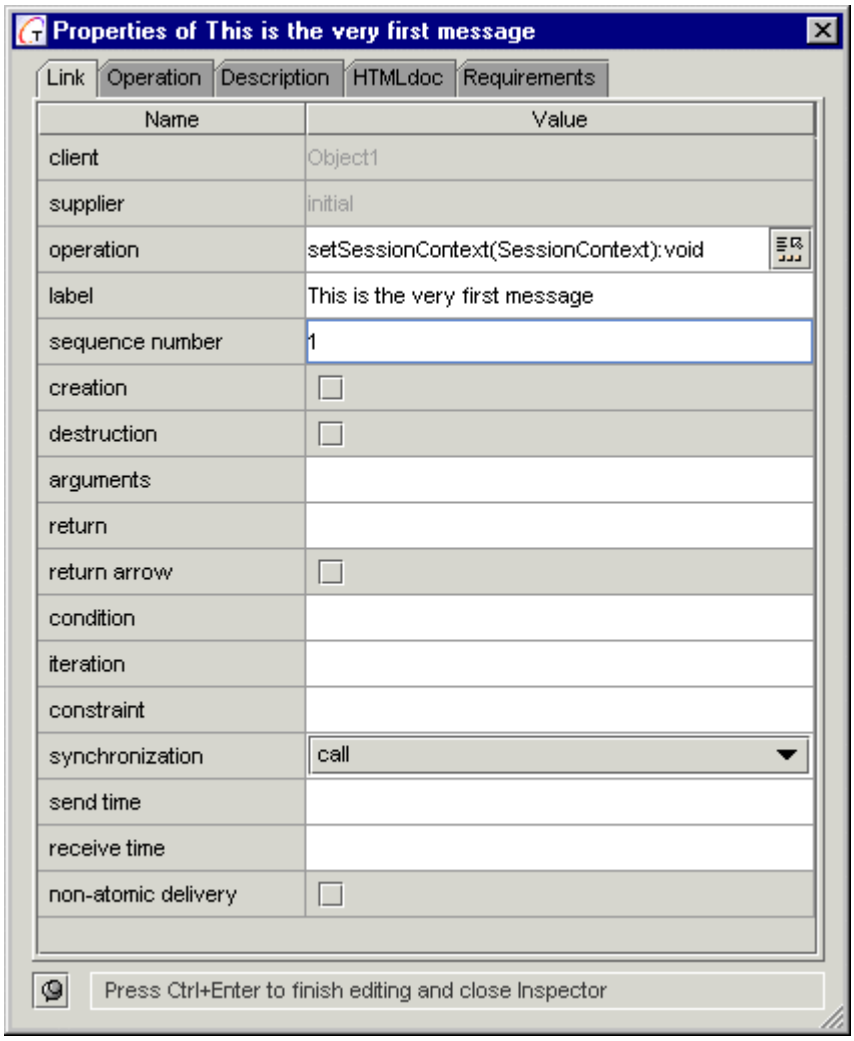
Specifies the Object's classifier. You can either enter the value yourself, or pick the class from a dialog that lists all the resources available to the project.

When you apply this property, the Object displays its fully qualified name. The properties Inspector displays the Class tab in which you can access the properties of the Object's class.

Tip: You can also specify the classifier by selecting the Object in the diagram and calling *Choose Class* from its speedmenu.

Key Message properties

Properties of Messages, and the labels they create on the diagram, convey a great deal of information about the dynamics of the interaction, especially in Sequence diagrams. You won't necessarily set a value for every property: for example you wouldn't set Condition unless modeling some elementary branching.



Operation

If the Message is sent by an operation, you can identify it here. Enter a value or pick the operation from an explorer dialog listing your project resources. You can specify *arguments* and/or *return* value, if any, in those properties.

Label

Optional arbitrary identifier. For example, if the message is one that creates another Object, you could label it "create".

Sequence number

Time-ordered sequence number. Value is automatically incremented as you draw Message links. You could modify this value for two or more Messages to reorder their time sequence.

Creation

Check if the Message is the one that creates the Object to which it is sent.

Destruction

Check if the Message is the one that destroys the Object to which it is sent.

Condition

Normally a Boolean expression (e.g., error > 0).

Iteration

Enter a value if the Message is iterative or sent on a given iteration.

Synchronization

Specify how or whether the target Object waits for some result. Useful when modeling multiple threads of flow control.

Working with Sequence and Collaboration diagrams

How to create messages to self
How to create Message links that call operation
How to reorder message links

Creating a Message-to-self

To create a message from an Object back to itself:

1. Click on the Self Message button on the diagram's toolbar
2. Click on the Object's life line at the point where you want the Message to appear

Creating a Message link that calls an operation

For such message links there is additional tab "Operation" in the Inspector. If the Editor pane is visible, it displays the source code of this operation.

To create a Message link that calls an operation:

1. Create a Message link between two objects. Both objects must have their *instantiates* property set to point to a class.
2. Select the message link.
3. In the Inspector, go to the *operation* field and launch the picker dialog using the field's browse button to display the Choose Operation Name dialog. The dialog displays the operations of the recipient object's class.
4. Select the operation and click OK. This renames the Message link to the operation's name.

If you now select this Message link, the Inspector displays two tabs. The tabs are MessageLink (Message link properties) and Operation. The Operation tab displays the operation's properties.

Reordering Message links

Select and drag Message links up and down the Object lifeline to reorder them. Reordering automatically updates the Message link numbers.

Sometimes you may wish to reorder message links keeping their sequential order and freeing the space between for new links. To do this, select a Message link line, press CTRL, and drag it . This shifts all succeeding links. If you select a number of Message links (pressing Ctrl key), then those selected are moved keeping their increments.

Tips and Tricks

Tips and Tricks for Sequence diagrams

If you've not previously drawn Sequence diagrams using *Together,* a few pointers may be in order.

- Objects display with a default lifeline when placed on the diagram. Their tops align vertically. If you draw a Message to an Object and then check the **creation** property of the Message, the created Object will move downward to show that it exists at a point forward in time from its creator.

- You can lengthen or shorten object lifelines as needed by dragging the horizontal line of the bottommost Message link upward or downward. You can arrange the position of other intervening Messages this way also.

- You can reorder the Sequence diagram, maintaining any Message links already created between the Objects. Select any Object and drag it to the desired position. Such change is performed across the X axis of Objects- you cannot move Objects vertically along the Y axis except as described in the first point above.

- The bold **X** indicating destruction of a created Object is rendered automatically: draw a Message to the Object and check the Message's **destruction** property (*Link* tab of the Inspector).

- The focus controls of Objects that show periods of time an object performs some action are also rendered automatically: just draw Message links to create them.

- You can nest Messages by originating Message links from a focus control.

- Use properties of Message links for specifying such things as:

    linked operation
    arguments
    return value
    simple branching condition
    iteration
    time and other constraints

Tips and Tricks for 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.

As you add Messages, they display in time-ordered sequence from top to bottom of the messages list. You can select Messages and edit their properties in the message properties Inspector just as you can do in a Sequence diagram.

The Collaboration diagram adds the capability of showing relationships between Objects. In addition to the default link, you can add links to show **Association** and **Aggregation** relationships. These links do not display if you view the diagram as a Sequence diagram.

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Approaches to modeling

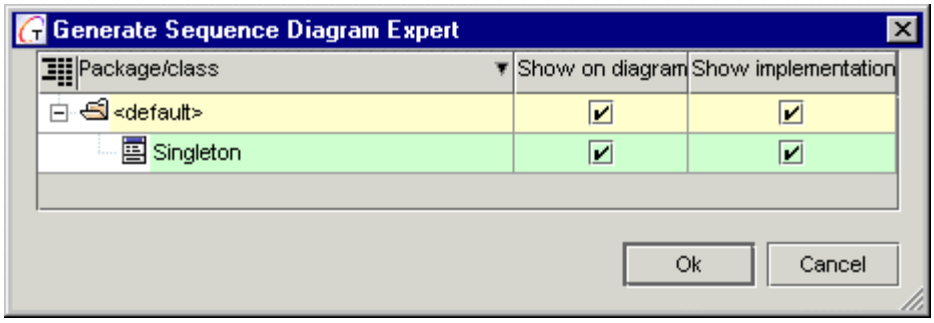
Generating Sequence Diagrams

Suppose that you want to understand more clearly the workings of a class's operations, and communicate this visually in your model. Normally you would have to study the relevant code and build Sequence diagrams for this purpose. Together can handle this chore for you by generating a Sequence diagram from any operation you select in the visual model of a class. This functionality can be extremely useful when you are implementing classes from patterns... especially third-party patterns with which you are unfamiliar.

> **IMPORTANT:** For Sequence diagram automation to function correctly, the Together installation directory name must not contain spaces.

Generating Sequence Diagrams Using the Expert

The *Generate Sequence Diagram* module enables to generate Sequence diagrams from class operations for visual modeling and source code analysis. To invoke this command, select the desired operation on the Class diagram, and right click to display the speedmenu. Choosing *Generate Sequence Diagram* option brings in the Expert dialog:



The two options define the outlook of the resulting Sequence diagram. Check *Show on diagram* flag to show the object of the corresponding class in the generated Sequence diagram. *Show implementation* allows to hide/show internal calls for any classes.

Using Sequence automation to analyze patterns

This section provides a short tutorial exercise that can help you learn how to use Sequence automation to analyze patterns.

For the exercise, you need a Together product that supports Sequence diagrams and Sequence diagram automation. (To learn where to get product feature information , see Where to Get Help.)

Create an example project for training purposes:

1. Launch Together.
2. From the main menu, choose File | New Project to invoke the New Project dialog.
3. In the New Project dialog enter a project name *patterns*. Use browse button to select or create a directory for your project (under *myprojects"* for example), return to the New Project dialog and click OK to create the project.

Analysis of the Singleton pattern

Now use the Patterns feature to create an instance of the GoF *Singleton* pattern in the project just created.

To create the Singleton pattern:

1. Create new package and name it *singleton*.
2. On the "singleton" package icon invoke the speedmenu and select *Open in New Tab* to open the default package diagram for the new package. This diagram is empty.
3. On the diagram toolbar, click *Class by Pattern*, then click again on the diagram workspace. The "Choose Pattern" dialog appear.
4. In the patterns tree, select the pattern GoF - Singleton and click the "Finish" button. A new class *Singleton* is created in the package and displays in the diagram.

At this point you have an instance of the Singleton pattern in the project. It is displayed visually in the default diagram and its source code is loaded in the Editor. You can see the attributes and operations of the class in the diagram.

Suppose now that you want to understand more clearly, and then model the workings of the getInstance() method. Normally you would have to study the code and manually construct a sequence diagram for this purpose. With Together, you can simply generate this diagram.

To generate a Sequence diagram on getInstance() method:

1. In the Singleton class icon, select the method *getInstance*:*Singleton*, invoke speedmenu and choose *Generate Sequence Diagram*. The Generate Sequence Diagram Expert dialog appears.
2. Click OK. A Sequence diagram named "Singleton.getInstance" is created and opened in a new tab in the Diagram pane.
3. Select the activation rectangle of the message labeled */getInstance()*:*Singleton*... that is, the destination of this message.
4. On the main toolbar click the *Editor Pane* icon to display the Editor pane, which is hidden by default for Sequence diagrams, or select View | Editor Pane from the main menu. The insertion cursor will be positioned on the method declaration for the getInstance() method.

Analysis of the Composite pattern

1. Left click on the diagram tab named "<default>". The <default> package diagram will be opened in the Diagram Pane.
2. Create new package and rename it to "composite".
3. On the "composite" package icon invoke speedmenu and select "Open in New Tab".
4. Press the diagram toolbar button "Class by Pattern" and click on the diagram workspace. The "Choose Pattern" dialog displays.
5. In the patterns tree select pattern GoF|Composite and click "Finish". Two classes will be created in the package.
6. In the "Composite" class icon select method "sampleOperation:void", invoke speedmenu and select menu item "Generate Sequence Diagram". The "Generate Sequence Diagram Expert" dialog displays.
7. Press "OK" button. Sequence diagram named "Composite.sampleOperation" will be created and opened in a new diagram tab.

The resulting sequence diagram shows some things which are not relevant: two java.util classes (Vector, Enumeration), and message to self in the "initial" object.

To refine the diagram and show only collaboration between the pattern classes the GSD options should be changed.

1. In the main toolbar press "Undo" button. The "Composite.sampleOperation" diagram tab will disappear, "composite" package diagram will become current diagram.
2. On the main toolbar press "Diagram view management..." button. The "Diagram options" dialog appears.
3. Uncheck option Sequence diagram - *Include messages to self* and press OK.
4. In the "Composite" class icon select method "sampleOperation:void", invoke speedmenu and select menu item "Generate Sequence Diagram". The "Generate Sequence Diagram Expert" dialog displays. This dialog contains a tree-table showing the classes that take part in the given collaboration. The tree-table consists of packages on the first level and classes on the second level. This table enables you to exclude from the generated Sequence diagram either the implementation of some classes, or the classes themselves.
5. In the tree-table select package java.util and uncheck checkbox in the column "Show on diagram".
6. Click the OK button. A Sequence diagram named "Composite.sampleOperation" is created and opened in a new diagram tab.

Generating implementation source code

This section demonstrates how to draw and edit a Sequence diagram that generates source code.

Creating project and class

1. Create a new project. In the New Project dialog enter project name and press OK. <default> package diagram displays.
2. Create new class *MyApplication*.
3. On the MyApplication class speedmenu select *Choose pattern* to display the Choose Pattern dialog.
4. In the patterns tree select *Main class* pattern.
5. Press the Finish button. New method *main()* is created.

Generating sequence diagram from a class

1. Select Generate Sequence Diagram from the *main* method speedmenu. The Generate Sequence Diagram Expert dialog displays.
2. Press OK button. A Sequence diagram *MyApplication.main* opens in a new diagram tab.
3. On the main toolbar press the Diagram View Management  button, or select Options | diagram from the main menu and click on View Management tab. The View Management page of the Options dialog displays.
4. Check option Sequence diagram|Show message numbers and press OK button.

Creating source-generating elements in the sequence diagram

1. On the diagram toolbar press the Statement Block button  and click on the activation rectangle of the message #1. Choose statement type dialog displays.
2. Check *for* radio-button and click OK. The statement *for* and its shaded rectangle show up on the diagram.
3. Invoke in-place editor and enter: `int i = 0; i < 4; i++`. The label reads: `for(int i = 0; i < 4; i++)`
4. Press Object button  on the diagram toolbar and create a new object with the name *frame*.
5. Select *Choose Class | More* on the object's speedmenu to display Choose Object's Class dialog.
6. In the Search/Classpath, select class *javax.swing.JFrame* and click OK. The name of selected class displays in the object's icon.

Creating message sends

1. Draw a message link from the statement block *for(int i = 0; i < 4; i++)* to the lifeline of the *frame* object. Message #1.1.1 is created. All the other messages should have same source and destination.
2. Select *message #1.1.1.* and choose *Type | Creation* on its speedmenu. The message will change visually to the *creation* type (the link now points to the *frame* object, which means that the object is being created).
3. Draw a new message. Its label is *message #1.1.2*. Select *Choose Operation* on the speedmenu. From this dialog choose *setDefaultCloseOperation(int):void*. The message label becomes: *1.1.2:setDefaultCloseOperation(int):void*.
4. Invoke in-place editor and enter `(JFrame.EXIT_ON_CLOSE)` after the message number and column. The message label becomes *1.1.2: {setDefaultCloseOperation(:int):void}(JFrame.EXIT_ON_CLOSE)* .
5. Draw message #1.1.3. Using Choose Operation dialog, select *setSize(int,int):void* method. Invoke the in-place editor and enter: `(600,400)` after the message number and column. The message label becomes *1.1.3: setSize(600,400):void*
6. Draw message #1.1.4. Using Choose Operation dialog, select setLocation(int,int):void method. Invoke in-place editor and enter `(50*i,50*i)` after the message number and column. The message label becomes: *1.1.4: {setLocation(50*i,50*i):void}*.
7. Draw message #1.1.5, invoke its speedmenu and select Choose Operation | More to display the Choose Operation Name dialog. Choose *show()* method from the list. The message becomes *1.1.5:show():void*

Tip: If you cannot find the desired name in the list, expand *All Operations* node and search through the loaded list of methods.

Generating implementation code

1. Select the activation rectangle of the message #1, invoke speedmenu and select Generate Implementation. to create implementation of the *main()* method. This implementation code displays in the Editor. Message labels for which implementation code has been generated, display **bold** style on the diagram. Message pane opens. If for some reasons code generation fails for certain messages, they remain normal style on the diagram, and appropriate messages show up in the Message pane.

2. Select any message on the diagram, and observe that the Editor scrolls to the point of appropriate method invocation.

3. In the Editor, add import statement for *javax.swing*.

4. The main menu | Tools and the diagram speedmenu allow to Run the generated code and Make the project. In the former case you will observe a cascade of four frames. In the latter case the following messages show up in the Builder tab:

```
*** Make started
*** Make completed
*** Output directory:
D:\Together5\out\classes\MyApp
```

See also

Sequence and Collaboration diagrams

Statechart Diagrams

Statechart diagrams enable you to model UML state machines. State machines are one of several ways to model the dynamics of a system or subsystem. They generally model the behavior of a single object as it goes through different states in response to events that occur during its lifetime, including its response to events. The state of an object is some situation during its life where it meets some condition(s), does something, or waits for an event.

Statechart diagrams emphasize the possible states of an object and the transitions between states. You can use them to model the lifetime of an instance of a class, an instance of a use case, or even an instance of a system. During that period, the object of interest can be exposed to different kinds of events to which it responds with some action, and the result of which is a change in the object's state.

Tip: To focus on the actions and activities of objects, use Activity diagrams.

Content

Statechart diagrams generally contain:

	**States**: to represent the states of an object during its lifetime
	**Start state**
	**End state**
	**History state**
	**Transitions:** directed link-lines representing the transitions between object states
	Fork/Join (or SyncBars): to represent multiple transition sources or targets
	horizontal Fork/Join
	vertical Fork/Join
	**Objects**: encapsulating states and events
	Notes and note links

Key elements and properties

Key elements

Transition

Most of the information for the Statechart diagram is communicated through the various properties of Transitions. With these you can specify *event name* and *event arguments, guard conditions, action expressions,* and specify general and send/receive time constraints.

History State

History states enable modeling of objects that remember the last active substate of a composite state before leaving it. The *deep* property of a History State supports the UML deep history concept for multi-level nested substates: when checked it signifies that the object remembers the last active substate down to the innermost nested level. Only one history element is allowed for a state.

Key Object properties

Name

Every object has a name to distinguish it from other objects. Name is a text string.

Instantiates

Specifies the Object's classifier. You can either enter the value yourself, or pick the class from a dialog that lists all the resources available to the project.

When you apply this property, the Object displays its fully qualified name. The properties Inspector displays the Class tab in which you can access the properties of the Object's class.

> **Tip:** You can also specify the classifier by selecting the Object in the diagram and calling *Choose Class* from its speedmenu.

Key Transition properties

Event

Specify the event name in the *event name* property of the Transition. Optionally specify any arguments in the *event arguments* property.

Guard condition

Enter an appropriate expression in the *guard condition* property of the Transition.

Action

Enter an appropriate value in the *action expression* property of the Transition.

Working with Statechart diagrams

How to draw self-transitions
How to create internal transitions
How to specify entry/exit Actions
How to create nested substates
How to draw multiple Transition sources or targets

Drawing a self-transition

Self-transition means that the flow leaves the State, dispatching any exit Action(s), then re-enters the State, dispatching any entry Action(s).

- Select the desired State on the diagram.
- Drag a Transition link from the selected state and drop it on the diagram background.
- In the Choose Destination dialog, navigate to the State where you began the link, select it, and click OK.

Tip: Another way is to draw a Transition between two States, and then drag the opposite end of the link line back to the desired State.

Creating internal transitions

Internal transition is a shorthand for handling events without leaving a State and dispatching it's exit/entry Actions.

- Select the desired State on the diagram
- From its speedmenu choose New Internal Transition.

You can immediately edit the event name in place. For additional information see Entry/Exit Actions below.

Specifying entry/exit Actions for a State

These are Actions executed upon entering or leaving a State, respectively. You create Entry and Exit Actions in *Together* Statechart diagrams as stereotyped internal transitions.

Create an internal transition in the desired State.

When you edit the name in place, use the following syntax: *stereotype/actionName[argument]*

Example: `exit/setState(idle)`

Optionally, you can create the internal transition and set the *event name, event arguments* and *action expression* properties in the properties Inspector (speedmenu | Properties).

Creating nested substates

You can create a *composite State* by nesting one or more levels of States (i.e. *substates*) within one State. You can also place Start/End states and History states inside a State, and draw Transitions among the contained substates.

The easiest way to create a nested substate is to place a State node on the diagram background, drag it on top of another State, and drop it.

Showing multiple Transition sources or targets

A Transition may have multiple sources, meaning it is a join from several concurrent states, or it may have multiple targets, meaning it is a fork to several concurrent States.

You can show multiple transitions with either a vertical or horizontal orientation in your Statechart diagrams. The Statechart diagram toolbar provides separate Fork/Join buttons each orientation. The two orientations are semantically identical.

To create multiple Transitions:

1. Identify the States involved. If necessary, place all the States on the diagram first, and lay them out as desired.
2. Place either a horizontal or vertical Fork/Join on the diagram. Resize it as needed.
3. If depicting multiple *sources*, draw Transitions from each of the source States to the Fork/Join;
4. If depicting multiple *targets*, draw a Transition from the source State to the Fork/Join; then draw Transitions from the Fork/Join to each of the target States.

Tips and Tricks

-You may want to resize the main State larger. You can essentially draw another Statechart diagram inside it, complete with Start/End/History states and transitions of all kinds, to create, in effect, a substate diagram

- You can nest multiple levels of substates inside one State. For really complex substate modeling, however, you may find it easier to create different diagrams, model each of the substate levels individually, and hyperlink the diagrams sequentially.

- You can easily reuse any elements that you have already created in other Statechart diagrams. Invoke *Add Shortcut* command on the diagram's speedmenu and navigate in the dialog's Explorer to the existing Statechart diagram and select one or a number of its elements, States, Histories, and/or SyncBars (Fork/Joins).

- Using the *Clone* command on the context menu of the diagram's Explorer node, you can quickly create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.

- The above can be useful if modeling complex composite states or substates.

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Approaches to modeling

Activity Diagrams

Activity diagrams provide one of the possible ways to model system dynamics. An Activity diagram is basically a flowchart that describes the flow of control from one activity to the next. You can show sequential and/or concurrent steps of a process, model business workflows, model the flow control of an operation, or the flow of an object as it passes though different states at different points in a process. Unlike interaction diagrams (Sequence, Collaboration) that emphasize the flow of control between *objects*, Activity diagrams emphasize the flow of control between *activities*.

An *activity* can be described as "an ongoing, non atomic execution within a state machine" *, the ultimate result being some action that affects the state of the system or returns some value.

Content

The content of an Activity diagram can vary depending upon what kind of control flow you are modeling and what level of detail you choose to provide. Activity diagrams most commonly contain:

	**Activities**: to represent activity states and action states
	**Transitions:** to show events and actions
	**Objects**: encapsulating states and events
	**Decisions**: for branching flows
	**States**: to represent the states of an object during its lifetime
	**Start state**
	**End state**
	Fork/Join (or SyncBars): for concurrent flows
	horizontal Fork/Join
	vertical Fork/Join
	**Swimlanes**: to show responsibility for activities
	**Signal Sending**: to represent sending transition
	**Signal Receipt**: to represent receipt transition
	**Object flow**: to represent a control flow for objects
	Notes and note links

Key elements and properties

Activity

Activities are atomic actions that cannot be further decomposed. For example, in an airline reservation system, "Request a reservation", the activity representing a customer's request, cannot be broken down further.

Use Activity for both action states and activity states-- graphically there is no distinction. Use Notes with Note Links if you need to differentiate.

Swimlanes

Swimlanes enable you to partition activities into groups based on responsibility for carrying out the activities. You could have swimlanes for different objects when modeling operation workflows, or different business entities when modeling business process workflows. For example, in an airline reservation context, you might have separate swimlanes for Customer and Ticket Agent.

Swimlanes normally have a vertical orientation. The flow of activity runs from top to bottom. Transitions between activities will cross swimlanes left-to-right or vice versa as the responsibility for carrying out activities changes.

Transitions

Transition links show the path of the flow control from one action or activity state to the next. Together provides two types of transition links:

Transition: used to show flow control between Activities or States.
Object Flow: used to show flow control when modeling flow control of objects.

Several properties of Transition links can be used to convey important information about what's happening in the diagram.

Decisions

Decisions show branching and specify alternate flows based on the evaluation of some Boolean expression. The Decision icon itself has only one property *Name* . Relevant information about the decision should be shown in the properties of the Transitions that represent different branches. For example, specify the Boolean expression for each branch in the *guard condition* property of the Transition link for each branch.

In an airline reservation context, the decision you want to show might be "is space available on this flight". So you could name the Decision "isSpaceAvailable", one *guard condition* property of the branch being "Available" and the other "Not Available".

Tip: If you want Together to show labels inside decisions, go to %Together_Home%\config\diagram.config file and change the value of the option `option.show_decision_label` to *true* (it is set to *false* by default).

Note that if your labels are long (for example, when the project from an older version of Together is loaded), it may affect the diagram layout.

Signals

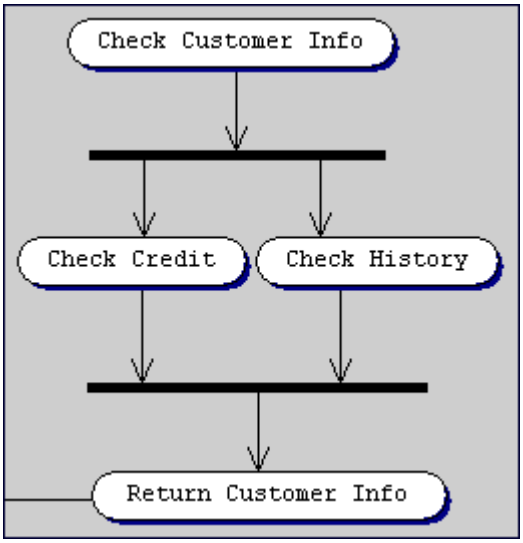
Two icons Signal Receipt and Signal Sending are provided to enable explicit symbols for certain kinds of information that can be specified on transitions: *signal receipt* and *signal sending*. You will probably not use these in Activity diagrams that model business process workflows... they are rather more useful in the context of object control flows.

Forks and Joins

These controls are used to model concurrent flows. For example, an activity "Check Customer Info" might check a customer's purchase history and current credit card information concurrently.

There are two Fork/Join controls available on the Activity diagram toolbar: one for showing concurrency with a vertical orientation  and the other for horizontal orientation  . This functionality is also called SyncBar, and in fact the default name for both horizontal and vertical variants of this element is "SyncBar*n*" (where *n* is an incrementing integer).

When representing a "fork" where flow becomes concurrent, the SyncBar typically has one Transition coming in, and two or more going out. When representing "join" where flow is no longer concurrent, the SyncBar typically has two or more Transitions coming in, and one going out.



Using SyncBars for fork and join in Activity diagram

Key properties

Event

Specify an event name that triggers a transition in the *event name* property of the Transition. For example, a Transition between the Start State and an activity *Check Flights* might be an event *TicketAgentRequest*. When modeling an object work flow, you might specify an actual system event and use the *event arguments* property .

Action

Specify an action that causes the transition. This could be a user action in business process, or a computation that results in a state change in the process.

Guard condition

Two icons Signal Receipt  and Signal Sending  are provided to enable explicit symbols for certain kinds of information that can be specified on transitions: *signal receipt* and *signal sending*. You will probably not use these in Activity diagrams that model business process workflows... they are rather more useful in the context of object control flows.

Tips and Tricks

Non-trivial systems will probably require many activity diagrams to capture the dynamics of a workflow or operation. Use the Clone feature to create new diagrams with identical content in the same package.

Using *Add Shortcut* command on the diagram's speedmenu, you can reuse any already created 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.

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
State diagrams

Component Diagrams

Component diagrams are the second way to shows the physical architecture of a computer-based system. Together with deployment diagrams, they are geared expressly toward computer systems.

A component - as the main element in a such type of diagrams - is used to package other logical elements, and represents things that participate in the execution of a system. Components also use the services of another component via one of its interfaces. Usually, components are used to visualize logical packages of source code (work product components), binary code (deployment components) or executable files (executions components).

A component diagram usually shows components, interfaces and relationships among them.

Content

Component diagrams usually contain:

	Subsystems
	Component
	Interfaces
	Notes and note links
	Relationship links
	Dependencies
	Supports

Component diagrams in *Together* use two kinds of relationships: dependency and realization (supports).

To group one or more logical elements of the model, component diagrams also can use packages or subsystems.

Key elements and properties

Components

Components typically represent a package of other logical elements and can be grouped themselves. They also can realize its own interfaces or use interfaces of another component.

Remember, that each component represents just one aspect, one view of the system.

Interfaces

See Deployment Diagram for more information about interfaces.

Links

Dependency: indicates that one component uses the services of another component
Support: represents that a component might realize an interface

Tips and Tricks

Using the *Clone* command on the speedmenu 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.

Using *Add Shortcut* command on the diagram's speedmenu, you can reuse any already created elements in other state diagrams. Note: Elements imported this way are independent copies of the existing ones.

You can represent realization in two ways: using the *support* stereotype and the dashed line (canonical form), and using the solid line (lollipop notation)

You can hide subcomponents on a component diagram. Go to Options | Default | View Management, and enter the following for one of the "User Defined" fields under 'Show':

Name: "Nested components" *Expression:* `hasPropertyValue("$shapeType", "Component" ) && !(getContainingNode() == null)`

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams

Deployment Diagrams

Deployment diagram provides one of the two ways to model the physical aspects of a system. It is a graph of nodes connected by communication associations and it shows the physical architecture of the hardware and software of the system.

Content

Deployment diagrams usually contain:

Nodes, representing a processing resource

Relationship links:

Association
Dependency
Realization (supports)
Aggregation

Deployment diagrams in *Together* can also show:

Components, that live on nodes and they may provide realization of interfaces

Interfaces

Objects, that may live in processes that live in components

Key elements and properties

Nodes

A Node is a run-time physical object that can include not only computing devices but also human resources or mechanical processing resources. Basically, nodes are things that execute something and they represent locations where other elements are deployed.

Components

Though components are a lot like nodes, they are things that are executed by nodes. Components typically represent a package of other logical elements and they can be deployed on one or more nodes.

Interfaces

Interfaces are used to specify a service of a component and they may be imported or exported by them. Therefore an interface specifies a contract that a component (or a class) must carry out.

Together uses the most common way to show a relationship between a component and its interfaces - through an elided (hidden) realization relationship.

Links

Association: usually they represent a communication between two elements. Use a stereotype to indicate the nature of the communication.
Dependency: represents a connection between two components (sometimes through interfaces)
Realization (supports): represents that a component might realize an interface
Aggregation: represents a connection between an aggregate (whole) and a component (part)

Tips and Tricks

Using the *Clone* command on the speedmenu 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.

Using *Add Shortcut* command on the diagram's speedmenu, you can reuse any already created elements in other state diagrams. Note: Elements imported this way are independent copies of the existing ones.

Organize components by specifying the relationships among them.

Objects and components can migrate from one component instance to another component instance, respectively from one node instance to another node instance. In this case, the object (component) will be on its component (node) only for a part of entire time. To show that, use the dependency relationship with a *becomes* stereotype.

A component may reside on nodes. You can represent this in two ways: using the *support* stereotype and the dashed arrows, and by graphically nesting the component symbol within the node symbol.

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams

Together Diagrams

Business Process diagrams

Business Process diagrams are a feature of Together ControlCenter. Though it is not yet specified as a diagram type in the UML, Together provides the *Business Process (BP) diagram* to enable you to apply some of the UML extensions for business modeling. Business object models model the structure, processes, use cases, and relationships of a business as part of an overall business object model. The business object model describes the realization of business use cases, providing an abstraction of how business workers and business entities are related and how they must work together to actually run the business.

A business object model describes the use cases of a business from the internal viewpoint of business workers . It defines the static and dynamic aspects of *relationships* between the workers and the classes and objects they use to produce the expected results. In aggregate, the objects of the model's classes should be capable of performing all the use cases of the business.

Using BP diagrams, you can model the static aspects of a business object model, especially business use cases. To one or more BP diagrams, you may add Sequence and Activity diagrams to show the dynamic aspects of the business object model and thus achieve a complete business model.

Content

Business Process diagrams contain:

	Business Actors
	Business Use cases
	Business Workers
	Business Entities
	Relationship links
	Association/Communicates
	Extends
	Includes
	Generalization
	Aggregation
	Subscription

	System boundary
	Notes and note links

Notation

The current UML specification (v. 1.3) does not specify any graphical variation for elements such as Use Case and Actor when used in a business object modeling context. Together BP diagrams use the standard UML graphical notation for these elements, and provides compliant notation for UML extensions such as business worker and business entity.

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams

Robustness Analysis Diagram

Robustness Analysis diagrams, part of the Objectory process, serve a number of useful purposes in a use case driven modeling effort. Robustness analysis involves working through the text of a use case, and taking a preliminary peek at how you might design some software to implement a given use case, using the objects you've discovered up to this point. Obviously one of the main purposes of this activity is to discover when you don't have all the objects you need, and then add them to your class diagram. While you're exploring possible designs, its useful to classify objects into one of the three stereotypes (boundary, control, entity) provided.

The robustness model provides a bridge between the "analysis level" view provided by the text of the use case and the "detailed design" view shown on a sequence diagram. Since it's very difficult to jump from analysis directly to detailed design, it's hard to do modeling successfully without this step.

Drawing the robustness diagram for a use case provides a visual completeness check that shows that the entire use case has been accounted for. It also serves to enforce the "active voice" style that is most effective when writing use cases.

It is beyond the scope of this guide to provide detailed review of the robustness analysis. The most comprehensive information on this issue can be found in "Use Case Driven Object Modeling with UML" by Doug Rosenberg with Kendall Scott.

Content

Robustness diagrams usually contain:

	Boundaries
	Entities
	Controllers
	Worker boundaries
	Worker controllers
	Actors
	Relationship links:
	Associations
	Robustness Associations
	Notes and note links

Key elements and properties

The robustness diagram symbols, such as Entity, Boundary, Worker Boundary, represent stereotyped views of classes. For each symbol on the robustness diagram, there is a corresponding class. There is a mechanism for converting a Controller and a Worker Controller to methods. Rosenberg's suggestion of using "invokes" in place of "includes" and "precedes" in place of "extends" is realized via specific Robustness Association.

Boundary

Boundary objects are the objects in the new system with which the actors will be interacting. These may be windows, screens, dialogs, and menus.

Entity

Entity objects refer to the database tables and files that store data, fetch data and perform computations that don't change frequently. These objects should be enough simple and generic, to provide the possibility of future reuse.

Controller

Control objects, or controllers, represent the functionality and system behavior of the use cases. Controllers may be converted into methods associated with the interface objects and entity objects.

Association

Links are represented by arrows. Unlike sequence diagrams, these arrows don't correspond to messages. They rather indicate logical associations.

The associations should follow certain rules:

1. Actors can talk only to boundary objects
2. Boundary objects can talk only to controllers and actors
3. Entity objects can talk only to controllers
4. Controllers can talk to both boundary objects and controllers, but not the actors.

See also

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Approaches to modeling

Entity Relationship diagrams

The Entity Relationship (ER) diagram is a high level data model that shows the major entities and relationships which support a general business area. The objective of the ER diagram is to provide a view of business information requirements sufficient to satisfy the need for broad planning for development of its information system.

Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects.

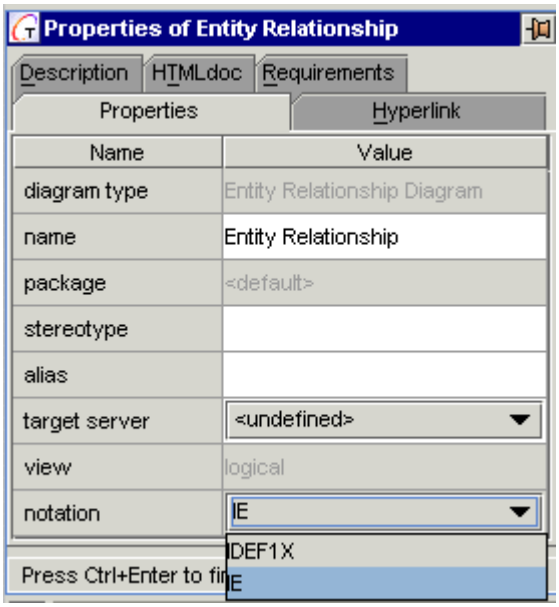
Visit **www.togethersoft.com** for more information about Together products that support ER diagrams.

Notation

The ER diagram is not specified in the UML. Thus, UML purists may not wish to include it in their modeling process. However, since many enterprise developers use ER diagrams (and asked for them in Together!), they are supported in Together ControlCenter.

There are two ways to represent data objects in ER diagram. Together allows a choice between the most popular notations IDEF1X and 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:



When notation is changed, the diagram automatically redisplays 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 ER diagrams. 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.

Contents

ER diagrams contain the following elements:

IDEF1X	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**
		Notes and note links

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.

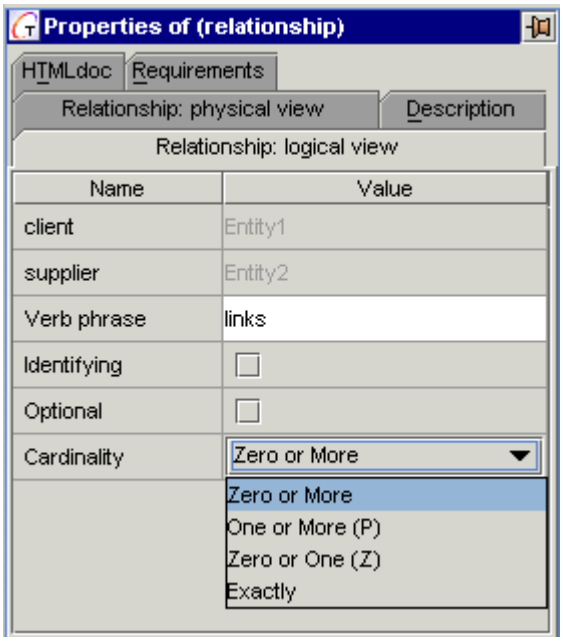
The physical view displays read-only fields *Physical type, Size and Digits*, which depend on the selected target server and specific driver.

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

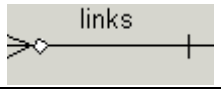
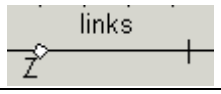
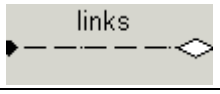
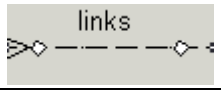
Relationship links are displayed according to the selected cardinality, with the names written above the line. It is advised to use verbs as the links' names.

You can create identifying or non-identifying relationships using the appropriate toolbar icons. Identifying relationships are displayed in solid line, and non-identifying relationships are in dotted line. Once created, a relationship link can be modified through its properties inspector.



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 can edit the relationship name.

Display of cardinality also depends on the selected notation. The following types of cardinality are possible:

Cardinality	IDEF1X	IE
Zero or more		
One or more		
Zero or one		
Exact cardinality (arbitrary integer value)		
Optional		

Related topics

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Import-Export Operations

EJB Assembler Diagram: Visual Assembling EJBs for Deployment

Enterprise JavaBeans (EJBs) can be used in multi-tier distributed applications, in which the business logic is usually implemented as a set of EJBs.

Together *EJB Assembler diagram* allows to model the way EJBs are assembled into an application. With the help of EJB Assembler diagram the user can combine all EJBs in a JAR and deploy to an Application Server. EJBs are added to the *EJB Assembler diagram* as shortcuts from other diagrams.

EJB Assembler diagrams are not supported in all products. For current product information, visit **www.togethersoft.com/together** or contact the nearest TogetherSoft sales office.

Together 's *EJB Assembler diagram* supports J2EE specifications (J2EE Support): EJB-references, security references, resource references, and environment references. You can show method permissions for the business methods of EJB's classes and link these to security roles that you define. All references are implemented as separate visual design elements. This way is very useful for linking EJBs, resources and other elements, and for providing reusability.

EJB Assembler diagram is a visual equivalent of the *Application Assembler* described by the EJB 2.0 Specification. According to the specification, the assembler "assembles enterprise beans into a single deployment unit." It provides application assembly information to the *Application Deployer*, which is represented in Together by the *J2EE Deployment Expert*. Refer to EJB 2.0 Specification, p. 432 for details.

Content

EJB Assembler diagrams contain:

	**Security Role**: Defines security role that stands for one of the recommended security roles for the EJB's client(s). *Security Role* element in an *EJB Assembler diagram* presents a simplified view of the EJB app's security to the app deployer (i.e., the *J2EE Deployment Expert*).
	**Principal**: Creates a visual design component representing a *User* or *Group of users* separated from their *Security Role*.
	**Method permission**: Enables you to show the required permission(s) on one or more methods. Represents a binary relation between the security roles and the methods of an EJB's home and remote interfaces. Represents a permission required to invoke methods on these interfaces.
	**EJB Properties**: Creates a visual design component representing EJB's properties as a separate element with its own properties
	**Container Transaction**: Enables to define transaction attributes for the methods of EJB home and remote interfaces. When linked to one or more EJB methods, it specifies that the linked method(s) are assigned the transaction attribute value defined in the *Container Transaction* element's properties (Alt - Enter). All methods linked to a single *Container Transaction* element should belong to a single bean.
	**EJB Reference**: Creates element with its own properties in the visual model, representing a reference to an EJB.
	**Environment**: Creates a visual design component with the properties of some static constant, which cannot be changed after EJB deployment.
	**Resource Reference**: Creates a visual design component which has all properties of the referenced resource.

	**Message Web Service**: Creates message Web Service for BEA Weblogic 6.1
	**Assembly Link**: Drawn between diagram elements to show their relationships in the application.
	**Note**: Creates a visual Note element
	**Note Link**: Creates a link between the note element and another visual component to show the note's relationship with this element.

The main difference between this diagram and other diagram types is that a key element of the diagram is not placed into the diagram using the *Diagram* toolbar. You need to create a shortcut to each EJB that you want to show in the Assembler diagram (see information below).

Notation

The *EJB Assembler diagram* is not specified in the UML. It is specific to Together as a part of J2EE specification support. You can consider this diagram an extension of standard UML.

Security roles

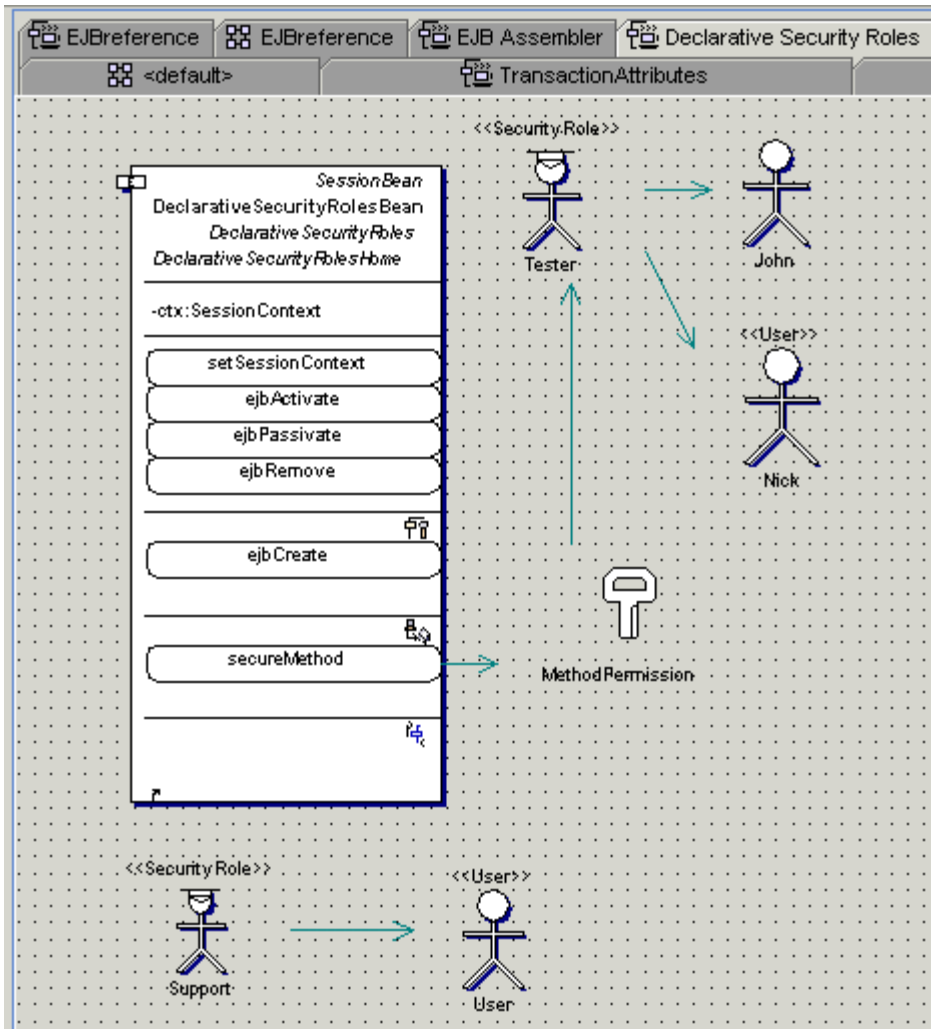
Together *EJB Assembler diagram* implements two variants of *Security Role*'s definition:

- Declarative Security Role;
- Programmatic Security Role;

According to J2EE specification (p.136), the *Declarative security* is realized in a form external to the application:

> "...The deployment descriptor is the primary vehicle for declarative security in the J2EE platform.
>
> A deployment descriptor is a contract between an Application Component Provider and a Deployer or Application Assembler. In the context of J2EEsecurity, it can be used by an application programmer to represent an application's security related environmental requirements. Groups of components are associated with a deployment descriptor.
>
> The application's logical security requirements are mapped by a Deployer to a representation of the security policy that is specific to the environment at deployment time.
>
> A Deployer uses a deployment tool to process the deployment descriptor. At runtime, the container uses the security policy that was derived from the deployment descriptor and configured by the Deployer to enforce authorization".

For this purpose Together provides a special visual element called *Method Permission*.



How to define the Declarative Security Role in the EJB Assembler diagram

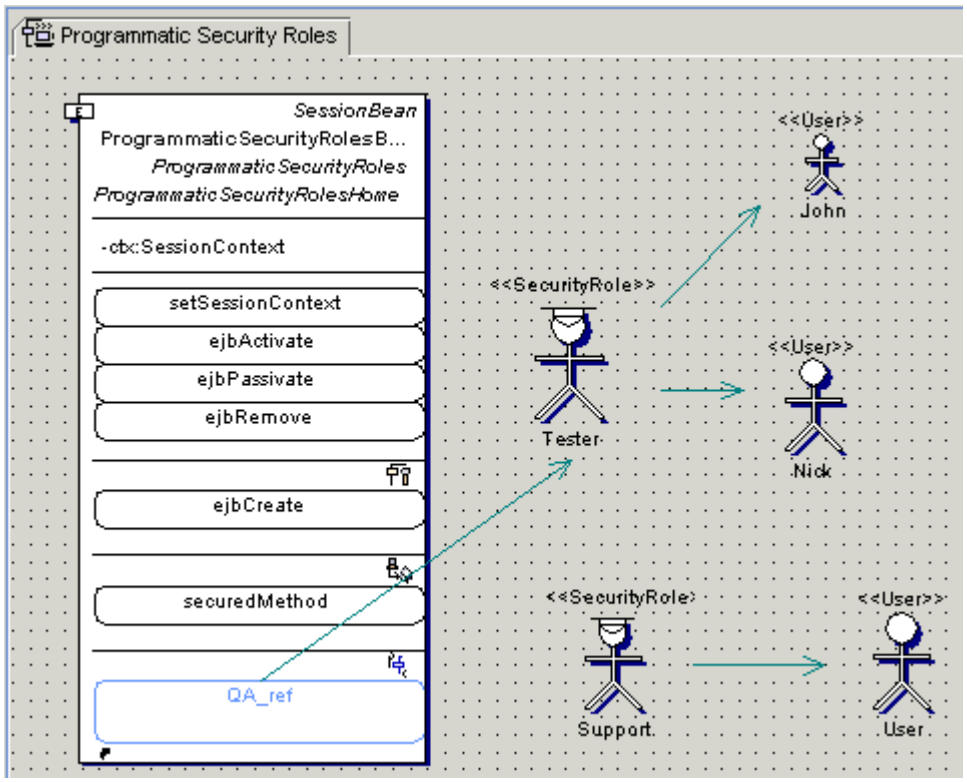
Programmatic security is used by security aware applications. According to J2EE Specification, " Programmatic security consists of two methods of the `EJBContext` interface and two methods of the `HttpServletRequest` interface:

isCallerInRole (EJBContext)
getCallerPrincipal (EJBContext)
isUserInRole (HttpServletRequest)
getUserPrincipal (HttpServletReque

These methods allow components to make business logic decisions based on the security role of the caller or remote user. They also allow the component to determine the principal name of the caller or remote user to use as a database key, for example."

In this case the user has to add *Security Role* icons to the *EJB Assembler diagram (support* and *tester* in the example) and link them to the corresponding principals (John, Nick, Robert).

If EJB has a security role reference, the user defines *EJB Security reference (QA_ref)* as an attribute of EJB. This security reference can be linked to the *Security Role* icon (*tester* in our case). Then the *J2EE Deployment Expert* will generate information about properties of *tester*-reference in the Deployment Descriptor.



How to define the Programmatic Security Role in the EJB Assembler diagram

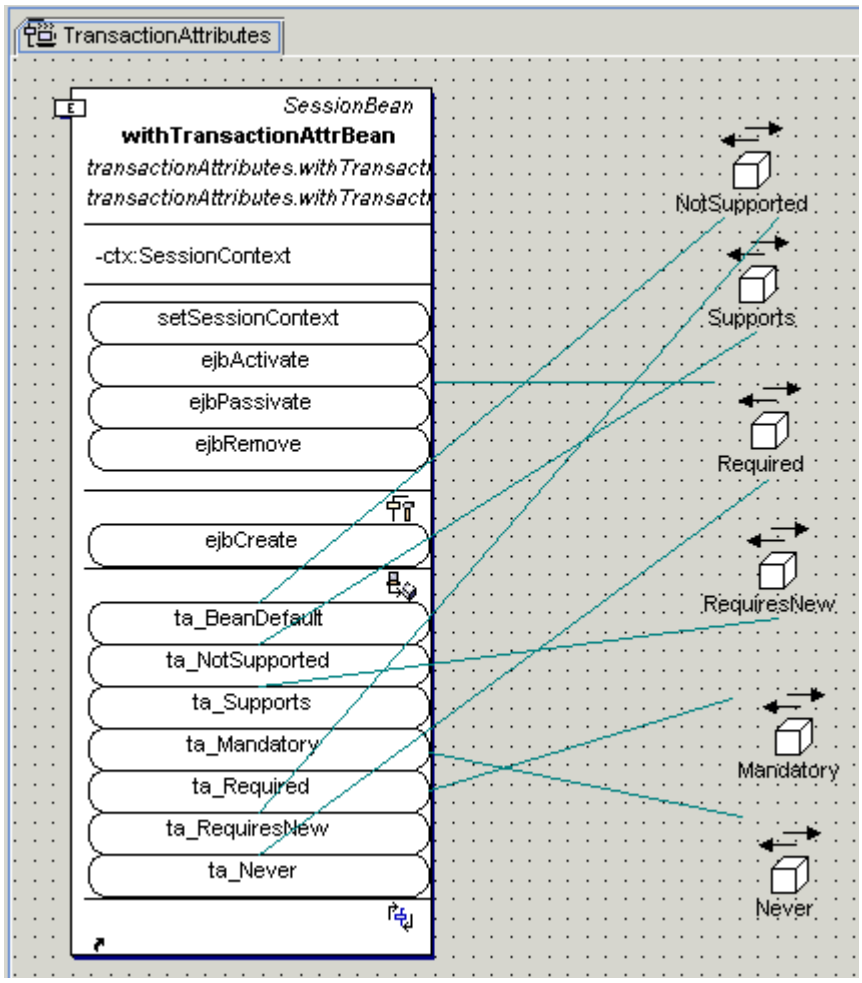
Container Transaction

There is a definition of a transaction attribute in "Sun Microsystems Enterprise Java Beans 2.0" specification:

> "...Every client method invocation on an enterprise Bean object is interposed by the container. The interposition allows for delegating the transaction management responsibilities to the container.
>
> The declarative transaction management is controlled by a transaction attribute associated with each enterprise Bean's home container. The container provider's tools can be used to set and change the values of transaction attributes.
>
> Enterprise JavaBeans define the following values for transaction attribute:
>
> ```
> TX_NOT_SUPPORTED
> TX_BEAN_MANAGED
> TX_REQUIRED
> TX_SUPPORTS
> TX_REQUIRES_NEW
> TX_MANDATORY
> ```
>
> The transaction attribute is specified in the enterprise Bean's deployment descriptor."

See description of possible transaction attribute variants in "Sun Microsystems Enterprise Java Beans 2.0" specification (p.100).

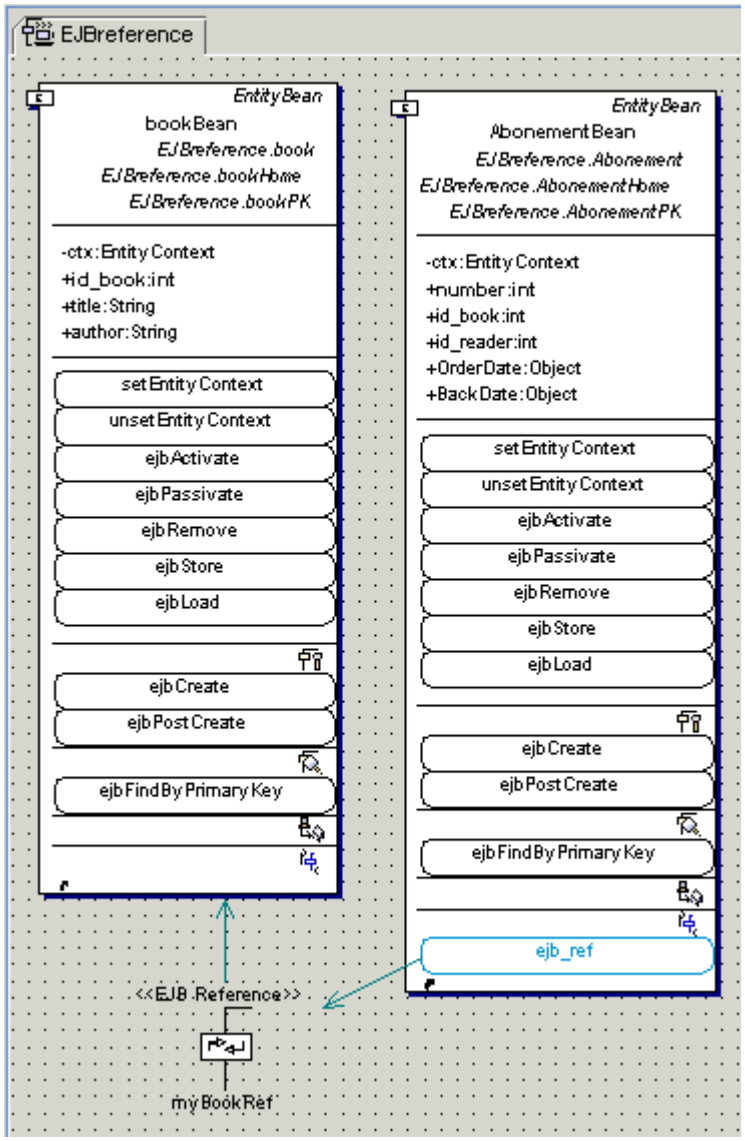
Together allows to define a transaction attribute using the *Container Transaction* visual component. The user defines appropriate methods of EJB and draws the links between these methods and the required *Container Transaction* elements:



When security roles, transaction attributes, and references are completely defined, you can invoke the J2EE Deployment Expert. If the Deployment Descriptor is required, don't forget to check the *Generate JAR Deployment Descriptor* option on the first page of the J2EE Deployment Expert.

EJB References

EJBs are not allowed to refer directly to each other. One EJB can refer to another EJB only via a visual EJB Reference element. User has to define an *EJB reference* (*ejb_ref*), add the EJB Reference icon (*bookBean*) to the EJB Assembler diagram, and draw the links:



Working with EJB Assembler diagrams

Apart from the basic mechanism of drawing diagrams, there are several fundamental things you should understand about EJB Assembler diagrams:

Creating EJB shortcuts

How EJBs are displayed in the diagram

Showing or hiding various elements of an EJB

How the EJB Assembler diagram relates to the J2EE Deployment Expert

Working with references

Creating EJB shortcuts

By the time you are ready to assemble one or more EJBs into an application, you should have completed EJBs in your Together project. If you want to use the EJB Assembler diagram to specify security roles, method permissions, etc. for deployment, you need to display the relevant bean(s) in the EJB Assembler diagram. You can do this by creating one or more shortcuts to the finished EJB(s) that comprise the application. A shortcut is just a visual representation of some EJB element that "lives" somewhere else in the project.

To begin:

1. Create a new EJB Assembler diagram, or open an existing one in the project that contains the EJBs for the application (File | New Diagram).

2. If you want to include EJBs that are not part of the current project, specify the path(s) to them in the Search/Classpath tab of the Project Properties dialog (File | Project Properties - Advanced).

To create a shortcut:

1. Right-click on the background of the open EJB Assembler diagram and choose *Add shortcut*.

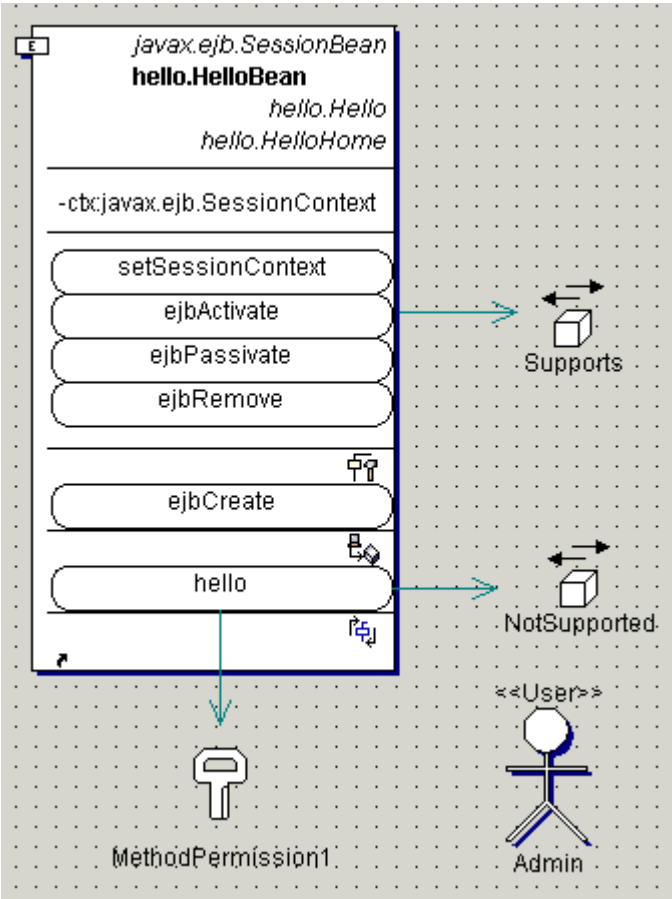
2. In the Add Shortcuts dialog, expand the Model node and locate the EJB classes and interfaces from your project which you want to display in the diagram. Select them in the tree view and click *Add*.

3. In the Add Shortcuts dialog, expand the *Search/Classpath* node and locate the EJB classes and interfaces from outside your project (if any) that you want to display in the diagram. Select them in the tree view and click *Add*.

4. Click *OK* to display selected EJB(s) on the diagram. **Tip:** Run auto-layout from the diagram speedmenu at this point.

How EJBs are displayed in the EJB Assembler diagram

EJBs are displayed in essentially the same kind of visual container as Class diagrams. The methods show as elliptical objects within the Class framework. These objects are highlighted when there are link sources or targets.



How an EJB displays in EJB Assembler diagram

Showing and hiding EJB elements

When creating your EJB shortcuts in the EJB Assembler diagram, you can select home or remote interfaces, or primary key classes in the selection dialog. If they don't appear in the diagram, it means that your View Management option settings hide them. These elements are hidden by default, and you need to change the settings if you want to see them. To preserve the settings at the project level, you can just change them for the specific diagram.

1. Right-click the diagram background and choose *Diagram Options*.
2. Select the *View Management* page and navigate to the *Show* node.
3. Expose the EJB-related options and check those elements that you want to display in the current diagram.

You can also hide individual elements in the diagram using the *Hide* command from the element's speedmenu. Restore with the *Show Hidden* command on the diagram speedmenu.

How the EJB Assembler diagram relates to the J2EE Deployment Expert

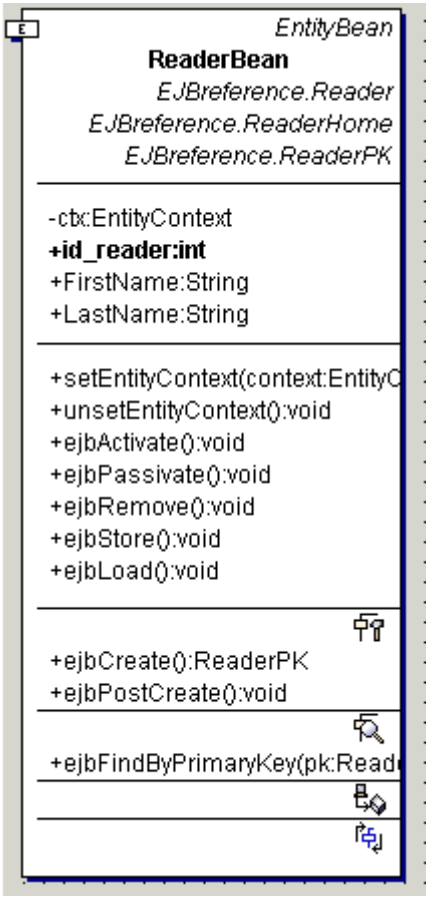
Together provides an "expert" dialog that simplifies the process of deploying EJBs. You can run the J2EE Deployment Expert against either a Class diagram, or an EJB Assembler diagram : Main menu | Tools | J2EE Deployment Expert.

- For "fast track" deployment with default values, or deployment prototyping, use the Expert from an appropriate Class diagram.
- If you need "full featured assembly information" with control over security and permissions, use the J2EE Deployment Expert from an EJB Assembler diagram.

For information on using the J2EE Deployment Expert, see *Deploying Enterprise JavaBeans*.

Working with references

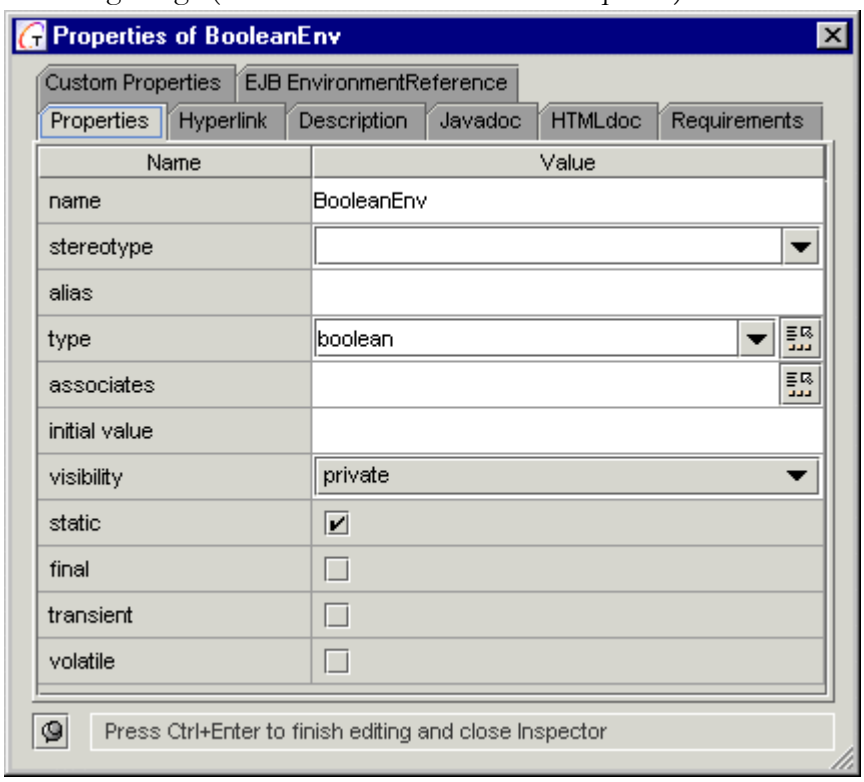
Use the following example of *Environment reference* to learn how to define references. The other types of references are defined similarly. First, define EJB-element (here *bookBean*) and its attributes in the Class diagram (*EJB Reference*).



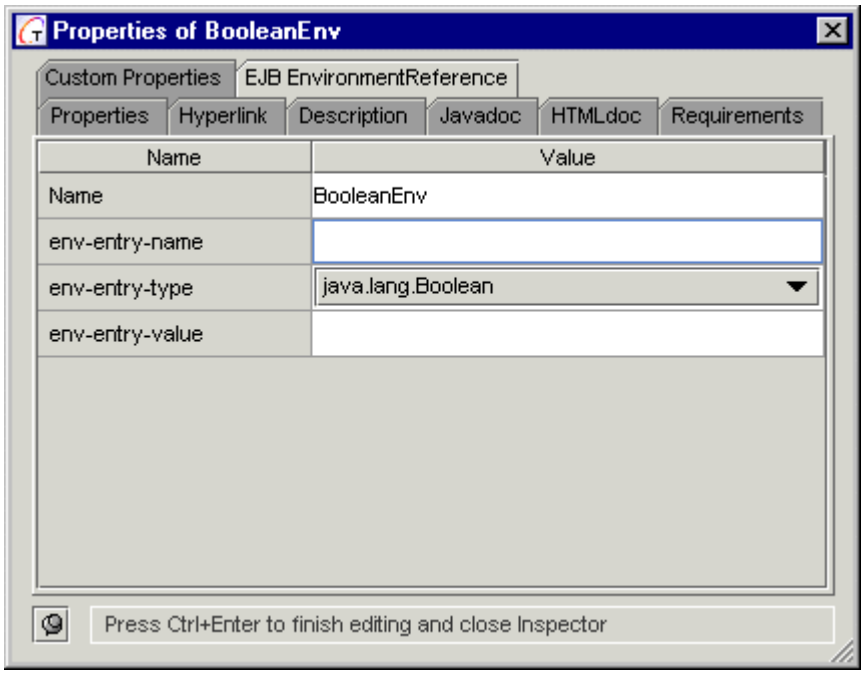
Next, create an EJB Assembler diagram (New diagram | Together | EJB Assembler) and add the shortcut to the *bookBean* (select *Add shortcut* from the diagram speedmenu and choose the necessary EJB in the Class diagram EJB Reference).

If there are environment variables in the project, the user can add Environment references as EJB's attributes (New | EJB Environment Reference on the EJB speedmenu). In this example we have environment variables of the following types: *boolean BooleanEnv, string StringEnv, integer IntegerEnv, double DoubleEnv*.

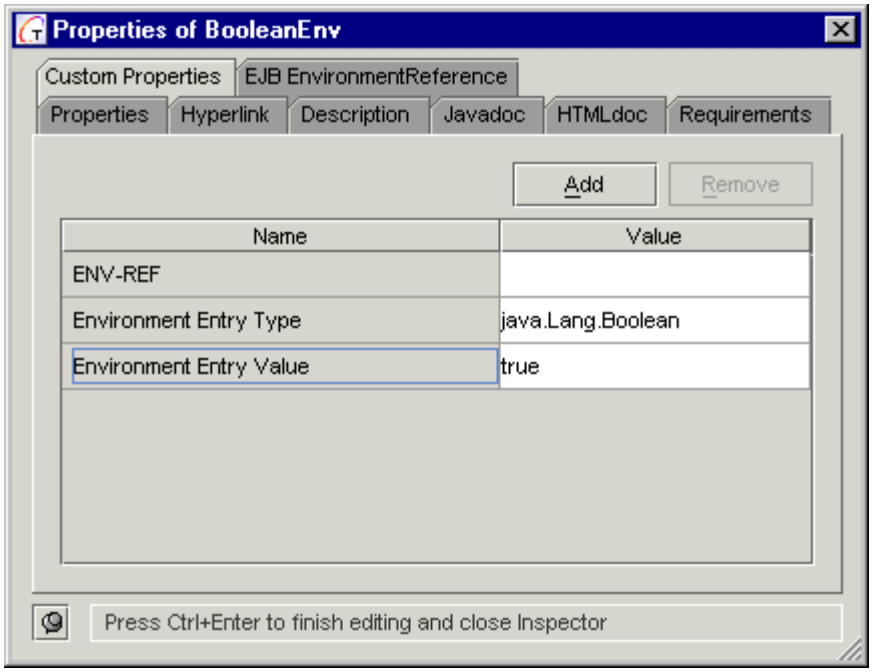
The properties of the Environment references are defined in the Properties Inspector. For example, we can define the properties of *BooleanEnv Environment reference* as shown in the following image (note that the initial value is in quotes).



On the *EJB EnvironmentReference* tab of the Inspector, specify the properties of this Environment reference:



The user can also define a specific *Environment* entry value using the Custom properties tab (True for BooleanEnv):

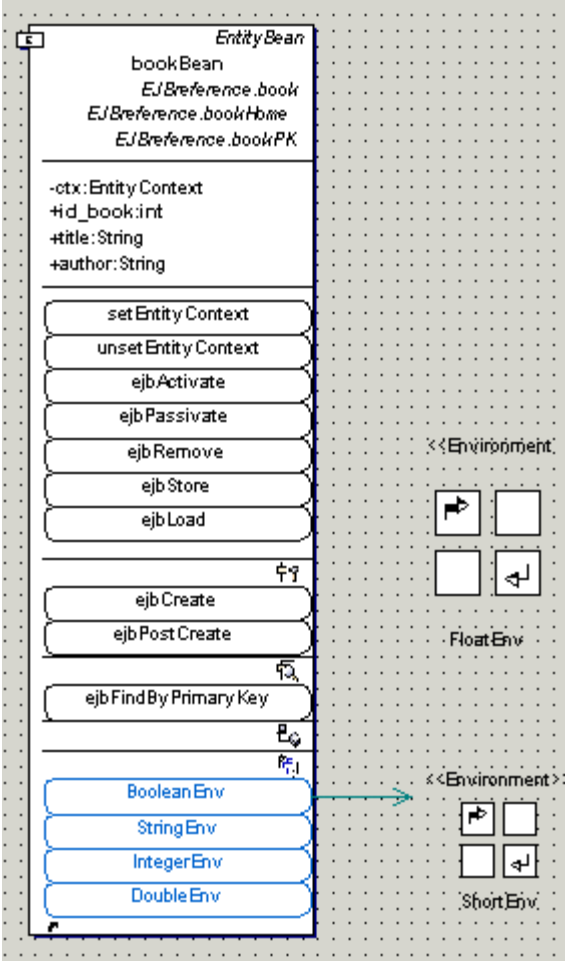


When environment references and their properties are completely defined, the corresponding records appear in the EJB's list of attributes. If the J2EE Deployment Expert is invoked from the Class (EJB Reference) diagram, the appropriate description is generated in the Deployment Descriptor (*.xml file).

However, there is an EJB Assembler diagram in Together that is designed to simplify the deployment process. Create an EJB Assembler diagram EnvironmentRef and add a new Environment element *ShortEnv*. Define its properties as described above. If you need to link *BooleanEnv* with *ShortEnv* , you can draw the Assembly link from this reference to the *ShortEnv* element.

If the properties of *ShortEnv Environment* design element and *BooleanEnv Environment* reference are different, and the J2EE Deployment Expert is called from *EJB Assembler diagram EnvironmentRef*, the properties of *ShortEnv* are preferred.

In our case the boolean type of the environment variable will be redefined as short type (See the EJB Assembler diagram).



Observe two icons on the diagram: *FloatEnv* and *ShortEnv*. The *FloatEnv* element is not linked to EJB's Environment reference and will be deployed as a separate environment variable. Draw a link from *BooleanEnv* environment reference to *ShortEnv* visual element. In this case *BooleanEnv* becomes a short type in *.xml file and the name *ShortEnv* is used as a "lookup".

Tips for Assembler diagrams

- The Properties Inspector of a *Container Transaction* element (speedmenu | Properties) enables you to define the transaction type (*Supported, Not Supported, Never* etc.)

- The Properties Inspector of the *Security Role* element (speedmenu | Properties) enables you to define the name of the element.

- You can create a link from one or more *Method Permission*(s) to a *Security Role*. This defines method permissions for the Security Role.

- You can create a link from the *Security Role* to one or more *Principals*. This defines the User or the Group for this Security Role.

- If an EJB defines any *Security Role Reference*(s) (shown in the last compartment of the EJB icon in the diagram), all of them must be linked to Security Role elements.

- Each *Security Role Reference* must have one and only one link to some *Security Role* element.

- One *Security Role* element can be associated with null to many *EJB Security Role References*.

- You can create a link from some method in EJB's implementation class to EJB's *Container Transaction* element or to a *Method Permission* element.
- A *Container Transaction* element can be linked to an EJB icon as a whole, and/or to sub-icons representing methods (see previous figure).
- If a Transactional Attribute is linked to the EJB icon, it means the corresponding transaction type will be used as default for all methods.
- If a Transactional Attribute is linked to method(s), it means that the corresponding transaction type will be used for the linked method. It overrides default value defined above.
- You can create a link from an EJB Reference in EJB's implementation class to EJB Reference icon.
- You can create a link from an EJB Reference icon to an EJB icon.
- You can create a link from an Environment Reference in EJB's implementation class to an Environment icon.
- You can create a link from a Resource Reference in EJB's implementation class to a Resource Reference icon.

How to Create an EJB Application Step by Step

To create an EJB Application:

1. Create all necessary Class diagrams that contain EJBs.
2. Create a new EJB Assembler diagram.
3. Create shortcuts for the necessary EJBs and add shortcuts to the EJB Assembler diagram.
4. Define EJB References and other references as the EJBs' attributes.
5. Add shortcuts to the necessary classes.
6. Define references as separate visual elements.
7. Define principals (users or groups) and add visual elements to the EJB Assembler diagram.
8. Define security roles as visual elements.
9. Draw links for the references.
10. Draw links for the security roles.
11. Define container elements.
12. Draw all the necessary links.
13. Invoke J2EE Deployment Expert.

See also

Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Working with View Management
Deploying Enterprise JavaBeans

Web Application Diagram: Visual Assembling of Web Applications for Deployment

Web Application diagram allows to visually create and build Web Application archive (WAR), where all JSPs and Servlets are stored.

If a web application is already implemented as a set of Servlets, JSPs and other Web files (for example, pictures), it is possible to generate the *Deployment Descriptor* and assemble Web components to the archive (WAR) with *.war extension.

Together's *Web Application diagram* enables modeling the way JSPs, Servlets and other web files are assembled into an application. With the help of a Web Application diagram the user can combine all JSPs, Servlets and other Web files in WAR and deploy it to an Application Server.

Web Application diagrams are not supported in all products. For current product information, please visit **www.togethersoft.com/together/** or contact your nearest TogetherSoft sales office.

Content

Web Application diagram contains:

	Shortcut(s) to one or more EJBs
	**Security Role**: Creates a visual design component for security role ... meaning one of recommended security roles for Web Application 's or EJB's client(s) . The Security Role element in a Web Application diagram presents a simplified view of the JSP's (Servlets) or EJB app's security to the app deployer (i.e., *J2EE Deployment Expert*).
	**Principal**: Creates a visual design component that represents a User or a Group of Users separated from their *Security Role*. **Note**: It is possible to change stereotype using in-place editing.
	**Security constraint**: Creates a visual design component for annotating the intended protection of the web contents. Security constraint includes web resource collection, authorization constraints and user data constraints.
	**Web resource collection**: Creates a visual design component for the set of resources to be protected. A *Web Collection* is a set of URL patterns or HTTP methods that describe resources to be protected.
	**EJB Reference**: Creates an element with its own properties in the visual model that represents a reference to an EJB.
	**Environment references**: Creates a visual design component with the properties of a static constant, which cannot be changed after EJB's deployment.
	Resource references: Creates a visual design component that has all properties of the referred resource.
	JSP: Creates a visual design component that represents JSPs and Servlets.
	**Web Files**: Creates a visual design component that represents Web files (images etc.).

	**Filter mapping**: Creates a visual design component that enables mapping servlets to filters. You can link a servlet to this component, and then map it to a filter. Refer to Servlets 2.3 Specification for details.
	**Error page**: Creates a visual design component for the error page returned if the user authentication fails.
	**TagLib**: Creates a visual component used to describe the JSP tag library.
	**Web Links:** Creates a link between diagram elements to show their relationships in the application.
	Notes and note links for documentation

Key elements and properties

Similar to the *EJB Assembler diagram*, some of the key elements cannot be added to the diagram with the toolbar icons. To show EJBs in the *Web Application diagram*, a shortcut to each EJB should be created. To include Home and Remote interfaces of an EJB, you must add shortcut to this EJB in the Web application diagram.

Notation

Web Application diagram is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

Having created this diagram, the user can call *J2EE Deployment Expert* to generate the *Deployment Descriptor* and deploy the application to the selected server. If you have a separate element (for example, some reference in the *Web Application diagram*), appropriate record appears in the generated *Deployment Descriptor*. For EJBs, Home and Remote interfaces are generated.

Properties of the Web Application diagram

Web Application diagram has certain Web properties, which are available in the *WebProperties* tab of the diagram object inspector. Specifying the properties, keep in mind that the application server should comply with the Sun Microsystems specification for EJB 1.1 and higher. Refer to **http://java.sun.com/products/ejb/docs.html** for details.

Module name: The name of `*.war` file in course of deployment process

Welcome File List: This property allows to define welcome pages in the Deployment Descriptor of a Web Application (web.xml). Welcome file list is an optional element that contains an ordered list of welcome file elements. When URL refers to a directory name, application server is the very first file on this list. If this file is not found, the server tries the next one. Actually, Welcome File List is the list of files, which can be used as a starting page. This is useful for making your site more friendly, since the user may type a URL without giving a specific filename.

Welcome pages are defined on the Web Application level. If your Server is hosting multiple Web Applications, you need to define welcome pages separately for each Web Application. If the Welcome Pages are not defined, WebLogic Server looks for the following files in the following order:

```
index.html
index.htm
index.jsp
```

Enter the list welcome pages in the appropriate field of the object inspector, using comma as a delimiter.

Error pages: You can configure the application server to display custom web pages or other HTTP resources in response to particular HTTP errors or java exceptions. In this case the user-defined pages appear instead of the standard error pages of the application server. If the Error page property is defined, the corresponding error-page element of the Deployment Descriptor (web.xml) appears. This optional element maps error code or exception type with the resource path in the Web Application.

Session timeout. This is an optional element defining one of the session parameters for the Web Application - the time in seconds that WebLogic Server waits before timing out a session. Minimum value is 1, default is 3600, and maximum value is integer MAX_VALUE. This parameter allows to override the session-timeout element in web.xml.

Small icon: Use this optional parameter to select small icon appearance for Weblogic Icon Element. In this case you must specify location of a small (16x16 pixels) *.gif or *.jpg image used to represent the Web Application in the GUI tool.

Large icon: Use this optional parameter to select large icon appearance for the Weblogic Icon Element. In this case you have to specify location of a large (32x32 pixels) *.gif or *.jpg image used to represent the Web Application in the GUI tool.

Context Root : optional parameter that defines context root for the Web Application.

Example:

```
<web> <web-uri>petStore.war</web-uri> <context-
  root>estore</context-root> </web>
```

Servlet Context Params: use this optional parameter to define a context-param element that declares context initialization parameters of the web Application servlet. The following methods are used to access these parameters:

```
javax.servlet.ServletContext.getInitParameter()
javax.servlet.ServletContext.getInitParameterNames()
```

Working with Web Application diagrams

Besides the basic technique of drawing diagrams, there are several fundamental things you should understand about Web Application diagrams:

Creating EJB shortcuts

Creating filters

How EJBs are displayed in the diagram

Showing and hiding EJB elements

How the Web Application diagram relates to the J2EE Deployment Expert

Creating EJB shortcuts

By the time you are ready to assemble one or more EJBs into an application, you should have completed EJBs in your Together project. If you want to use the *EJB Assembler diagram* to specify security roles, method permissions, etc. for deployment, you need to display the relevant bean(s) in the Assembler diagram. You do this by creating one or more shortcuts to the completed EJB(s) that comprise the application. A shortcut is just a visual representation of some element that "lives" somewhere else in the project.

To begin:

1. Create a new *Web Application diagram*, or open an existing one in the project that contains the EJBs for the application (*File | New Diagram*).

2. If you want to include EJBs that are not part of the current project, specify the path(s) to them in the Search/Classpath tab of the *Project Properties* dialog (*File | Project Properties - Advanced*).

To create a shortcut:

1. Right-click on the background of the open *Web Application diagram* and choose *Add shortcut*.

2. In the *Add Shortcuts* dialog, expand the Model node and locate the EJB classes and interfaces from your project which you want to display in the diagram. Select them in the tree view and click *Add*.

3. In the *Add Shortcuts* dialog, expand the *Search/Classpath* node and locate the EJB classes and interfaces from outside your project (if any) that you want to display in the diagram. Select them in the tree view and click *Add*.

4. Click *OK* to display the selected EJB(s) in the diagram. Tip: Run auto-layout from the diagram speedmenu at this point.

Creating and mapping filters

The objective of Filter mapping design component is to provide filter mapping element for the deployment descriptor.

To create a filter or a collection of filters in a Web Application diagram:

1. Right-click on the Filter mapping element to invoke its Properties Inspector,

2. Choose the *MappedFilters* tab.

3. Click *Add* to create a filter.

4. Add as many filters as required and press CTRL+Enter apply changes and close the Inspector.

This generates a filter class that implements `javax.servlet.Filter` interface. This class is attached to the Filter mapping element by a Web link. Properties Inspector of the filter class provides the *FilterProperties* tab where you can specify the following properties:

Filter name: used to map the filter to a servlet or URL

Display name: visible name of the filter

Icon: fully-qualified path to an image file used as the filter's icon

Init Params: initialization parameters for a filter.

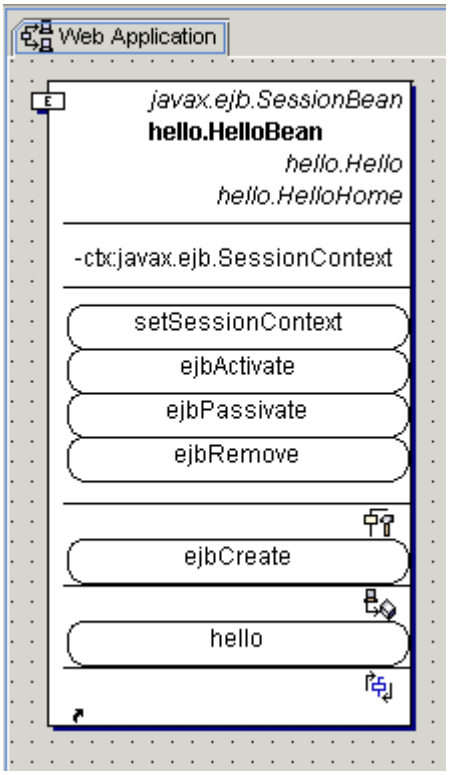
All filters of a Filter mapping element are listed in its properties inspector.

Note that *Remove* button of the Filter mapping Inspector only deletes the link between a Filter mapping and a filter. The filter class is still preserved on the diagram.

Use *URLPatternMapping* tab of the Filter mapping Inspector to define groups 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.

How EJBs are displayed in diagrams

EJBs are displayed in essentially the same kind of visual container as the *Class diagrams*. The various methods show as elliptical objects within the Class framework. These objects exhibit highlighting when they are link sources or targets.



Showing and hiding EJB elements

When creating your EJB shortcuts in the *Web Application diagram,* you can select home or remote interfaces, or primary key classes in the selection dialog. If they don't appear in the diagram, it means that your *View Management* option settings are hiding them. These elements are hidden by default, and you need to change the settings if you want to see them. To preserve the settings at the project level, you can just change them for the specific diagram.

- Right-click the diagram background and choose *Diagram Options*.
- Select the *View Management* page tab and navigate to the *Show node*.
- Expose the EJB-related options and check the elements that you want to display in the current diagram.

You can also hide individual elements in the diagram using the *Hide* command from the element's speedmenu. Restore with the *Show Hidden* command on the diagram speedmenu.

How the Web Application diagram relates to the Deployment Expert

Together provides an *J2EE Deployment Expert* dialog that simplifies the process of deploying Servlets, JSPs, Web files and possibly EJBs. You can run the *J2EE Deployment Expert* against either a *Class diagram,* or an *Web Application diagram* : *Main menu | Tools | Deployment Expert*.

For "fast track" deployment with default values, or deployment prototyping, use the *J2EE Deployment Expert* from an appropriate *Class diagram*. If you need "full featured assembly information" with control over security and permissions, use the *J2EE Deployment Expert* from the *Web Application diagram*.

J2EE Deployment Expert is used same way as in case of *EJB Assembler diagram*, with the only difference: check *Generate WAR Deployment Descriptor* option in the *J2EE Deployment Expert*'s dialog.

When deployment is performed from the *Web Application diagram*, Together takes files with the directory structure from the *WebFiles* source folder.

Tips for Web Application diagrams

- Properties Inspector of a JSP element (speedmenu | *Properties*) enables defining the component's type (JSP or Servlet) and Servlet Init Parameters.

- JSP Editor is accessed from *Web Application diagram*.

- Properties Inspector of a Security Role element (speedmenu | Properties) enables defining the element name.

- You can create a link from a *Security Role* to one *Principal.*

- If it is necessary to define an EJB Reference, use EJB Reference visual component linked to the corresponding EJB.

- To define Resource and Environment references, use separate visual Resource Reference element and Environment Reference element.

- To define Web files, use the separate visual WebFiles element placed on the Web Application diagram.

- If a JSP or Servlet defines any *Security Role*, it must be linked to the corresponding *Security Role* element.

- If it is necessary to define an *EJB Reference*, use *EJB Reference* visual component linked to the corresponding EJB.

- To define *Resource* and *Environment* references, use separate visual *Resource Reference* element and *Environment Reference* element.

- To define Web files, use the separate visual *WebFiles* element placed on the *Web Application diagram.*

See also

Creating diagrams in projects

Drawing diagram elements

Opening diagrams

View Management

Deploying Enterprise JavaBeans

Enterprise Application Diagram : Visual Assembling of Enterprise Applications for Deployment

To support J2EE specification, a tool is required that enables to collect elements for EAR in a single diagram. Such tool is the *Enterprise Application diagram* that combines Web Applications and EJB Applications. Thus *Enterprise Application Diagrams* are used to visually assemble Enterprise Applications for deployment.

We suppose that the developer already has one or more EJB Assembler and Web Application diagrams.

Enterprise Application diagram contains shortcuts of EJB Assembler and Web Application diagrams. This diagram provides all necessary elements for generating the Deployment Descriptor and creating *.ear archive file.

> **Note**: If the target application server doesn't support *.ear generation (which is the case for WebLogic 5.1 and WebSphere 3.5), multiple *.jar files are generated instead. WebLogic 6.0 can deploy *.ear's, and thus a single *.ear file is generated, when deployment is performed from the Enterprise Application diagram.

Enterprise Application diagram supports *Security Roles*.

Enterprise Application diagrams are not supported in all products. For current product information, please visit www.togethersoft.com/together/ or contact the nearest TogetherSoft sales office.

Creating and drawing Enterprise Application diagrams

If you need to learn how to create new diagrams in projects, or the techniques for placing elements and drawing links, consult the User's Guide topics found in the Part II of this manual (Working with Diagrams).

Content

Enterprise Application diagrams contain:

	**Module:** Creates a visual shortcut to an existing *EJB Assembler, Web Application* or *Class diagram*.
	**Archived Module:** Creates a visual shortcut to an existing *.jar file.
	**Security Role:** Defines the *Security role* that stands for one of recommended security roles for the EJB's client(s). *Security Role* element in an *Enterprise Application diagram* presents a simplified view of the EJB app's security to the app deployer (i.e. the *J2EE Deployment Expert*).
	**Note:** Creates a visual *Note* element.
	**Note link:** Creates a link between the note element and another visual component to show the note's relationship with this element.

The *Enterprise Application diagram* is not specified in the UML. It is specific to Together as a part of the product's EJB development and J2EE specification support.

Creating diagram shortcuts

By the time you are ready to assemble one or more diagrams into the *Enterprise Application diagram*, you should have the completed *Web Application* and *EJB Assembler diagrams* in your Together project. As in the case of EJB shortcut, a diagram shortcut is just a visual representation of some element that 'lives' somewhere else in the project.

To begin:

In your application, create a new Enterprise *Application diagram* (*File | New Diagram*), or open an existing one in the project that contains *Web Application* and *EJB Assembler diagrams*. If you want to include diagrams that are not part of the current project, specify the path(s) to them in the *Search/Classpath* tab of the *Project Properties* dialog (*File | Project Properties - Advanced*).

To create a shortcut:

Select *Module* icon on the diagram toolbar and draw an element on the Diagram pane. This invokes Selection manager dialog that enables choosing available *EJB Assembler, Web Application* or *Class diagrams* to be added to the current *Enterprise Application diagram.*

Alternatively, you can right-click on the background of the *Enterprise Application diagram* and choose *Add shortcut* command on the diagram speedmenu, to invoke Add Shortcut dialog, expand the *Model node* and locate the *Web Application or EJB Assembler diagram*, which you want to display in the resulting *Enterprise Application diagram.* Select them in the tree view and click *Add.*

Working with Enterprise Application Diagrams

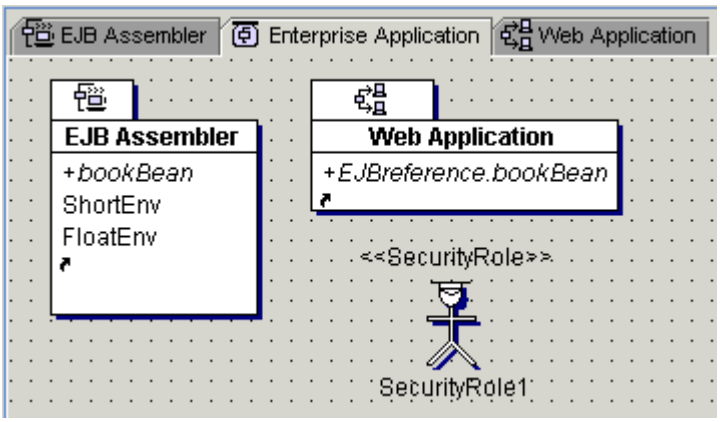
Outside of the basic mechanism of drawing diagrams, there are some fundamental things you should understand about *Enterprise Application diagrams*:

How EJB Assembler and Web Application diagrams are displayed in the Enterprise Application diagram

How the Enterprise Application diagram relates to the J2EE Deployment Expert

How EJB Assembler and Web Application diagrams are displayed in the Enterprise Application diagram

Diagrams are displayed in a kind of a visual container that includes all internal elements, as shown below:



How the Enterprise Application diagram relates to the J2EE Deployment Expert

Together's deployment expert simplifies the process of deploying EJBs to various application servers. You can run the *J2EE Deployment Expert* against either a *Class diagram* or an *Enterprise Application diagram* using *J2EE Deployment Expert* command on the *Tools* menu. For information on using the *J2EE Deployment Expert* see Deploying Enterprise JavaBeans.

TagLib Diagram

TagLib diagram allows to easily create tag handlers, and declare the tags in a special file called *tag library descriptor*. The tag handlers must implement either of the two interfaces `Tag` or `BodyTag`, or extend one of the classes `TagSupport` or `BodyTagSupport`. To learn more about the tags and tag handlers and their implemented methods, refer **http://www.javasoft.com/products/jsp/tutorial/TagLibraries.pdf.**

TagLib diagram helps minimizing the amount of Java coding in JSP applications and facilitates re-usability of statements used in the tag handlers in JSP's.

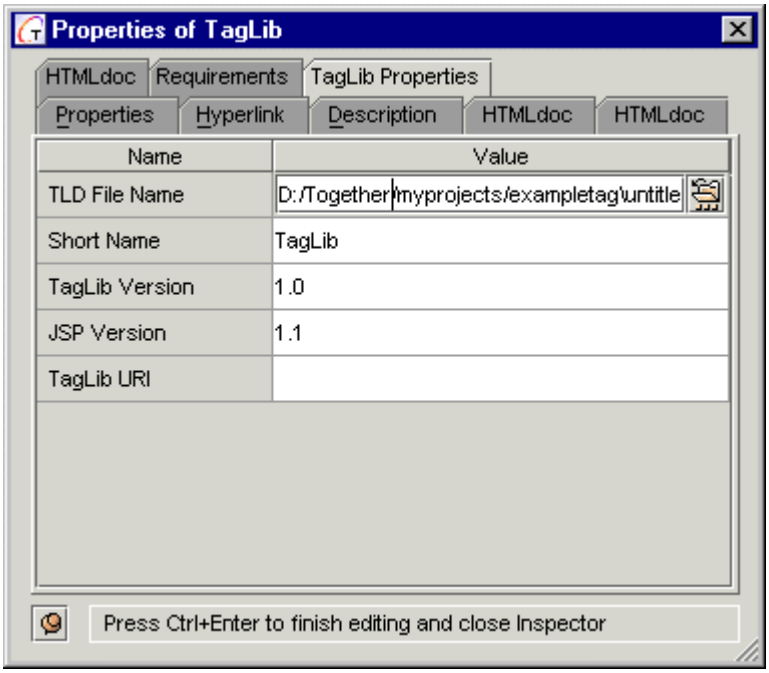
Content

TagLib diagram contains the following design elements:

	Creates a handler for a simple tag without a body. The handler class implements `Tag` interface.
	Creates a handler for tag with attributes. The handler extends `TagSupport` class.
	Creates a handler for a tag with a body that contains tags, scripting elements, html text etc. between start and end tags. The handler class implements `BodyTag` interface.
	Creates a handler for a body tag with attributes. The handler class extends `BodyTagSupport`.
	Creates a visual design element for a class that extends `TagExtraInfo`, and provides information to the JSP container about the scripting variable.
	Notes and note links

Properties

Inspector of a TagLib diagram contains specific page *TagLib Properties* that define a tag library.



TLD File Name

This field allows to specify fully qualified name of the tag library descriptor, or select it using the *File Chooser* button. The descriptor generated by *Generate TLD* command of the diagram speedmenu resides in this location.

Short name

A simple default name that could be used by a JSP page authoring tool to create names with a mnemonic value; for example, short name may be used as the preferred prefix value in `taglib` directives and/or to create prefixes for IDs.

TagLib Version

Version of the tag library.

JSP Version

The JSP specification version used by the tag library.

TagLib URI

The URI that uniquely identifies the tag library.

Working with the TagLib diagram

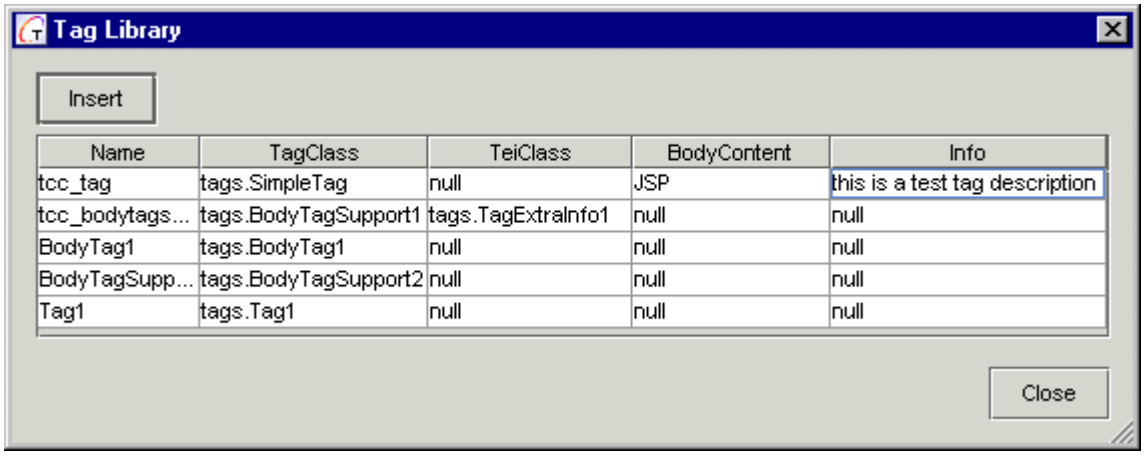
Apply usual technique to create the diagram and populate it with the necessary elements. For each element specify its tag name, implemented TEI class, and tag body content (if any). Additional information can be added in the *Description* tab of the object inspector. Having created the diagram, right click on the diagram pane to display the speedmenu and select *Generate TLD* command. Thus tag library descriptor file is written to the location specified in *TLD file name* field of the diagram inspector.

Tag handlers are referred to from the Web Application diagram. To add a tag handler to the WAD, choose *TagLib* button on the diagram toolbar and click on the diagram pane. This invokes Select *TagLib diagram dialog* in the form of Selection manager. Expand the desired node and select taglib to be added.

Subsequently, working with JSP applications, you have to specify that your JSP is supposed to employ certain tag library, using the `taglib` directive before any custom tag (refer to the tutorial for details).

Tag Library Helper

To insert the tags from your library, right click on the JSP editor and select *Tag Library Helper* command from the Tools node. The helper displays contents of the tag library according to the current library descriptor.



Choose the required tag and click *Insert* to add the tag to the desired line in your JSP code.

See also

How to Use Taglibs in a Web Application

JSP and HTML Editor

XML Modeling

XML Structure Diagrams

XML structure diagram enables you to create an XML structure definition from scratch. It is assumed that you are familiar with the basics of XML, since training is beyond the scope of this help topic. You can refer to the following books on XML:

1. Java and XML (O'Reilly Java Tools) -- Brett McLaughlin, Mike Loukides
2. XML by Example (By Example) -- Benoit Marchal
3. Professional XML -- Mark Birbeck, et al

To learn more about XML Scheme, refer to XML Schema Part 0: Primer (02 May 2001, David C. Fallside), XML Schema Part 1: Structures (02 May 2001, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn), and XML Schema Part 2: Datatypes (02 May 2001, Paul V. Biron, Ashok Malhotra) at **http://www.w3.org/TR/.**

Alternatively, you can create an XML structure diagram by importing an existing DTD or XSD file. This can be very useful if you are not familiar with the DTD or XSD and want to get a quick overview of the elements it contains, and their relationships.

> Warning: **If you want to use an XML structure diagram created in previous versions of Together, you have to invoke Convert to New Diagram Format command on the diagram speedmenu. Failing to do this leads to incorrect diagram behavior.**

Content

XML diagrams contain the following components, represented by the buttons on the toolbar:

Icon	Name
	**Element**: creates a visual design component of an element type.
	**Groups**: this component supports re-use of elements and attributes in an XML Structure diagram; creates a visual design component of a group, Sequence by default. Select the required compositor property in the *Properties* tab of the inspector, or on the group speedmenu.
	**Reference Link**: cardinality of a link is selected from the link speedmenu. The following cardinality values are possible: **?** Follows an element or group of elements and indicates that it occurs zero times or once. ***** Follows an element or group of elements and indicates that it occurs zero or more times. **+** Follows an element or group of elements and indicates that it occurs one or more times. **1** Follows an element or group of elements and indicates that it is required.
	**Attribute**: this design element can be created by a toolbar button, or from an element speedmenu.
	**Attribute Group**: another means of reusability. Creates a visual component for groups of attributes that are accessible for the elements of the diagram. This design component can be created by a toolbar button, or from an element speedmenu.
	**Data type**: creates a visual design component for an arbitrary data type by means of inheritance (*type* field in the *Properties* tab of the inspector). This component is specific for XSD.

Icon	Name
	**Complex type**: creates a complex data type component that allows elements in its content and may carry attributes. This component is specific for XSD.
	**Entity**: creates a visual design component for an entity. This component is specific for DTD.
	**Notation**: creates identification for external binary entities format. This component is specific for DTD.
	**Notes and Note Links**

There are two possible formats of an XML Structure diagram: DTD and XSD. As you can see, the sets of elements for DTD and XSD formats are overlapping: the majority of design elements are common for both formats. However, there are elements specific for a particular format, which are not enabled in another format. Thus, special care should be taken when exporting a diagram to a DTD or XSD file, to avoid loss of information.

Creating XML Structure Diagram

If you need to learn how to create new diagrams in a project, or the techniques for placing elements and drawing links, consult the User's Guide topics found under " Working with Diagrams" in the Table of Contents. Also refer to the topic Creating diagrams in projects.

XML Structure diagram is implemented as an activatable module. Thus, you have to check the flag *XML Structure diagram* in the list of Activatable modules under Options menu.

Note that the set of toolbar buttons depends on the selected diagram format.

Changing XML Diagram Format

When creating a diagram, choose its format in the diagram properties: open *Properties* tab of the diagram inspector and select DTD or XSD Schema from the drop-down list of the *format* field. The set of controls in the inspector is specific for the selected format.

When converting from one format to another a warning message is displayed that the elements irrelevant to the selected format will be permanently deleted from diagram. For example, if DTD format is selected, entities and notations disappear from the resulting diagram.

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

Example:

- Create XML diagram in XSD format.
- Add an element (Element1) and a complex type (ComplexType1).
- Add an attribute to the complex type (Attribute1).
- Select the Element1 and change its base type (field *base*) in the *Properties* tab of the inspector to ComplexType1.
- Change format from XSD Schema to DTD, and observe that Attribute1 adds to the Element1 in the resulting diagram.

Step by Step How to Create XML Structure Diagram

Here we shall consider creation of a sample XML diagram that represents the structure of personnel management data in XSD format.

Creating XML Structure diagram

Activate the module *XML Structure Diagram*. To do this, select Options | Activatable modules | XML Structure Diagram and check appropriate checkbox.

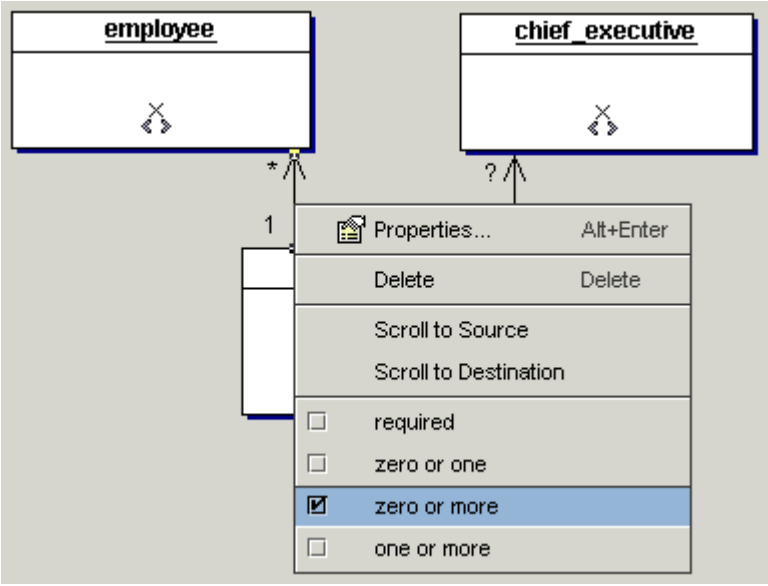
To create a new XML diagram, select File | New Diagram and choose *XML Structure Diagram* from the *Together* tab.

Right-click on the diagram and invoke the diagram inspector. In the *format* field of the *Properties* tab select *XSD Schema* from the drop down list. *Properties* tab of the inspector displays the fields relevant to the selected format.

Creating elements, links, and attributes

Select **Element** icon  from the toolbar, create an element and change its name to *personnel*. Create two elements and call them *employee* and *chief executive* respectively. Invoke the Inspector for each element (speedmenu | Properties) and see that the property "content" is set to "empty".

Create a **Reference link**, using the button  , 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 speedmenu, which means that there can by any number of employees. Click on the *chief executive* link and check the box *zero or one*, to specify that there can be one chief executive only.



Add an **Attribute** to each element. Select New | Attribute on the speedmenu of this element, or click the *Attribute* icon on the toolbar and then click inside the element to place the attribute there; change its name to *identifier*. This attribute is a unique identifier of an employee. In order to verify or change the attribute's properties, invoke its speedmenu, select Properties, click the *Properties* tab and set the field *occurs* to *required*, which means that this property is mandatory.

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

- *url* to a file that contains additional data, for example some graphics information.
- *hierarchy* with the attributes *subordinates* and *supervisor*. The attribute *subordinates* should have data type *idrefs*, which means that a person may have a number of people supervised; the field *occurs* of Properties is set to "optional". The attribute *supervisor* should have data type *id*, which means that a person may have one immediate supervisor only; the field *occurs is* set to "optional". Both properties are optional.
- *personal*, with the name and date of birth. The attribute *Date of birth* is mandatory (field *occurs* in Properties is set to "required"), and the field *type* is set to "CDATA".

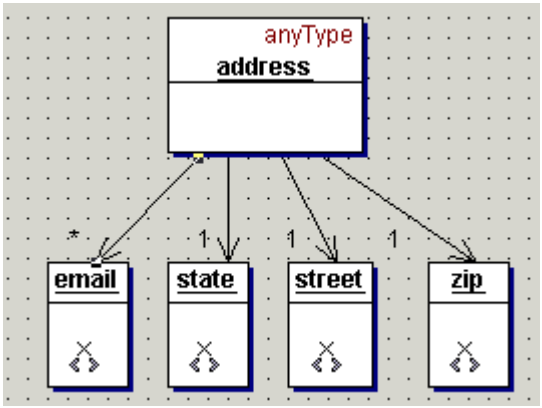
Note: To insert data type of an attribute, in the Properties Inspector click *File chooser* button and select appropriate type in Search/Classpath | XML data types directory.

Invoke speedmenu of each element and see that the property content is set to "empty".

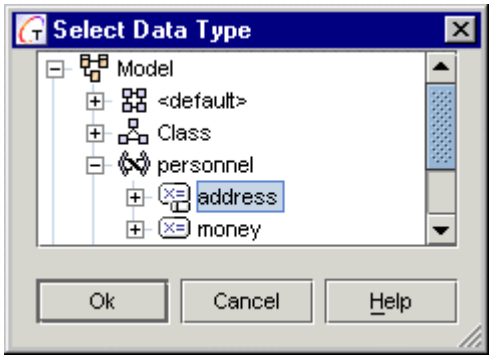
Creating and using complex type and data type

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

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



Now create an element *address*, using  button, invoke its properties inspector, select *base* field in the *Properties* tab and click *File chooser* button. This brings in *Select Data Type* dialog. Expand the *Model* node and select address type under *personnel* diagram:



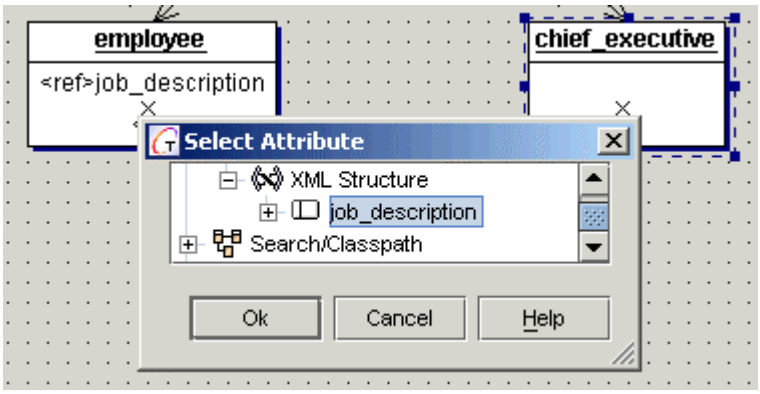
Create *data type* visual element, using  button, and rename it to *money*. In the *Properties* tab of the inspector click *File Chooser* button to select data type from the Model, Search/Classpath or Favorites.

Next, create *salary* element. In the Properties tab of the inspector click *File chooser* button of the *base* field, and select *money* type under *personnel* node of the Model tree. You can use *derivation type* field to select the desired type of inheritance. Each derivation type displays its own set of controls. If you select *restriction*, you can assign minimum salary value.

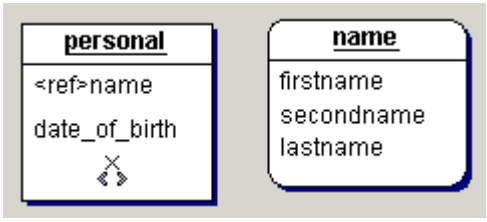
Creating re-usable attributes

XML diagram enables creating Global attributes. Choose *Attribute* button on the diagram toolbar and add *job_description* attribute to the diagram. All elements of the diagram can refer to this attribute.

To add this attribute to the elements *employee* and *chief executive*, right click on the appropriate element and select New | Attribute Reference on the speedmenu. *Select Attribute* popup window displays, where you can select the reference from the model and search/classpath tree. Same way, you can create global attribute *comments,* and add it to any element where needed.



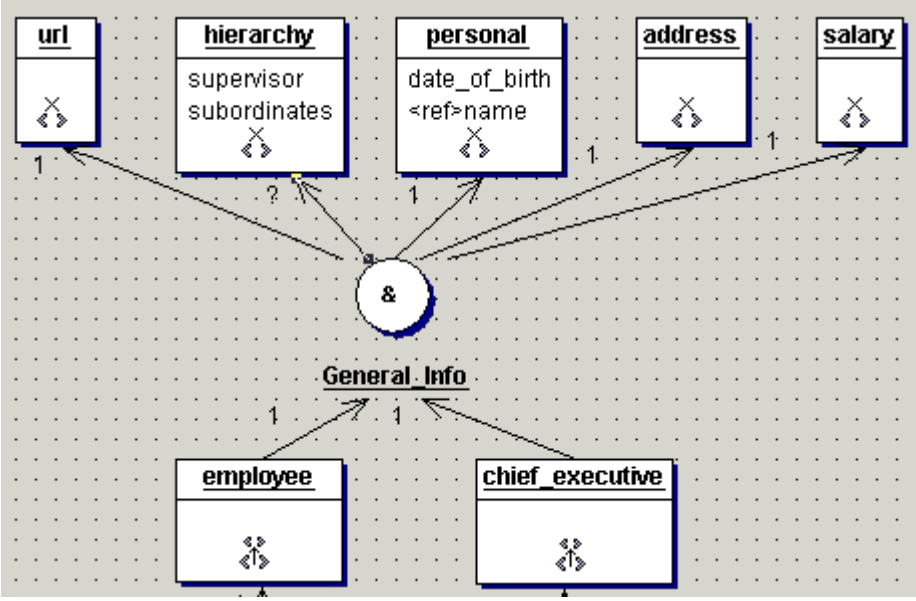
Select *AttributeGroup* button on the toolbar and create global group of attributes *name*. Add the attributes *first name, second name, last name* to this group. All these attributes have String data type. *Firstname* and *lastname* are mandatory (the field *occurs* is set to *required*), while the attribute *second name* is optional (the field *occurs* is set to *optional*). Now right click on the element *personal* to invoke its speedmenu, and select New | Attribute Group Reference to refer to this attribute group in open *Select Attribute Group* dialog.



Creating groups

You can try to create links from the *employee* and *chief executive* elements to each of the elements *hierarchy, url, personal, address, salary*. However, the result looks too entangled. The more elegant way to link multiple elements to multiple elements lays with the Groups. Select the *Group* icon  on the toolbar and create a new group *General_Info*. By default Sequence group is created. Link the *employee* and *chief executive* elements to this group. Invoke the speedmenu of *employee* link, and set the flag *occurence* to "required".

Next, link the group to each of the above elements. By default, the links have the property *zero or one*. Same refers to *url* element. Select the link to *personal* element and in the Properties set the flag *occurence* to "required", which indicates that personal information is unique and mandatory. At last, select the link to the *hierarchy* element and set the flag *zero or one* in its speedmenu, thus indicating that a lucky one can easily do without any subordinates or supervisors.

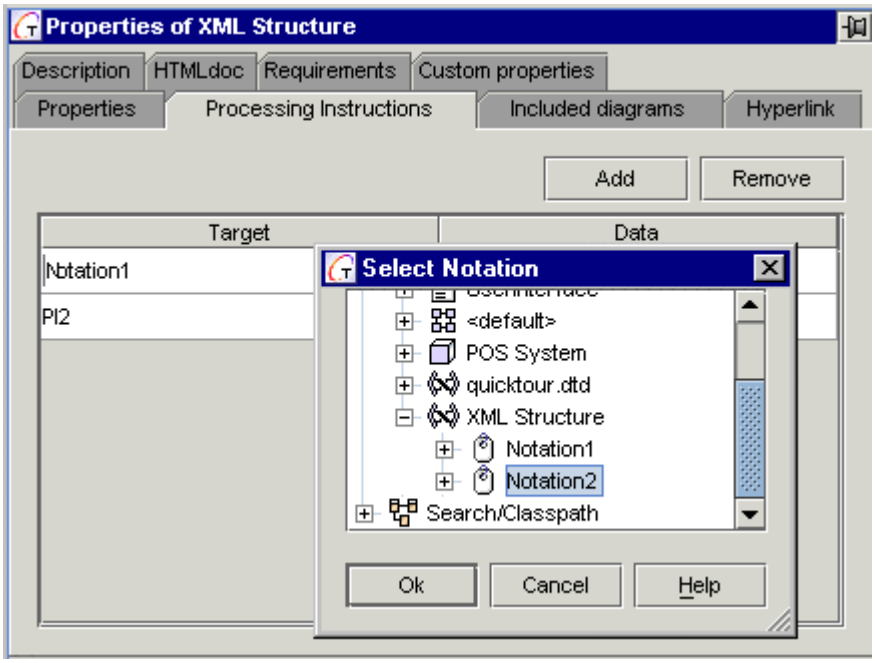


Working with DTD-specific components

Entities, notation and *processing instructions* are specific for DTD format only. Entities and notations are created in a usual way using toolbar buttons. Properties inspector for an entity provides *external* checkbox. When this flag is set, additional controls for external entities show up. In particular, there is a reference to a notation.

To assign notation for an external entity, click *File Chooser* button on the *notation* field of the Properties inspector to display *Select Notation* dialog. You can select the required notation from the Model, Search/Classpath or Favorites.

Processing instructions are defined for the entire XML structure diagram in DTD format. Right-click on the diagram background, select Properties, and on the *Processing instructions* tab press *Add* button. To specify notation for a processing instruction, click *File Chooser* button and select required notation. Enter appropriate information in the *data* field. Press Ctrl+Enter to apply the changes and close the Inspector.



DTD Import-Export

When XML structure diagram is ready, you can perform Export to DTD. Choose Tools | DTD Import-Export | Export command. This displays Export to DTD dialog, where you can specify the target location and invoke the editor to view and modify the file. Once created, DTD file can be imported for further use. On the Tools | Import menu choose Import from DTD option and select the required DTD file from the dialog window. This invokes a standard File chooser dialog, where you can select the desired source DTD file.

XSD Import-Export

It is also possible to export the created XML diagram to a schema file. Choose Tools | XSD Import-Export | Export command. This displays Export to XSD dialog, where you can specify the target location and invoke the editor to view and modify the file. Once created, a schema file can be imported for further use. On the Tools | Import menu choose Import from XSD option and select the source schema file from the dialog window. This invokes a standard File chooser dialog, where you can select the desired source schema file.

Note: When exporting or importing DTD or schema files, keep in mind that some components are only enabled for specific format. In particular, complex type and data type are allowed in XSD format, while entities, notations and processing instructions are allowed in DTD only. Check view formats of the diagram components to avoid loss of information.

See also

Import-Export Operations
Opening diagrams for editing
Creating and opening a project

DTD Interchange

Together supports exchange of information between Class and ER diagrams, and XML diagrams. You can export an existing Class diagram, or database structure to a DTD file, and import information from DTD.

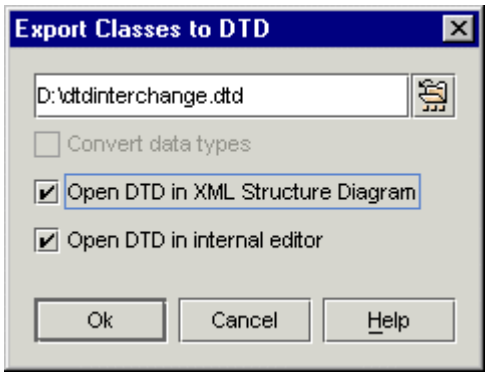
This exchange of information is implemented as an activatable module *DTD Interchange*. Find this module in the list of activatable modules on the Options menu and set the checkbox. This adds *DTD Interchange* node to the Tools menu.

The node contains four commands:

- Export classes to DTD
- Export ER entities to DTD
- Import classes from DTD
- Import ER entities from DTD

Note that commands are only enabled when the diagram pane gets the focus. The set of enabled commands depend on the current diagram. For example in a Class diagram *Export ER entities to DTD* command is disabled, and vice versa.

Export commands invoke *Export* dialogs for appropriate diagrams. The dialogs differ only in title.



In the text area specify target location of the generated dtd file. You may opt to immediately create an XML structure diagram and open the generated dtd file in Together's internal editor.

The reverse operation is to import information from a dtd file to a Class diagram, or to an ER diagram. Import commands invoke File Chooser dialog, where you have to select the source dtd file.

See also

XML Structure diagram

Entity Relationship diagram

Class diagram

XML Editor

Key features of the XML editor

Required information

This XML Editor stands out for it's unique capability to edit XML files while rigorously following the underlying DTD-file specification. Prior to editing XML file, the XML Editor requires certain information from DTD file:

1. List of all allowed XML elements
2. List of all XML attributes declared for each XML element
3. "Rule" specifications for each element.
4. "Cardinality" specifications for each group member in compound element declarations.

For compound XML elements, the Editor supports any grouping rules except those having more than one entry of a child element in the group. For example, grouping rule in the declaration:

```
<!ELEMENT papers ((subject, title) | (authors, title))>
```

will not be correctly supported by the editor, because the element 'title' has two entries in the group. However, equivalent declaration:

```
<!ELEMENT paper ((subject | authors), title)>
```

will be supported without problems.

Creating child elements

When a compound element is selected for editing, the editor analyzes existing child elements. Next, according to the element grouping specification, the editor creates child elements' buttons pane. Each button represents a particular child element and, if enabled, can create one and put it in the appropriate place among the other already existing child elements. Each child element button has 3 states: *required, allowed* and *prohibited*. These states are evaluated according to the grouping/cardinality rule and the current set of children of a certain element, under the assumption that none of them will be deleted:

Required state means that the current set of existing children will never satisfy the grouping/cardinality rule unless a new instance of at least one of the "required" button's child elements is added. The "required" button displays red highlighted.

Allowed state means that a new instance of the child element can be added to the current children set, but is not obligatory to make it eligible for the grouping/cardinality rule. The "allowed" button displays normal.

Prohibited state means that appending of a new instance of the child will make the current children set not eligible for the grouping/cardinality rule. Such button displays disabled.

When a new instance of a child element is created the states of all buttons are recalculated.

Example

Consider a declaration:

```
<!ELEMENT book (subject?, author+, title)>
```

Here an instance of the element 'book' has children *'subject', 'author', 'title'*. Hence, the editor creates 3 child element buttons.

- If the current set of children is void, the buttons' states are as follows:

Button name	State
subject	*allowed*
author	*required*
title	*required*

- If the current set of children is Author1, Title1, the buttons' states are:

Button name	State
subject	*allowed*
author	*allowed*
title	*prohibited*

- If the current set of children is Subject1, Author1, Title1, the buttons' states are:

Button name	State
subject	*prohibited*
author	*allowed*
title	*prohibited*

Location of DTD files

As emphasized in the previous section, to edit an XML-file, XML Editor requires the underlying DTD-file. If DTD file is missing, error is reported.

Opening an XML file, the XML Editor locates it's DTD file, using parameters from the `<!DOCTYPE ...>` directive, unless otherwise is specifically defined. However, one can redirect the search of DTD file, using special `DTDMapping.properties` file, located in the XML Editor package directory:

```
$TGH$\modules\com\togethersoft\modules\xmleditor
```

File `DTDMapping.properties` contains a number of line groups. Each group describes redirection of one DTD file. Each line in the group should start with the same prefix tag. For example:

```
ejb-jar.VendorName = Enterprise JavaBeans 1.1

ejb-jar.PublicID = -//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN

ejb-jar.SystemID = http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd

ejb-jar.Local = ejb-jar_1_1.dtd
```

'ejb-jar.' prefix defines the lines of one group. Values of the keys 'PublicID' and 'SystemID' are used to identify the group by appropriate values taken from <!DOCTYPE ...> directive in the XML file. 'Local' key specifies the local DTD file to be used. This DTD file should be located in the same directory with the file DTDMapping.properties file.

> **Note**: It is important in the current implementation. Do not specify any local file path - this won't work.

Opening an XML file, the editor takes the values of 'PublicID' and 'SystemID' keys from <!DOCTYPE ...> directive. Then, it looks through all groups defined in DTDMapping.properties. If some group has one of 'PublicID' or 'SystemID' keys' values matching with those taken from the XML file, then the local DTD file defined in this group is used. If no matching is found the editor tries to open DTD file by the URL specified as SystemID in <!DOCTYPE ...> of the XML file.

DTD configuration file

DTD-config file is intended to declare additional properties for the XML Elements and the XML Element Attributes, (now associated mainly with the EJB deployment process) cannot be described in DTD file itself.

Each DTD-config file should reside in the same directory with the DTD file it extends. A DTD-config file consists of a number of declarations specified for some XML elements and their attributes. The common form of such declarations for the XML Element and XML Element Attribute looks like:

Element declaration

```
<xml_element_spec>.<modifier> = <modifier_value>
```

Attribute declaration

```
<xml_element_spec>.attr.<attr_name>.<modifier> =
<modifier_value>
```

where

<xml_element_spec> := <xml_element_name> | <xml_element_path>
<xml_element_name> - tag name of the XML Element
<xml_element_path> - XML Element Path (see description below)
<attr_name> - name of the XML Element Attribute
<modifier> - specifies the property of XML element/attribute to be set
<modifier_value> - value of the property

Examples

1. Specifies value type of the Text XML Element 'description'

```
description.valueType = ExtendedStringField
```

2. Specifies that 'id' attribute of the XML Element 'ejb-jar' shouldn't be visible in the XML Editor (modifier: 'hidden')

```
ejb-jar.attr.id.hidden = true
```

XML element path

Declarations based on XML Element tag name, like

```
<xml_element_name>.<modifier> = <modifier_value>
```

specify an XML element's property for all instances of this element in XML Document Tree. However, in some cases it might be needed that the specified property would impact the instances located in the particular places of the XML Tree. only. In the other locations this property should be disabled, or treated differently (i.e. have different 'modifier_value').

In such cases it is possible to define particular location of the XML element instance by it's path in the XML Document Tree. Thus, in the property declaration we can use this path instead of the solely element's tag name.

For example, the declaration

```
ejb-jar.enterprise-beans.session.description.rwiProperty =
$doc
```

sets the value '`$doc`' for the property modifier '`rwiProperty`' of the XML element '`description`', only for those instances that appear in the path '`ejb-jar.enterprise-beans.session.description`' of the appropriate nested XML elements. For the other instances of '`description`' the property '`rwiProperty`' has no definite value.

Such property declarations are dubbed as "XML Element Path Related Declarations" (or just "path related declarations").

Sometimes there are many places in the XML tree where some of the XML element instances (but not all of them) have the same property. In this case, it is awkward to repeat the same declaration for all eligible paths. For example, the element '`description`' must have property '`param.XMLElement`' set to value '`ejb-class`' for those its instances only that belong to the XML element '`session`', no matter where the instance of the element '`session`' itself is located in the XML Tree. For such a case, it is possible to designate unimportant initial segment of the XML Tree path by the asterisk:

```
*.session.description.param.XMLElement = ejb-class
```

If the element '`session`' is the root of XML Document Tree, such declaration will still work.

Consider situation when the two declaration like the above one are in conflict:

```
*.session.description.param.XMLElement = ejb-class
*.enterprise-beans.session.description.param.XMLElement =
ejb-name
```

Both declarations specify the property '`param.XMLElement`' of the element '`description`'. However, the first one says the property should be set to the value '`ejb-class`' as soon as the element '`description`' belongs to the element '`session`'. The second declaration says the property should be set to '`ejb-name`' as soon as, in addition, the element '`session`' belongs to the element '`enterprise-beans`'.

Such contradicting declarations are resolved according to the following rule: *more detailed declaration has higher priority*.

Hence, in the above example, if the instance of '`description`' element is in the path that ends with the elements's chain "`enterprise-beans.session.description`", the valid declaration is the second one. However, if '`session`' element doesn't belong to the element '`enterprise-beans`', the first declaration is in force.

Dynamic modifiers

Normally, the <modifier_value> is specified as a constant string after '=' sign in DTD-config file.

However, it is possible to calculate the modifier value dynamically for each instance of the XML element in the XML Document Tree (i.e for all instances with this modifier specified). To do so the Environment Variables of the Element Instance are used. Each environment variable and its calculation method are declared as a set of several XML element path related properties:

```
<xml_element_path>.env.<var_name> =
<data_provider_class_name>
<xml_element_path>.env.<var_name>.param.<param1> =
<value_of_param1>
<xml_element_path>.env.<var_name>.param.<param2> =
<value_of_param2>
  ...
<xml_element_path>.env.<var_name>.param.<paramN> =
<value_of_paramN>
```

where

<var_name> - name of the variable

<data_provider_class_name> - full name of the class that provides this variable's value. This class should implement the interface ..xmleditor.api.EnvDataProvider. Default implementations of this interface reside in the package ..xmleditor.api. For these classes it is enough to specify class name only, omitting full package name prefix.

<param1> .. <paramN> - parameters specific for the particular Data Provider implementation.

You can access environment variable for any instance using %%env.<var_name>%% macro in the string that specifies the modifier value. In this case the macro is substituted with the variable value.

Example

Suppose we have to adjust the appearance of XML element 'session' in the editor's tree view, showing the entire XML Document Tree.

Let each instance of the XML element 'session' describes an object 'session' of some particular kind. This object's name is stored as a value of an instance of 'ejb-name' element nested in the appropriate element 'session'.

Now assume that we need the name of the 'session' object to appear on the label of appropriate 'session' element in the XML tree view.

This should be declared in the following way:

```
*.session.title = Session: %%env.name%%
*.session.env.name = ValueOfXMLTextNode
*.session.env.name.param.XMLElement = ejb-name
```

where "ValueOfXMLTextNode" is the standard Data Provider that returns the value of XML Text Element - the child of the current element instance (i.e. contained in the sub-tree of the current element) and whose tag-name is specified in the variable's parameter "XMLElement".

List of XML element modifiers

valueType

```
<xml_element>.valueType =
```

This modifier applies for XML Text elements only and specifies which inline editor will be used to edit the value in XML Editor. The modifier is specified for the element tag name (all instances of the XML element).

Allowed values:

StringField // default
ExtendedStringField // multi-line text
BooleanField
IntField
FloatField
FileField
DirectoryField
PasswordField

values

```
<xml_element>.values = "v1", "v2", ...
```

Specifies a predefined list of values for the XML Text element. In that case, the XML editor will create a combo-box for editing the element value. The modifier is specified for the element tag name.

title

```
<xml_element_path>.title =
```

Specifies the label that represents an instance of XML element in the treeview.

disableNew

```
<xml_element_path>.disableNew = true
```

Disables the possibility to create new instances of XML element with the specified path.

Example:

```
*.session.disableNew = true
```

Disables creation of a new instance of the XML element 'session'.

readOnly

```
<xml_element_path>.readOnly = true
```

Prohibits to modify the value of XML element instance with the specified path

type

This modifier allows to connect an instance of the XML element with the particular RWI-element of the current diagram or model. Further, this connection may be used to modify some properties of the RWI-element with the values of this XML element instance, its attributes or its children. (See 'rwiProperty' modifier)

Connection is declared as follows:

```
<xml_element>.type = <rwi_element_provider>

<xml_element>.param.<param1> = <value_of_param1>

...

<xml_element>.param.<paramN> = <value_of_paramN>
```

where

`<rwi_element_provider>` - fully qualified name of the class that provides the RWI-element. This class should implement the interface:
..`xmleditor.api.RwiElementProvider`

`<param1> .. <paramN>` - parameters specific for the particular RWI Element Provider implementation

By default, if connection with the RWI-element is not specified, the instance of the XML element inherits the RWI-element from its parent. RWI-element of the current diagram is assigned by default as the root element of the document tree.

rwiProperty

Specifies for the XML Text element instance that when the XML document is saved the instance's value should be written to the property of the RWI-element connected with this element instance. (See 'type' modifier)

```
<xml_element_path>.rwiProperty = <rwi_property_name>
```

Example

Consider an XML document where each instance of XML element '`entity`' represents a class of the current diagram. Its child element '`ejb-class`' contains the SCI qualified name of the class, and its child element '`description`' contains the class documentation. We need the '`description`', modified within XML editor, to be written in the model when the XML document is saved. Then, the required declarations in DTD-config file should be:

```
entity.type =
com.togethersoft.modules.ejbframework.xmleditor.RwiClassByXM
LTextNode

entity.param.XMLElement = ejb-class

*.entity.description.rwiProperty = $doc
```

where

RwiClassByXMLTextNode is the implementation of RWIElementProvider that returns the RWI-element of the class by its SCI qualified name taken from the value of the child XML element, whose tag-name is specified in the provider's parameter 'XMLElement'

`$doc` - name of the RWI-property of the class RWI-element storing the documentation

List of XML attribute modifiers

valueType

```
<xml_element>.attr.<attr_name>.valueType =
```

Same as for XML Element.

values

```
<xml_element>.attr.<attr_name>.values = "v1", "v2", ...
```

Same as for XML Element.

title

```
<xml_element_path>.attr.<attr_name>.title =
```

Same as for XML Element.

readOnly

```
<xml_element_path>.attr.<attr_name>.readOnly = true
```

Same as for XML Element.

rwiProperty

```
<xml_element_path>.attr.<attr_name>.rwiProperty =
<rwi_property_name>
```

Same as for XML Element. Specifies that the attribute's value should be written to the property of the RWI-element connected with the XML element instance owning this attribute. The connection of XML element with the RWI-element should be specified by 'type' modifier for this XML element.

hidden

```
<xml_element_path>.attr.<attr_name>.hidden = true
```

Specifies that the given attribute of the XML Element instance does not show up in the XML Editor attributes' inspector window.

Using XML Editor

XML Editor launches from Together **Tools** menu. First, make sure the XML Editor module is activated: go to Options | Activatable Modules | XML Editor menu item and make sure it is checked. Next, select Tools | XML Editor (it displays if XML Editor module is active). XML Editor dialog window shows up.

You can load an existing XML file or create a new one. In both cases appropriate DTD file should exist and be specified to the editor. When opening an existing XML file, you can specify physical location of DTD files in the file "DTDMapping.properties" (see Location of DTD-files for more details).

If you create a new XML-file, you are asked to choose the appropriate DTD-file and XML element to be the root of a new XML document.

Use *Save* button to save the results.

Example

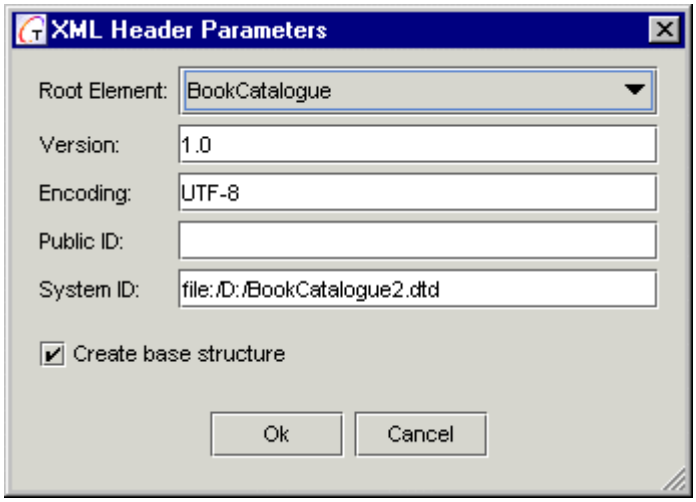
Consider creating an XML file based on the existing DTD file.

Create the following dtd file, and save it with the name `BookCatalogue.dtd` to your disk:

```
<!-- A book catalogue contains zero or more books -->
 <!ELEMENT BookCatalogue (Book)*>
 <!-- A Book has a Title, one or more Authors, a Date, an ISBN, and
a Publisher -->
 <!ELEMENT Book (Title, Author+, Date, ISBN, Publisher)>
 <!-- A Book has three attributes - Category, InStock, and
Reviewer. Category must be either "autobiography", "non-fiction",
 or "fiction". A value must be supplied for this attribute whenever
a Book element is used within a document.
 InStock can be either "yes" or "no". If no value is supplied it
defaults to "no". Reviewer contains the name of the
 reviewer. It defaults to "" if no value is supplied -->
 <!ATTLIST Book
   Category (autobiography | non-fiction | fiction) #REQUIRED
   InStock (yes | no) "no"
   Reviewer CDATA "">
 <!ELEMENT Title (#PCDATA)>
 <!ELEMENT Author (#PCDATA)>
 <!-- A Date may have a Month. It must have a Year. -->
 <!ELEMENT Date (Month?, Year)>
 <!ELEMENT ISBN (#PCDATA)>
 <!ELEMENT Publisher (#PCDATA)>
 <!ELEMENT Month (#PCDATA)>
 <!ELEMENT Year (#PCDATA)>
```

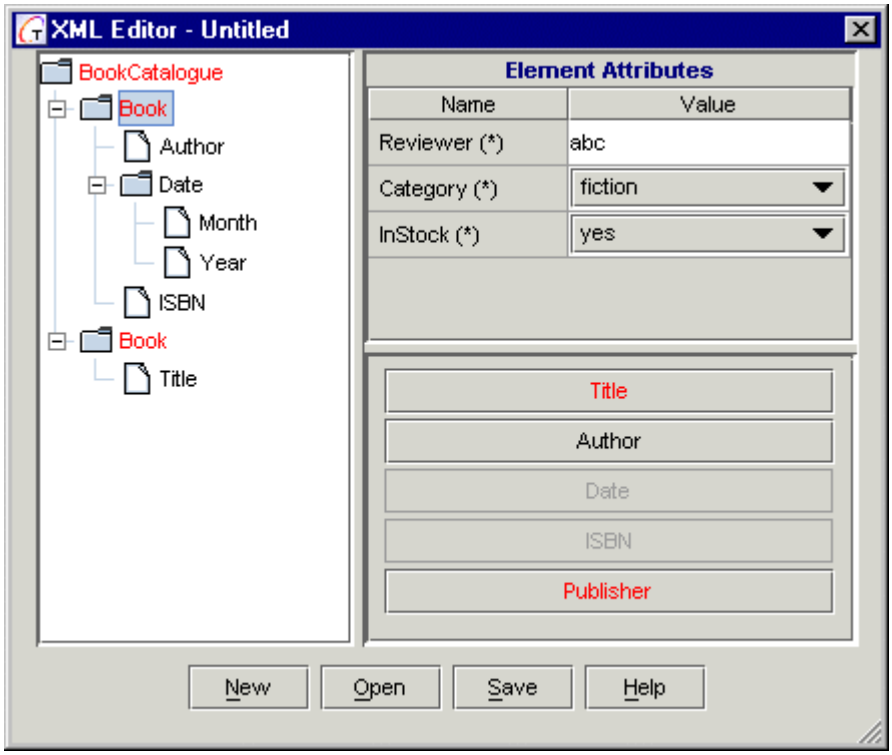
Invoke XML Editor module on the Tools menu and click *New* to create an XML file. Select `BookCatalogue2.dtd` file in the treeview of the *Choose DTD File* dialog.

Next, specify the root element for the xml file, selecting it from the dropdown list, and set the flag *Create basic structure*:



The Editor creates an entry for the root element *BookCatalogue* in the structure pane, and a button *Book* for the child element. Pressing this button adds the Book element to the resulting xml document, displays the attributes defined in the ATTLIST section of the underlying DTD file, and adds buttons for the child elements.

While adding the instances of the child elements, observe the changes of the button states. When the current set of child elements is void, the buttons are in the *required* state (red highlight). If the cardinality rule for a certain element doesn't allow creating new instances, the button becomes *prohibited* (in our case this element is Title, since a book may not have more than one title). The buttons in the *allowed* state enable creating more entries (a book may have several authors).



When the process is complete, you can save the resulting XML file, pressing Save button.

Launching XML editor from Java code

You can start XML Editor from you Java code, using XML Editor API class

`com.togethersoft.modules.xmleditor.api.XmlEditorAccess`

Standard XML Editor window is invoked by `showEditor()` method.

You can launch XML Editor for editing the only XML file you need, using boolean showEditor method with the name of the required XML file as a parameter. In this case the editor opens with the specified XML file and enables Save and Cancel buttons only, so that no other file can be opened from the editor window.

Method returns '`true`' if the XML-file was modified. Exception throws when the file cannot be parsed as XML file, no DTD file found, DTD file is incorrect or doesn't correspond to the XML file. Appropriate error message is passed to the Exception object and displayed in the Together's Message Pane.

By default, XML file opens same way as if selected from the File Chooser dialog. However, you can specify the following fields for the XML file:

- DTD file used to edit this XML-file
- DTD-config file (not necessarily located in the same folder with DTD)
- XML file prolog to be used when the file is saved
- Encoding used in this XML file

These parameters are set in the appropriate methods of `XmlEditorAccess` class. All parameters should be set before method `showEditor` is called.

Example

```
import
com.togethersoft.modules.xmleditor.api.XmlEditorAccess;
...
XmlEditorAccess editorAccess = new XmlEditorAccess();
editorAccess.setDtdFileName (myDtdFileName);
editorAccess.setXmlProlog (myXmlProlog);
editorAccess.setEncoding (myEncoding);
try
 {
 editorAccess.showEditor (myXmlFileName);
 }
 catch (Exception e)
 {
 System.out.println (e.getMessage());
 }
```

Enterprise SW Development Features

Various Language Support

Java is Together's originally supported language. Eventually, Together extended support to IDL, C++, C#, Visual Basic and VBNet. Working with languages other than Java, engenders certain limitations related to specific features of those languages. These limitations stem from the lack of deep parsing, and non-object-oriented nature of some languages. Special chapter is devoted to C++. This section provides summary information on the features for the supported languages, and brief notes about language-specific properties.

	Java	CORBA IDL	C++	VB	C#	VBNet
Basic functionality	+	+	+	+	early access	early access
Deep parsing	+	n/a	+	-	-	-
Default Templates	+	+	+	+	+	+
Additional textual templates	+	-	+	+	+	+
Code-based patterns	+	-	+	-	+	-
Properties	+	-	+	-	+	-
Events	+	-	-	-	+	-
Syntax highlight	+	+	+	+	+	+
Formatter	+	+	+	+	early access	early access
Metrics	Full set of metrics	not supported	limited set of metrics	limited set of metrics	limited set of metrics	limited set of metrics
Audits	Full set of audits	not supported	limited set of audits	not supported	not supported	not supported
Search for usages	+	+	+	in declarations only	in declarations only	in declarations only
Class diagram toolbar (language-specific components)	Entity, Session and Message Driven Beans	ValueType, Exception	Aggregation	common	common	common
Doc generation	+	+	+	+	+	+

Languages Support

Language support is customizable. To disable an unnecessary language, comment out appropriate line in the `$TGH$/config/languages.config` file. Inspector options are described in the `*.config` files for each language. Recognition of the language-specific properties is defined in the *View Management* page of the *Options* dialog.

SCI Implementation

Basic functionality

For the supported languages Together provides parsing of the syntactical constructs that map directly to UML objects (classes, interfaces, methods etc.). As of this writing, C# and VBNet are in the early access state.

Deep parsing

This term is used to describe functionality that handles syntactical constructs within the method bodies, initialization of variables etc. Deep parsing is only supported for Java and C++.

Re-use support

Default templates

Textual templates, whose names start with *Default_,* create one-click elements on diagrams. These types of templates are available for all supported languages. You can see the list of currently available default templates in the *Templates* folder of the *Directory* tab. Default templates do not show up in the Pattern Chooser.

Additional textual templates

Additional templates are invoked from the Choose Pattern dialog. You can observe the up-to-date list of available templates on the Pattern Chooser panel, or in the *Templates* folder of the *Directory* tab. The textual templates in the Pattern Chooser are marked with an asterisk, to distinguish them from the code-based patterns. The users can create their own custom code templates.

Code-based patterns

Together patterns are public Java classes that implement `SciPattern` interface. You can observe the up-to-date list of available patterns on the Pattern Chooser panel. Those patterns that are inapplicable to a certain language, are grayed out in the list. *Minus* sign in the table entry means that no ready patterns are currently available for this language. However, the users are free to develop the patterns of their own.

Properties support

The properties feature originates from the Java Bean properties, where a property is represented by a triad of the property itself, and its getter and setter methods. Subsequently, when properties support was extended to the other languages, similar rules were accepted. Thus, an additional tab adds to the inspector: *Bean* tab for Java, *C++ properties* tab for C++, *Properties and Events* tab for Visual Basic. To recognize the properties, select appropriate option in the *View Management* node of the *Options* dialog.

Refer to Inspector Customization section, and Open API for the list of properties, supported by the model.

Language-specific inspector

As you already know, choosing language for the project means that any component created on the Class diagram, will correspond to the selected language. However, a project may contain components in different languages. Dynamic inspector tabset complies with the origin of each component.

VB6-specific inspector

Properties and Events tab

This tab adds to the VB6 class inspector when appropriate option is selected in the *View Management* page of the *Options* dialog.

Class / Interface

Class and interface elements have only the properties common for all languages.

Operation

return value: Byte, Boolean, Integer, Long, Currency, Decimal, Single, Double, Date, String, Object, Variant
visibility: public, private, friend
static : Boolean property adds *static* modifier to an operation.

Attribute

type: Byte, Boolean, Integer, Long, Currency, Decimal, Single, Double, Date, String, Object, Variant
visibility: public, private
const: Boolean property adds *const* modifier to an attribute.

C#-specific inspector

Class

namespace: text area for the class namespace
abstract: Boolean property adds *abstract* modifier to the class. If the class is sealed, this field is disabled.
sealed: Boolean property adds *sealed* modifier to the class. If the class is abstract, this field is disabled.
extends: text area and file chooser button for the base class name.
implements: Multi-string field for the list of implemented interfaces.
visibility: public, protected internal, protected, internal, private (Visible for inner classes only).
new: Boolean property adds *new* modifier to a class. (Visible for inner classes only).

Interface

namespace: text area for the class namespace
extends: text area and file chooser button for the list of inherited interfaces.
visibility: public, protected internal, protected, internal, private. (Visible for inner interfaces only).
new: Boolean property adds *new* modifier for class. (Visible for inner interfaces only).

Operation

return type: bool, byte, char, decimal, double, float, int, long, object, sbyte, short, string, uint, ulong, ushort, void
new: Boolean property adds *new* modifier to an operation.
visibility: public, protected internal, protected, internal, private
static: Boolean property adds *static* modifier to an operation. Disabled for virtual, abstract and override operations.
virtual: Boolean property adds *virtual* modifier to an operation. Disabled for static operations.
override: Boolean property adds *override* modifier to an operation. *New, static* and *virtual* modifiers are disabled if this property is selected.
abstract: Boolean property adds *abstract* modifier to an operation. *Static* and *virtual* modifiers are disabled if this property is selected.
extern: Boolean property adds *extern* modifier to an operation. *Abstract* modifiers are disabled if the property is selected.

Attribute

type:bool, byte, char, decimal, double, float, int, long, object, sbyte, short, string, uint, ulong, ushort, void
new: Boolean property adds *new* modifier to an attribute.
visibility: public, protected internal, protected, internal, private
static: Boolean property adds *static* modifier to an attribute.
readonly: Boolean property adds *readonly* modifier to an attribute

IDL-specific inspector

Links

In addition to the common *client* and *supplier* fields, the following fields are added:
label: text area to enter the label of the link
label direction: drop-down list of possible directions of the label (default, forward, reverse)
association class:
client role / supplier role: two text areas to specify the appropriate roles
client cardinality / supplier cardinality: drop-down list of possible cardinality values (zero or one, required, zero or more, one or more)
client qualifier / supplier qualifier: two text areas to specify the appropriate qualifiers
directed: drop-down list of the link directions. (Automatic, directed, undirected. Defaults to undirected)
type: drop-down list of the link types (association, aggregation, composition)

C++-specific inspector

C++ properties tab

Recognition of C++ properties is defined in the *View Management* page of the *Options* dialog. If this option is selected, *C++ Properties* tab adds to the object inspector of a cpp class.

Links

In addition to the common *client* and *supplier* fields, two more fields are added:
visibility: public, protected, private
virtual: Boolean property adds *virtual* modifier to the link.

Using Together with C++

Important Notes for C++ Support

Together is a file-based product destined for round-trip engineering of source code, and as such concentrates effort on retrieving and keeping in sync the fragments of source code that directly map to the object modeling entities. Using Together with C++ engenders a number of complexities and limitations of functionality that stem from specific characters of C++ language semantics.

For example, fully qualified class names in C++ don't correspond to the actual physical locations of the classes. It implies that to retrieve a class by name, all known classes need to be visible right at project open time, which implies that all available files should be processed. This leads to increased memory demand for C++ projects. It becomes crucial assuming that development involves wide usage of libraries, which provide sets of headers. The size of these libraries is quite significant (for example, MFC, VCL, OWL), while the real projects make use of just a small portion of the libraries.

Another issue lies with the fact that C++ makes use of a preprocessor. Usage of a preprocessor falls into two parts.

The first one concerns the usage of macro definitions. Contents of the object model significantly depends on the results of resolving macros. Besides that, the macros are used in course of conditional compilation, and thus define the portions of the source code that may or may not appear in the object model.

Each encountered macro, defined by the preprocessor directive, is considered global, and falls to the global table. Thus C++ semantics, which assumes the use of macro definitions in the context of compilation unit, is violated. However, the experience of using Together shows that in absolute majority of projects this violation is not critical and can be avoided by proper tuning of the project structure and configuration options

The second issue concerns the usage of #include directives. Together project is folder oriented, and thus Together processes all files, both headers (*.h) and sources (*.cpp), it finds in folders pertaining to the project. Generally, this is done in an arbitrary order. If Together encounters an #include directive while processing a file, it switches to processing of that file. Each file is processed only once.

Considering the problems described above, the task of creating and configuring C++ projects, especially the ones already existing, becomes especially creative. The project must achieve a proper balance between the desired completeness of the model at run-time and resources used.

These problems can be solved by proper organization of project structure and usage of macro-definitions.

Project management

Defining Project Structure

Together provides various means of handling files included in the *Project Path* and in *Search/Classpath*. The former are parsed anyway, while the latter are only parsed if `#include'd` in one of the files under the *project path* (maybe, through a chain of `#include` directives).

Search Path in Large Projects

In the large-scale projects, it is advisable to add to the *search path* the sources, that can be considered external, or standard, with respect to the entire project structure. This helps restrict the number of files handled within a project.

Internal and External #include Directives

Together makes use of a well-known technique to discriminate the project internal and external headers. The external files are included by means of `#include< >` directive with angles, while the internal files are included via `#include" "` with quotes.

Skip Standard Includes

It often happens that external `#include` directives contain information useless for object modeling. Hence, the external files can be skipped, which is done by default in Together: *Options* dialog provides *Skip standard includes* flag in C++ node of the *Source Code* tab. This flag is checked by default for the sake of memory saving.

Using preinclude file

In order to address problems described above, *Together* introduces technique of preinclude file. The idea is that there is a special file preinclude.inc that is processed before any other C++ file available within the project. Preinclude file is treated by *Together* almost same way as other C++ files. Namely it may contains #include directives as well as macro definitions. The difference is that Together ignores symbol declarations in this file.

The global preinclude file is delivered with *Together* and resides in the `/lib` folder of your installation. As well it is possible to create project local preinclude file (`preinclude.inc`) in the project root (for example using Together Editor). *Together* merges information from both global and project local preinclude files.

Managing includes

The possibility to add #include directives into preinclude file allows to specify all the necessary files from the required external library. This is done by means of #include " " directives (with quotes). This form provides independence from the Skip Standard Includes option. All files thus included will be processed (if found relatively to search path specified), and the classes declared in those files become available even from new empty project (see "Add shortcut...;").

> **Note:** Definitions of the class members are stored in physically different locations, other than class files. It implies that Together binds definitions and declarations of members in the classes by signature. To provide seamless binding, the definition files *must* be visible in the project and properly parsed. Hence, `preinclude.inc` should contain `#include` directives for the definition files.

By means of #include directives in preinclude file it is also possible to strictly define the order of processing file when it is critical for correct processing of project sources or sources of external libraries.

You can see the example of this technique in the preinclude files delivered with Together. These files configure the usage of Standard Template Library (`%TGH%\lib\pi_*.inc` for MS Visual C++ and Borland C++).

C++ Macro Definitions

Preinclude.inc file is also used for another purpose. When the project source code is parsed, all macro definitions, required for proper parsing, should be already known in advance, regardless of whether the containing files were processed or not. Note that macros can be defined either in various external libraries, or in internal company headers that, although external to the project structure, are still part of the source code that *Together* should process into diagrams. If these library headers are not included in the *Together* project resources (and thus are not parsed), *Together* cannot perform the substitution of the macros encountered in course of parsing. This leads either to error messages, or to some desired visual information not showing up from the resulting diagrams.

The latter occurs if some constructs that are part of the object model, have been defined with macros - class member declarations for example. If no substitution is defined for it, *Together* cannot properly recognize such constructs and thus cannot properly display them in diagrams.

On the other hand, including all library sources into a *Together* project is not advisable because it makes the project overly large and rather clumsy, and usually leads to dramatic decrease in parsing speed. To address this problem, `preinclude.inc` is provided so that you can more narrowly define what macro substitutions should be used by *Together* when parsing the project.

There are different approaches to writing definitions. The first way is to use macro definitions exactly as they are used in the library. However, this is often unnecessary because the original definitions can be quite complex and yet make no sense for *Together*. Remember that your real goal is to make Together to parse the sources without error messages, and display all desired information on the diagram.

It means that macros can often be defined just to nothing if the corresponding constructs are not supposed to have any visual feedback on the diagram. *Together* simply skips such macro declarations.

Where you expect to get visual feedback from a macro declaration, you have to be more accurate and define a non-empty substitution. You can use the full library definition if that's what you need to see, or you can simplify the definition to bring in only the item(s) you want displayed in the diagrams. Note that any substitution to these definitions made for Together in `preinclude.inc` does not affect the compilation of your sources in any way. Your compiler will use the real library headers.

To illustrate the above, refer to the macro sets provided in the default `preinclude.inc` file for MFC which are controllable via wrapper macro
TS_PREINCLUDE_MICROSOFT_VISUAL_C:

MFC_MACROS_SHOW_EXPANDED (complete as in MFC)
MFC_MACROS_SHOW (show as functions)
Otherwise empty

You can refer to the System Macros section for the list of standard macros that *Together* provides for C++ projects.

Default library support

The default `preinclude.inc` addresses the problems that can occur on parsing a number of libraries commonly used by C++ developers. Sets of defines are provided for:

Microsoft Windows
Microsoft C++
Microsoft Foundation Classes (MFC) library
COM
ATL
Borland C++ version 5.0
Borland C++ Builder

There are additional wrapper macros that are used for controlling which of these sets of defines are "enabled" currently via #ifdef. These are:

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

(See the comments in the `preinclude.inc` file)

It is of course impossible to anticipate which of the above you actually use. Neither is it possible to include something for every conceivable library, or anything at all for your own company standard headers. Therefore, some resetting of C++ configuration options and some hand-tuning of the `preinclude.inc` will most likely be necessary before you begin using Together for C++ projects.

Setting wrapper macro options for default libraries

Note that some "widely used libraries" may use macros with identical names but different definitions. For example, MFC and OWL both define "BEGIN_MESSAGE_MAP" in completely different ways and that will cause problems with parsing.

Before using Together for C++ projects, make sure that the wrapper macros for the libraries you use are defined in your C++ configuration options, and those for the libraries you don't use are "un-defined". If you don't do this, some macro names will be duplicated but have different definitions, which will cause Together to display numerous error messages while parsing C++ source code files.

To define or un-define wrapper macros:

1. Choose Options | Default to display the Default Options dialog
2. Select the Source Code page
3. Expand the C++ node on the options tree and click Define.
4. Remove the names of wrapper macros for libraries you don't use from the value string.
5. If the name of the wrapper macro for a library you do use is not present in the value string, add it to the string. Observe the all-upper case naming convention and be sure to delimit all the macro names with a semi-colon (;).
6. Click OK to close the dialog.

Note that the above configuration can be done at the Default or the Project level. See the Configuring Together topic for information about multi-level configuration.

Configuration issues

Entries in ".config" files

Configuring C++ source and header file definitions

The file %TGH%/config/resource.config (version 3.x and above) contains definitions of file types for Together and sets of extensions corresponding to the file types.
For C++ sources and headers the following definitions are specified by default:

```
# -- C++ --
 resource.file.cpp_source.extension.1 = "cpp"
 resource.file.cpp_source.extension.2 = "c"
 resource.file.cpp_source.extension.3 = "CPP"
 resource.file.cpp_source.extension.4 = "C"
 ...

 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, corresponding files will also be recognized as C++ sources and headers.

Specifying codegen file extensions

The default file extensions for generated header and source files are declared in the following properties in `%TGH%/config/cpp.config:`

```
codegen.cpp.declaration_file_ext = "h"
codegen.cpp.definition_file_ext = "cpp"
```

To generate files with different extensions, you need change the value for these properties. You can do this in the Options dialog: choose Options | [level] | Source Code | C++ and edit the options labeled Extension of generated C++ header file and Extension of C++ definition file respectively.

External Tools

Using a C++ compiler

Together provides no default C++ compiler. Set up a compiler at your choice, following same procedure as described in configuration sections (Configuring other Java compiler/make utilities).

Configuring C++ compile/make utilities

Configuring can be done from Together Options Dialog (use Options | Default. menu command to display it) or Project Options Dialog (Options | Project. menu command). After opening the dialog expand Tools node. There you can see the treeview of available tools. Select the one you are going to configure and enter the required settings.
Select C++ from the drop-down list in *Language* field, specify command for the executed tool, command line parameters, filtering for the tool's output, menu settings - where configured tools should be displayed or not, etc.

Tips:

- Do not overwrite default compiler/make settings; use empty "Tool #x" slots instead.
- For proper setup use on-the-fly help, displayed in the Description window of the Options Dialog.
- Pre-defined command names in the Options Dialog display like ["toolEditor/Compiler"]. To create the desired name, use just a text, without '[' and ']' characters. For example, you can name the compile command *C++ Compiler*.

Executing C++ compile/make utilities

When configuring the tools, you have to specify appropriate menu command names and where these commands should be displayed. So you should already know how to execute compile and make utilities.

Known Problems

C++ support in Together has some limitations that impact the usage of certain features. You can contact support group for more details and workarounds. The following sections outline problem issues:

Deep parsing

This feature provides handling of statements and variables in the methods' bodies and initializers. With C++, the usage of this feature affects the completeness of results in Update Package Dependencies, Autodependencies between classes, Show dependencies, Add Linked, Find Usages and Generate Sequence Diagrams.

Namely:

1. Expressions in the initializers of attributes and variables are not processed.
2. The operations are matched to the operation calls by number of parameters, rather than by types (types are ignored).

Class usages

Classes used as the arguments in templates, are not handled. For example:

```
class A{
SomeTemplate<FirstClass, SecondClass>pSome;
}
```

In this situation the occurrences of FirstClass and SecondClass are ignored. This affects the completeness of results in Update Package Dependencies, Autodependencies between classes, Show dependencies, Add Linked, and Find Usages. These calls are not renamed, when the referenced class is renamed. The usages from template arguments are handled in read-only mode.

This issue will be addressed in the next versions.

Recognizing Member definitions

To be recognized, member definitions should have exactly same signature as the member declarations.

For example:

--- header ---

```
class A {
void op( std::string str );
}
```

--- source ---

```
using namespace std;
void A::op( string str ) {} // this definition will not be recognized
```

This mean in particular:

- Shift+click on a class member on diagram doesn't navigate to the source
- definition file property in the inspector is empty
- editing of operation 'op' declaration doesn't affect definition in the source

In order to avoid this use either short names in both places ("string"), or full names in both places ("std::string")

Processing of enumerations, typedef's and global symbols

As of this writing, these elements are not processed. This feature is under construction for the future releases.

See also

Important note for C++ users in Find Usages dialog

DefDocComments

DefDocComments module provides access to the doc comments of c++ member definitions. The status of this module is Early Access. It is compatible with Together versions 4.x and higher.

Installation

If you have Together 4.2 and higher, you can skip this section.

To install the module, place defDocComments subfolder under `%TGH%\modules\com\togethersoft\modules`. If you've unpacked archive at `modules\com\togethersoft\modules`, this is already done. Otherwise copy this folder to the specified location. Upon restart of Together, command "c++ definition doc-comments" appears in Options | Activatable modules. Same command is available under Early Access node on the Modules tab of the Explorer.

Usage

Make sure that the module defDocComments is activated. If this module is not activated, check the box "c++ definition doc-comments" on the menu Options | Activatable Modules.

Being activated, the module adds Definition documentation tab to the object inspector (Properties...) for c++ members with definitions. This page allows to view/edit documentation of member definition in the same way as used for Description and Javadoc tabs.

The module registers support for new boolean property "DefDocComment" that represent definition doc-comment. Instance of this property has subproperties representing tags from doc-comment. Description part (text preceding any tag) is available as subproperty with a null name.

Access through API

```
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");
```

Generate documentation issues

Documentation of member definitions is available also for Documentation Generation feature. In order to make use of this possibility, one should change metamodel settings of GenDoc module. Edit the file `%TGH%\modules\com\togethersoft\modules\gendoc\templates\MetaModel.mm`

Add line

```
DefDocComment = "Definition doc-comment", boolean;
```

to the end of first "# RWI specific properties" section.

This property must be also added to the lists of properties for metatypes GENERIC_OPERATION and ATTRIBUTE. Search for occurrences of strings GENERIC_OPERATION and ATTRIBUTE respectively, and add entry DefDocComment; to the list of metatype properties.

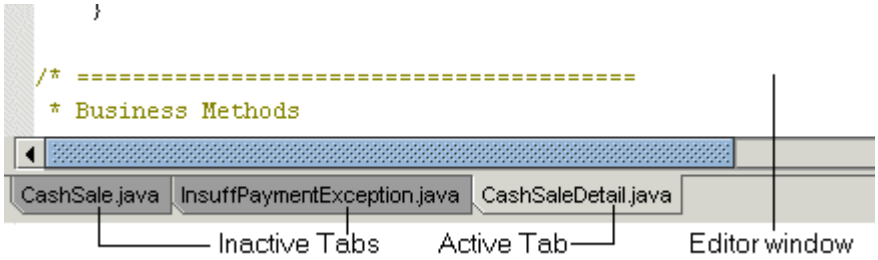
These changes let gendoc recognize "DefDocComment" property as the standard property of elements.

Modified gendoc `ClassReport.tpl` template supplied with the module (renamed to `ClassReportDefDoc.tpl`) can be used as a sample of new feature usage in documentation. There is a new "Definition documentation" stock section in the template that demonstrates usage of the new feature. This section is called from Operation description section. Try to generate documentation for a sample project supplied with this module, using both modified template and its original version, and compare results.

Editing

Using the Editor

Together comes with a multi-page built-in code/text editor with a set of standard and extended features. You can switch between tabs by clicking on tab's header, located right under the editor.



Standard features

These commands are available from the main Edit menu, or the speedmenu of Editor. You can use hot-key combinations to speed up access to a particular command.

Edit menu commands and hot keys

Command	Function	Hot-key
Undo	Rolls back last several changes.	Ctrl-Z
Redo	Reinstates last Undo operations.	Ctrl-Y
Cut *	Cuts selected text to Clipboard.	Ctrl-X
Copy *	Copies selected text to Clipboard.	Ctrl-C
Paste *	Pastes text from Clipboard at the cursor position.	Ctrl-V
Delete	Deletes selected text. (Restore with Undo)	Del
Select All	Selects all the text in the currently focused file	Ctrl-A
Insert Text From File	Displays standard "Open File" dialog to pick up the file containing the text to be inserted.	(none)
Go To Line	Displays dialog to specify a line number, then jumps to the specified line number	Ctrl-G

* *(Note:* Clipboard commands also work with diagram elements in the diagram pane.)

Editor speedmenu commands

The Editor speedmenu can be invoked from the Editor pane and from a tab of each tabbed page in the Editor pane. Both speedmenus contain a number of commands, some of which are self-explanatory or highly intuitive (Cut, Copy, Paste, Select All), while others require some brief explanation. The detailed information about the features listed below can be found in the relevant sections under Enterprise Software Development chapter.

Command	Description
Select on Diagram	Selects the visual class in the default Class diagram for the class currently being edited. Opens the diagram if it is not already open in the Diagram pane.
Preserve Tab	Appears on the page tab speedmenu. This allows you to prevent the contents of active tab from being replaced with another file when you use the *Open* command. When you select this command, another *Untitled* tab is opened and then you can safely use *Open* menu command
Format Source	Performs syntax formatting of the source code in active tab
Tools	Contains a sub-menu with a list of tools you can apply to active tab. You can configure tools using "Tools" tab of *Together* Options Dialog
Text Editor Options	Invokes the Test Editor page of the Options Dialog.
Version Control	Contains a sub-menu with a list of version control commands.
Bookmarks	Contains a sub-menu with a list of bookmarks commands.
Refactoring	Contains a sub-menu with a list of refactoring commands

Commands for the Integrated Debugger

The Editor interacts with the Integrated Debugger by means of several speedmenu commands:

Command	Description
Toggle breakpoint (F5)	Toggles breakpoint to the current line of source code
Disable/Enable breakpoint	Disables or restores a breakpoint.*
Breakpoint properties	Allows you to edit breakpoint's properties.*
	*(Hidden when no breakpoints are set in the file)

Extended features

The Editor provides the following advanced features:

Rectangular blocks (Ctrl-L): Gives you the capability to work with rectangular blocks. Ctrl-L key combination toggles rectangular block capabilities on and off. Any time you need to work with a rectangular block, press Ctrl-L, then select the desired block of text with your mouse, or by holding down the Shift key and using arrow keys. When rectangular block capabilities are no longer needed, press Ctrl-L again to switch to normal block mode.

User defined snippets (Ctrl-J): You can define as many snippets as you wish and then use them easily in your code by typing the snippet name and then pressing Ctrl-J. See Defining Snippets in this chapter.

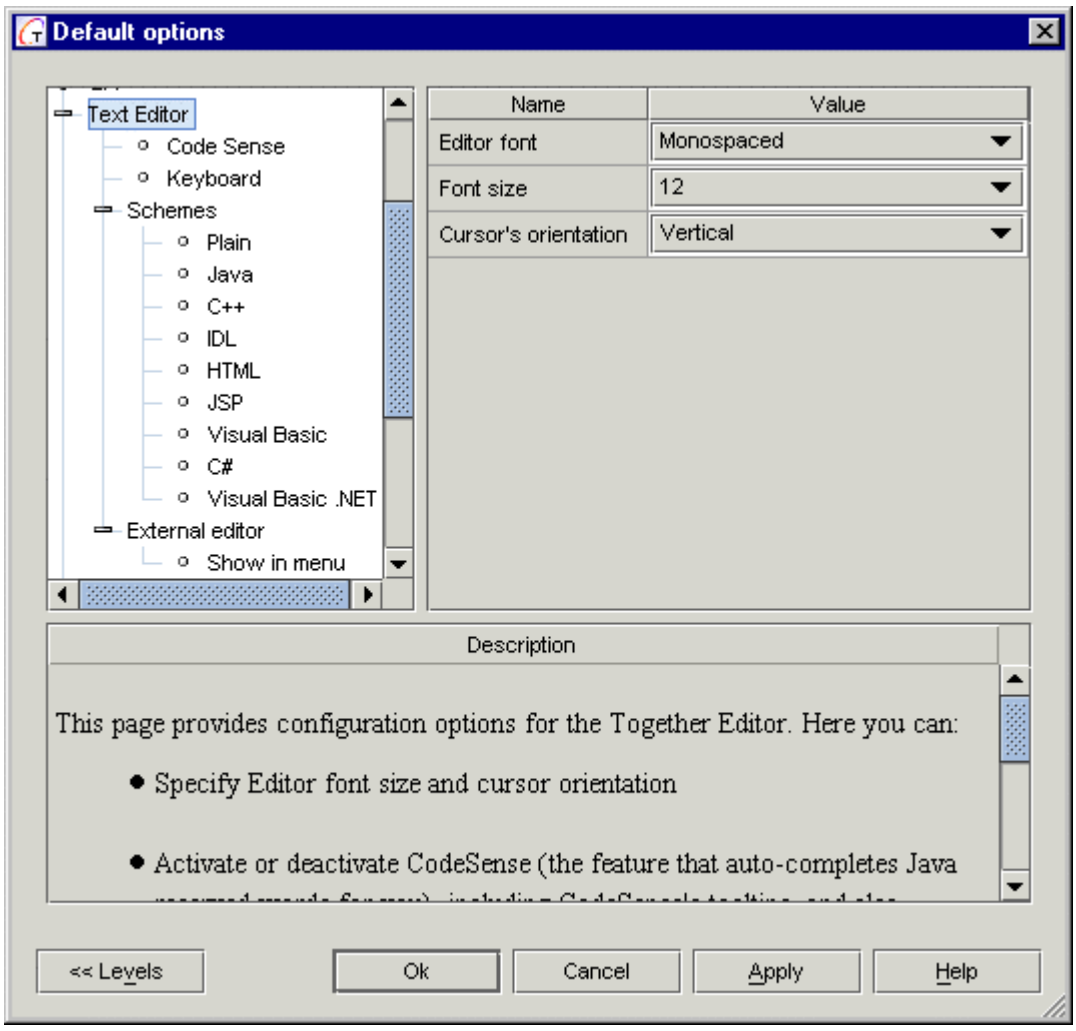
Code Sense (Ctrl-Space): This allows you to auto-complete methods of standard Java classes. See Using Code Sense in this chapter.
Browse symbol allows to pass on to the source code of a class, attribute or operation.
Bookmarks: mark specific lines in files and navigate to them from any file in the project.

> **Note:** you can change the hot keys for extended features using "Editor" tab of the Options dialog. See Configuring the Editor for more information.

Configuring the Editor

You can configure the Editor using the Options dialog at any of the multiple configuration levels. On the Main menu choose Options | [*level*], where *level* is the configuration level at which the settings will apply (*Default* for example) . Select the Text Editor node. You will see the following nodes representing the main categories of configuration options for the Editor:

Code Sense:a set of options for CodeSense, the feature that auto-completes reserved words in a supported programming language (only Java is currently supported).
Keyboard: a set of options for customizing Editor hot-key combinations
Schemes: a set of options with sub-options enabling you to customize colors, indentation, and text formatting for text editing and programming languages.
External Editor: a list of external editors available.



Code Sense node

Expand the Code Sense node to view and modify options for the Code Sense feature.

Keyboard node

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.

Schemes node

This node presents a tree of option subnodes. Use these options to tune the Editor for working with different contents:

- Plain: for plain text files
- Java: for Java source code files
- C++: for C++ source code files
- IDL: for Interface Definition Language files

For each of these types of text you can set up color schemes, Snippets for programming languages, and common Editor parameters such as Tab Size, Right Margin, etc. The Editor will automatically pick up the particular scheme depending on the type of file loaded and focused in the Editor.

External Editor node

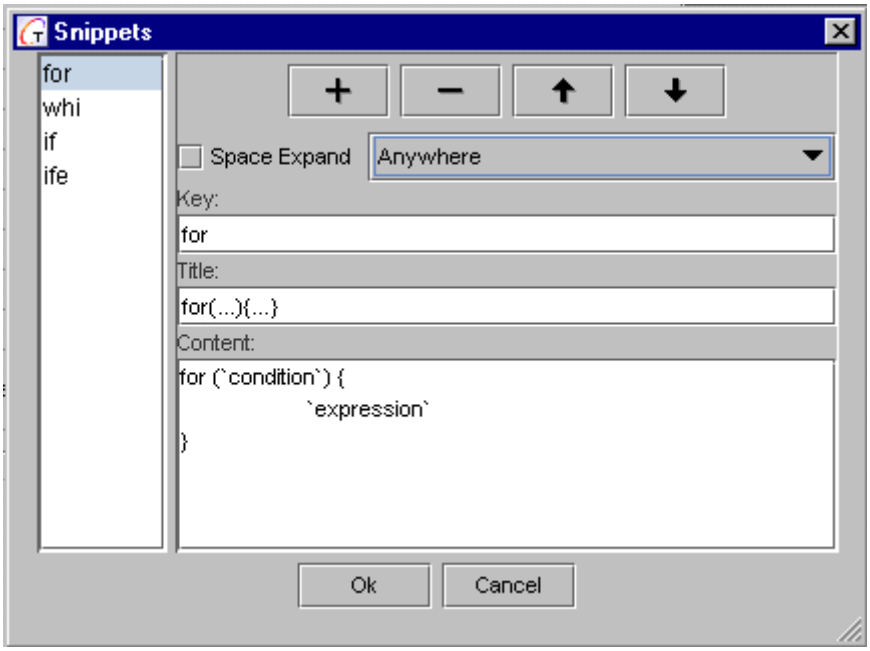
Expand this node to define an external text editor and customize its display in menu.

Defining Snippets

Use the snippets to significantly speed up the process of developing your applications. Define as many snippets as you want. Then, while coding in the Editor, type the name of the snippet and press Ctrl-J to insert the entire block of code contained therein. There are already several default snippets for common constructs in each supported language: *if, for, while,* etc.

Defining your own snippets

1. Open *Options* Dialog.
2. Select the *Text Editor* node.
3. Expand the Schemes option node in the tree of options.
4. Expand the node for the language you want to configure (plain text, C++, Java, or IDL).
5. Select Snippet option node and press *Edit* to display Snippets Editor dialog.
6. Select one of the existing snippets listed at left, or click the "+" (plus) button to add a new snippet. If adding a new snippet, give it a name.
7. Edit or add the actual code that you want inserted in your source files when you invoke this snippet as you work later on.
8. Check *Space Expand* to enable expanding a snippet by hitting space bar rather than CTRL+J.



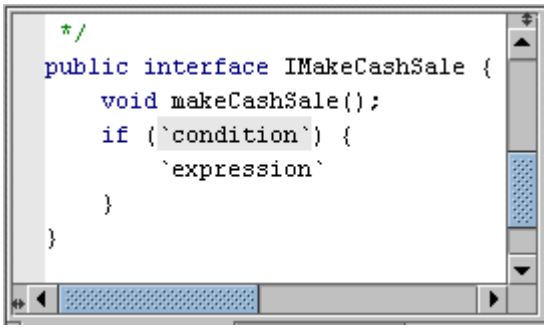
9. From the drop-down list, select the range, where the snippet is applicable.
10. Click OK when done and close the Options dialog.

Deleting Snippets

1. Perform steps 1-5 listed above.
2. Select the snippet name you no longer need at the left.
3. Click the "-" (minus) button. The selected snippet is permanently deleted.

Using Snippets

To make use of the snippets, type in the keyword and press CTRL-J. The code completes automatically, leaving you a chance to do some job of replacing the placeholders with the required expressions.

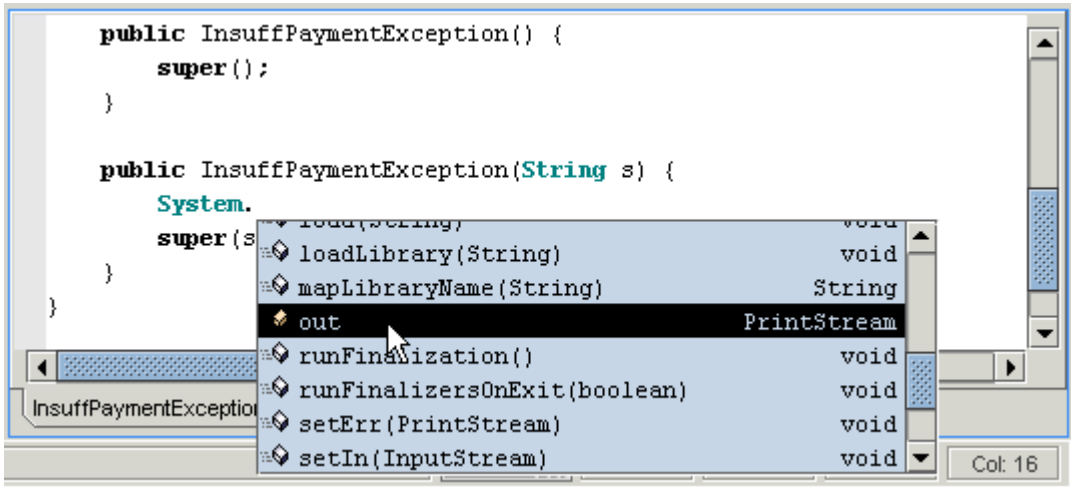


Using Code Sense

The Code Sense feature is currently available in products with Java language support. It helps you to auto-complete references to standard Java classes in Java code, significantly speeds up your coding and helps you avoid syntax errors. Code sense works in two modes: it is invoked by a hotkey stroke (CTRL-SPACE by default), or activated upon delay. This is defined in the Editor tab of the Options dialog, Code Sense and Keyboard nodes.

Example 1

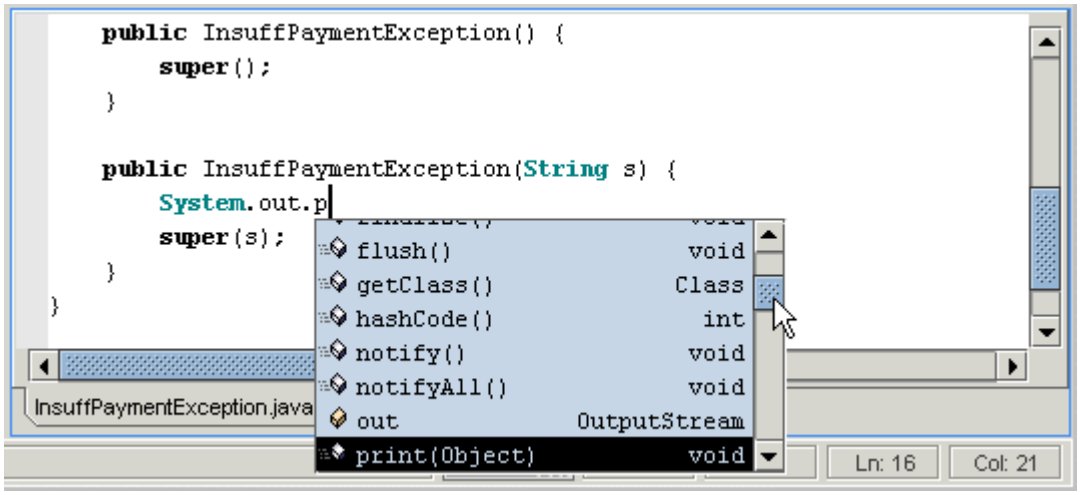
1. Type `System.`
2. With the insertion point cursor placed after the second period character, press Ctrl-Space. The list of available methods for this class is displayed.



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

Example 2

1. Type `System.out.p`
2. With the insertion point cursor placed after the letter *p*, press "Ctrl-Space". The list of available methods for this class beginning with the letter *p* is displayed.



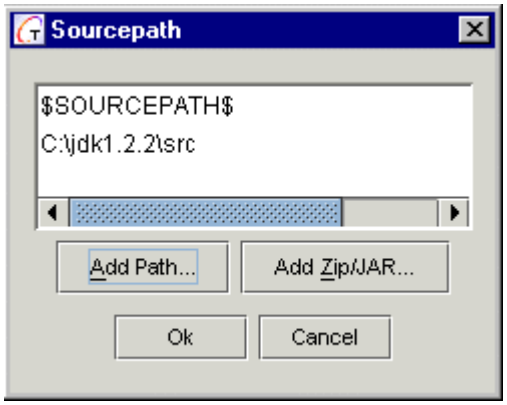
3. Select the desired method with mouse or arrow keys and press Enter. The name of the selected method is inserted into the line.

Browse Symbol

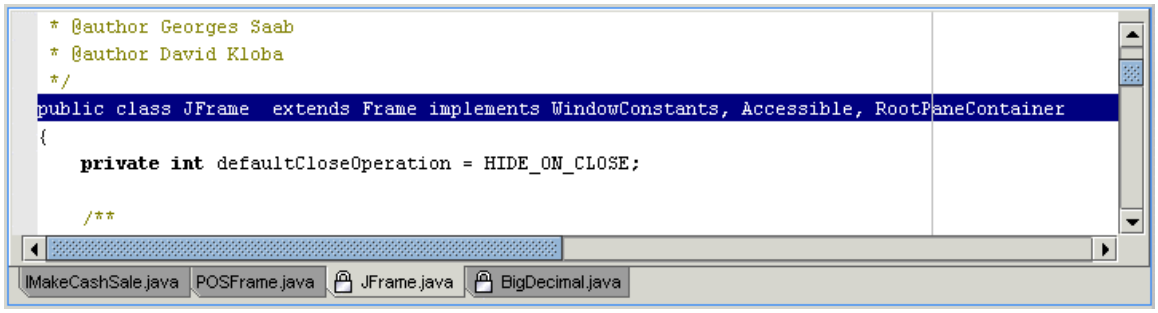
When you click on a class name or attribute name, Browse Symbol opens the source code of this class and highlights declaration. When you click on an operation name, the source code of the corresponding class opens, highlighting the method signature.

By default, Browse Symbol works with the project classes only. To work with the library classes, it needs additional customization of the $SOURCEPATH$.

In the *Builder* page of the Options dialog expand the node *Compiler Options,* select the field *Sourcepath* and specify path to the JDK source jar file. If your JDK home directory is `C:/jdk1.3`, then the sourcepath will be
`$SOURCEPATH$$PS$C:/JDK1.3/src.jar!/src .`
Alternatively, you can unzip `src.jar` and specify path to the source library:
`$SOURCEPATH$$PS$C:/JDK1.3/src`.
You can press the Browse button to invoke the Sourcepath dialog. Press "Add Path" button and select the required path from the file chooser.



- If you wish to add jar file this way, you will need to edit the field and append "`!/src`" to the end of the sourcepath string.
- Press Apply to apply the changes and OK to close the Options dialog.
- The library classes open in read-only tabbed pages.



Breakpoints

Setting Breakpoints

Breakpoints specify where to stop code execution during debugging to permit inspection of variables, expressions, class members, etc.

Setting and removing breakpoints

There are several ways to set breakpoints in the Editor. In every case, place the insertion point cursor at the beginning of the line of code at which execution should stop, and then...

- Click on the extreme left margin next to the line, or...
- Press F5, or...
- Right-click on the line and choose Toggle Breakpoint from the Editor speedmenu.

When a breakpoint is set, a red dot appears in the extreme left margin.

To remove a breakpoint, place the insertion point cursor at the beginning of the line that has the breakpoint, then use any of the techniques listed above to remove the breakpoint.

Working with breakpoints

Once set, a breakpoint can be disabled or re-enabled using the speedmenu commands Disable Breakpoint and Enable Breakpoint.

You can edit properties of a breakpoint by placing the insertion point in the line containing the breakpoint and choosing the speedmenu command Breakpoint Properties.

"Throw-away" breakpoints

You don't need to set a breakpoint that you think you will need only once. Instead you can use the Run to Cursor option to run your application up to a specific line in your program.

To use Run to Cursor:

- Place the insertion point cursor at the line where execution should stop.
- Press F4 (or choose the Run to Cursor speedmenu command).

Setting and navigating 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. Alternatively, use CTRL+M shortcut to toggle bookmark in the current line. Once set, the bookmarks are stored in the project profile and will be re-used when the project opens next time.

Bookmarks have lower priority then the breakpoints. It means that if you set a breakpoint and a bookmark on the same line, the breakpoint will override the bookmark and executable line icon.

The other type of bookmark provided by *Together* is the local bookmark . It is slightly different from previously described bookmark in scope of operation and the way of usage. Global bookmarks are project-wide, while the numeric bookmarks are only valid for the currently opened file.

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. Right-click and choose *Toggle Bookmark* from the speedmenu. (Note that the menu command displays a keyboard shortcut which is user-defined in Text Editor options.)

The default bookmark icon displays in the margin. A default bookmark description is saved using up to the first 50 characters of the line. You can edit this description if you wish (see *Viewing, Editing, and Classifying Bookmarks* below).

To remove a bookmark:

1. Navigate to the line in the open file where the bookmark has been set.
2. Right-click and choose *Toggle Bookmark* from the speedmenu. (Note that this menu command displays a keyboard shortcut which is user-defined in Text Editor options.)

Tips

- You can also remove bookmarks using the Bookmarks dialog (see next section).
- Undo/Redo does not apply to bookmarks.
- By default CTRL+M shortcut is used to toggle bookmarks.

Viewing, editing, and classifying global bookmarks

The 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 individual bookmarks

To display the Bookmarks dialog:

1. Open any file in the project in the Editor.
2. Right-click and choose *Show Bookmarks* on the speedmenu. (Note that this menu command displays a keyboard shortcut which is user-defined in Text Editor options.)

Editing global bookmark descriptions

By default, the *Description* field for a bookmark contains the characters of the bookmarked line (up to maximum 50 characters). The field has in-place editing, or you can use the Edit Bookmark dialog (invoked from the Bookmarks dialog toolbar) to change the description.

Note: You cannot edit the Line and File fields.

You can change the list order of any bookmark by selecting its row in the Bookmarks dialog and using the Up/Down icons on the dialog's toolbar to reposition the bookmark in the list.

Classifying global bookmarks

The Bookmark dialog enables you to classify the various bookmarks in your project. A set of icons is provided for this purpose. You can use this icon set to devise any sort of classification scheme that is meaningful to you.

For example, you might use one icon only for bookmarking constructors, another for start of business methods, another for JavaBean getter/setter methods, etc.

When you select a bookmark in the Bookmarks dialog, you can change its associated icon in either of two ways:

In place: click the *Icon* field in the list and pause for the drop-down list of available icon variants.

Edit Bookmarks dialog: invoke from the Bookmark dialog's toolbar, then use the *Icon* drop-down list to change the icon.

The icon you select for each bookmark displays in the margin of the Editor next to the bookmarked line.

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 Bookmarks dialog (as previously described).
2. Select the target bookmark in the list.
3. Click the *Go To* button.

If the target file is not open, it opens in a new Editor tab and the cursor moves to the bookmarked line. No diagram is opened... just the file.

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

If you have a small number of bookmarks, you can navigate within a single file using hotkeys. CTRL/Grey+ moves forward, CTRL/Grey- moves backwards. Hotkeys do not work for the entire project.

Setting and removing local bookmarks

Local bookmarks are very fast and handy to operate. They are numeric, in the sense that they are numbered from 0 to 9. Hence, there can be only ten local bookmarks per file.

To set local bookmarks:

1. Open the file to be bookmarked in the Editor.
2. Scroll to the line where you want to set a bookmark and place the insertion cursor anywhere on the line.
3. Press CTRL+SHIFT+number.

The numbered bookmark icon displays in the margin. Each new bookmark gets the next number. The local bookmarks don't add to the list of bookmarks and don't display in the Bookmark dialog.

To remove a bookmark:

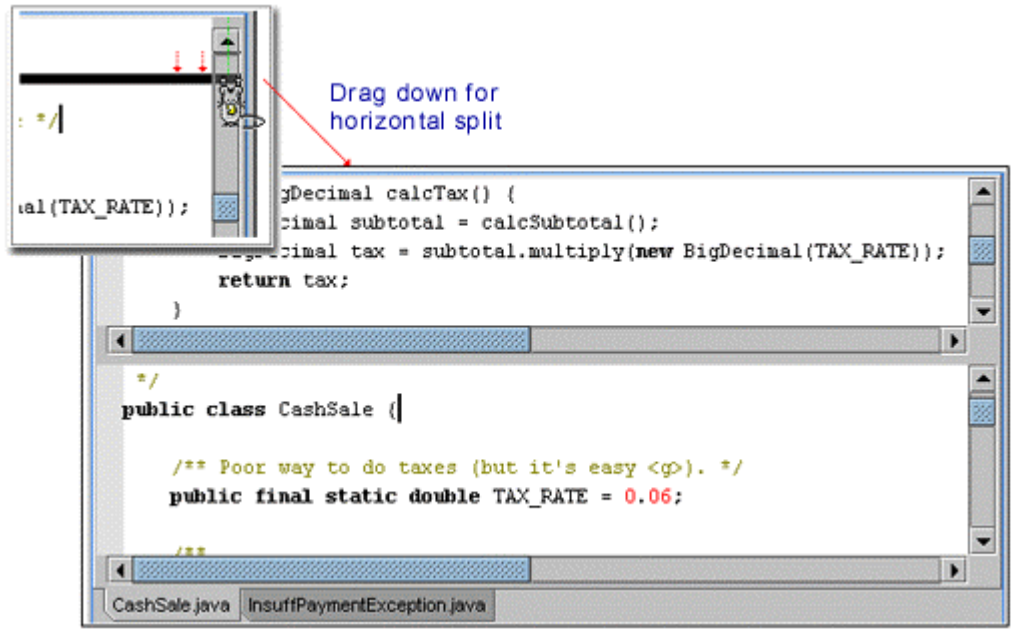
1. Navigate to the line in the open file where the bookmark has been set.
2. Use same keyboard shortcut: CTRL+SHIFT+number displayed on the bookmark icon.

Navigating with the local bookmarks

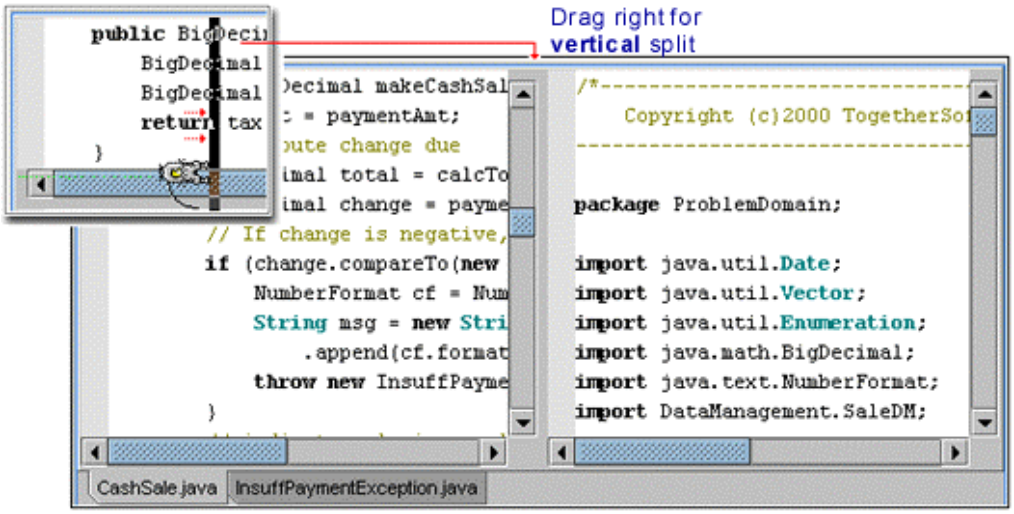
Navigating with the local bookmarks is nice and easy. All you have to do, is to press CTRL+number, and the cursor moves to the start of the bookmarked line.

> **Tip:** Local bookmarks work with the main keyboard only. Small numeric keypad buttons produce no result.

Split pane



For the ease of editing large files, the you can split the Editor pane into two or four segments, each one with own scroll bars. To split the Editor pane horizontally, grab the upper right corner of the pane and the drag splitting line down. If you grab the lower left corner of the pane and drag the splitting line to the right, you can split it vertically. Typing in any split pane is reproduced in all the other panes.



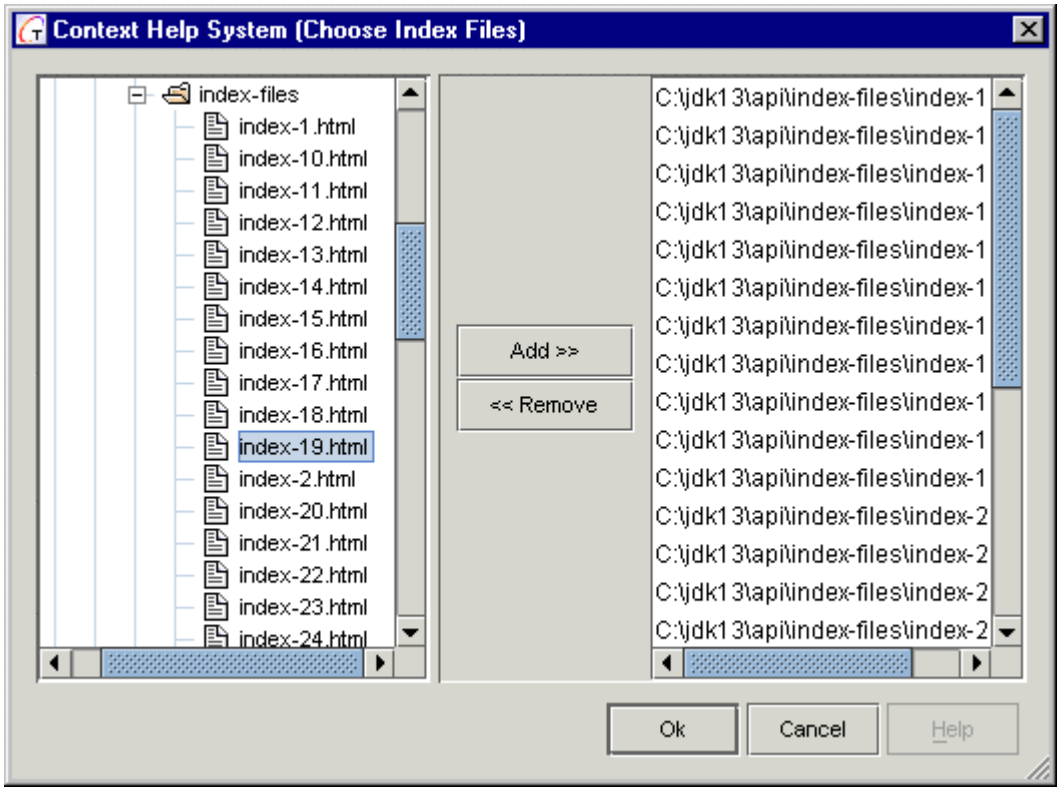
To remove split you need only to drag the splitting line away.

Context Help

Text Editor allows to create and use context help that provides access to the specification of JDK used with Together (for example, jdk 1.3), and other documentation at the user's discretion (EJB specifications, J2EE, WebLogic etc.)
To enjoy the advantages of this feature, you have to create the database. It is possible to create context help system on the General level, that covers all resources required for development, and on the Project level, that includes information related to the specific project.

Creating context help database

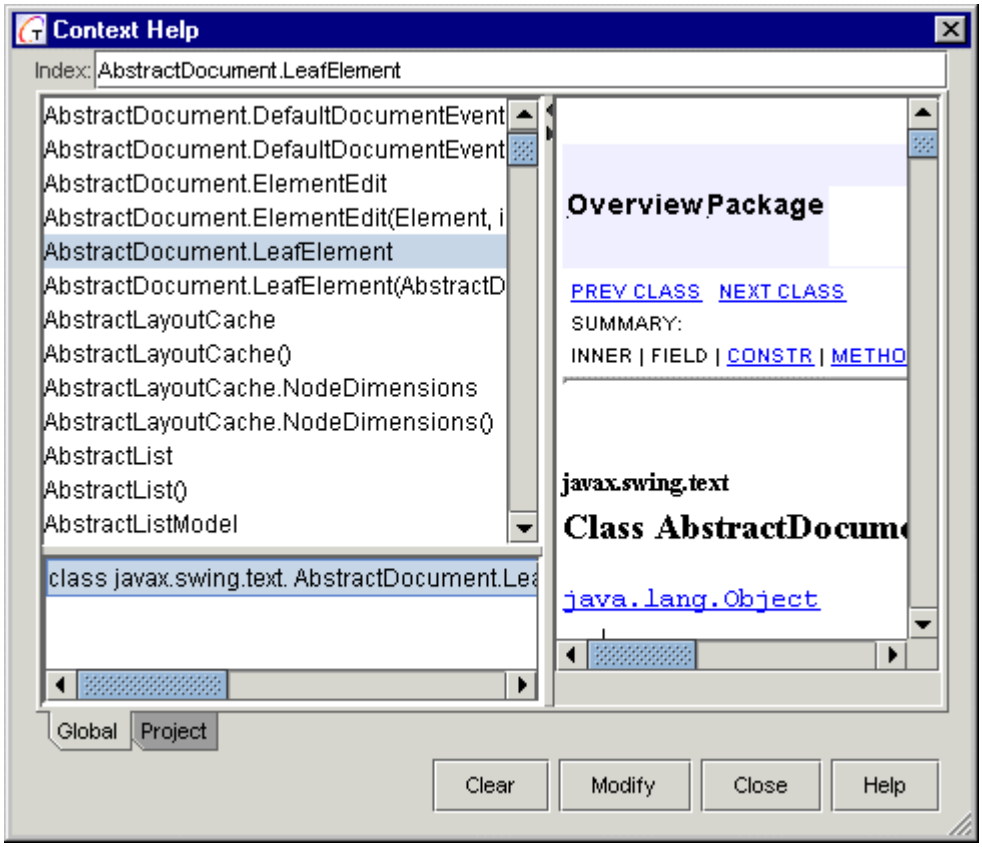
Help databases are not supplied with Together. It is the user's responsibility to create context help system according to the specific requirements. If you have to create a new database, or update an existing one, press *Modify* button. Together asks for confirmation. If you are eager to proceed, answer "yes".



Context Help System dialog in the regular form of Selection Manager shows up. The left pane displays the treeview of available resources. Select required documents and add them to the target pane.

Using context help

Pressing F1 button on the Editor pane, displays the Context Help dialog, that contains two tabbed pages. *Global* tab displays the default level help database, and the *Project* tab is for the project-specific database.



When the database is successfully created, pressing F1 with the cursor on a certain keyword will put in action a search throughout the context help database.

Index field displays the keyword for which the context help is invoked. The upper left frame displays the list of available documents. The lower left frame displays fully qualified path to the selected document. The right frame displays documentation for the selected class or package.

You can navigate through the list of available contents with arrow keys of a mouse. When an entry is selected, the relevant path displays in the lower frame. However, the documentation frame doesn't refresh, until you click on the path string in the lower frame, or hit Enter on the selected entry.

Refer to the Context Help description in the Dialogs chapter for detailed information about controls.

See also

Using the Integrated Debugger
Using compile and make from Together
Using Code Templates

Editor tips and tricks

Opening files for editing

You can open one or more files for editing. Each file opens in its own tabbed page in the Editor pane.

There are several ways to open files for editing:

Right-click on a file name in the Explorer and select **Edit** or **Edit in New Tab** from the speedmenu. The file opens in the Editor pane. (Only source code, configuration, and properties files have these speedmenu commands.)

Right click on a file name in the Explorer and select **Tools | External Editor**. The file opens in the application configured as External Editor in Options: Tools.

Choose **Open** on the File menu and select file from the Open file dialog. The file opens in the Editor pane. (File must be text, not binary.)

Click on a class (interface, link, member, method) in the Diagram pane. The source code opens in the Editor pane tab, highlighting the line that corresponds to the selected element.

To close a file you don't need anymore, select it and choose Close on the File menu.

Showing - hiding the Editor pane

You can easily hide or show this pane using the View menu or the Main toolbar.

The Editor pane may be hidden by default depending on the *Role after restart* selected in Options : General

Using 'Preserve Tab'

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 by checking *Preserve Tab* in the Editor pane speedmenu while the active tab is selected. The current tab remains and a new tab appears named <*untitled*>. Your next selection of a source-generating element in the diagram opens its source in this new tab (or you can open some other file using the pane speedmenu).

Checking *Preserve Tab* for a selected tab doesn't prevent you from closing it later. It only means that if you open another file while the tab is active, the other file will open in a new tab thus preserving the flagged tab. This enables you keep several source files from the same diagram open in the Editor pane.

Using the Editor with an open project

When a project is open, you can have one or more source-generating diagrams open concurrently in the Diagram pane. As you click on source-generating elements in the current diagram, the contents of the Editor updates to display the source code for the selected element. It displays that same file until you select a different source-generating element in the same or another open source-generating diagram.

You may also open non-source diagrams such as Use Case or State concurrently with source-generating diagrams. When you first open a project, the following default Editor pane behavior is in effect:

- When you then open or select the tab of a non-source diagram, the Editor pane hides automatically.

- If multiple diagrams, some source-generating and some not, are open concurrently, the Editor pane shows when you select a source-generating diagram in the Diagram pane, and hides when you select any other diagram.

This default behavior prevails unless you override it by using the Main Toolbar or View menu to show the Editor pane while you are focused on a non source-generating diagram. From that point on, until the end of the current session, you control the display of the Editor pane using the View menu or Main Toolbar.

Note: If you have a file already opened in some of the Editor's tabs, *Together* will open the existing tab on attempt to open that file again.

Using the Editor with no open project

When you launch *Together,* the Editor pane fills the entire right side of the Main Window and displays a single tab. A new file "<Untitled>" is open. You can immediately edit and save this file, or you can use the Directory tab in the Explorer or the Editor speedmenu to open one or more other files (see Opening Files below). Files supported by the Editor pane include the source files of the supported language(s), 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.

Using JSP and HTML Editor

Together's Editor allows working with HTML and JSP files.

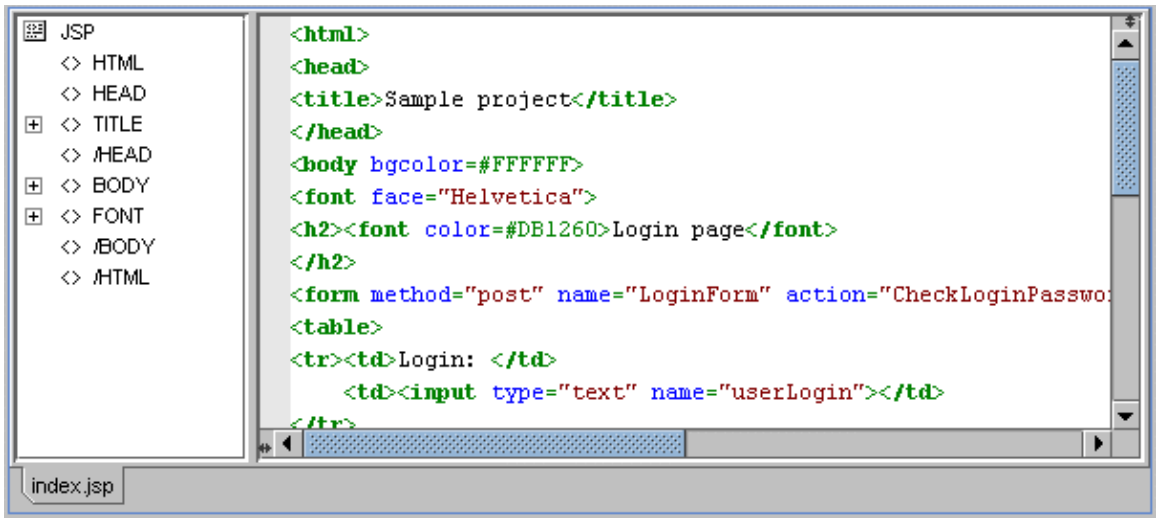
Opening files for editing

There are several ways to open *.html or *.jsp file in the Editor pane. The first way suggests to navigate to the desired file in the Explorer and double-click on it, or choose Edit / Edit in New Tab on the file speedmenu. Another option is to use File | Open on the main menu. This brings in the File Chooser dialog, where you have to navigate to the required file and select it. In this case, each file opens in its own tab in the Editor. You can select file to edit by clicking on the required tab.

Alternatively, you can open *.jsp file in the Editor from the Web Application diagram, selecting the required visual component on the Diagram pane. To use this way of opening jsp files, make sure that *JSP Source* is specified in the *JSP Properties* tab of object inspector.

Specific view of HTML/JSP Editor

The file content displays in the Editor pane. HTML Editor page is divided into two sections. The right section contains the source code in html format, with java code lines, added in case of JSP. The Editor highlights keywords and tags. The left section displays structured treeview of the document tags and highlights typical constructs (opening and closing tags for headers, body text, tables, hyperlinks etc.)



Structured Browser

The structured treeview of the html or jsp document provides a convenient way to navigate through the document. You can expand the nodes and select the required tags or elements. One click on an element highlights it, the second click highlights appropriate location in the edited code.

All changes in the code are immediately reflected in the tree structure, and vice versa. These features of the editor significantly speed up the coding process.

Code sense in JSP Editor

Code sense is also available for JSP Editor. It works basically same way as in the Java Editor. However, to make use of this feature, you have to modify Search/Classpath of your project. Invoke Project Properties dialog on the File menu, and add `$TGH$\bundled\tomcat\lib\servlet.jar` to the Search/Classpath. Actually, this resource is not required for jsp debugging, but makes code sense working.

Viewing HTML files

There are two ways to view an HTML file:

1. Choose Tools | View in Browser command on the Editor's speedmenu.

2. Navigate to the desired file in the Explorer and choose View command on the node's speedmenu.

Thus the HTML file displays in the default browser window.

Tag Library Helper

Tools node of JSP Editor speedmenu contains *Tag Library Helper* command that allows to re-use tags in JSP code. Provided that the tag library is already created, this command displays the list of available tags. Refer to Taglib diagram section for details.

Compile-Make-Run

Using Compile and Make from Together

When developing with Together you can compile classes and make your project without leaving the Together environment. There are several options:

Using default compiler/make: This will execute the default *Java* compiler and make utility installed along with Together.

Using another (external) Java compiler/make: Executes a user-defined external Java compiler and/or make utility. You need to modify the compiler specification in your configuration options (see below) to point to a different compiler.

Using C++ compiler/make: Compiles a C++ project with your preferred external tools. You need to modify the compiler specification in your configuration options (see below) to point to a different compiler.

Configure compile and make tools using Options dialog (Options | Tools on the required level). Refer to User's Guide: Configuring Together: Multi-level configuration for details of multiple configuration levels.

Using the default Java compiler (SDK)

On Windows platforms, Together installs and uses `javac.exe` from Java 2(tm) SDK version 1.3 as the default compiler.* This compiler is located in `$TGH$/jdk/bin`.

For other operating systems, you need to get the recommended Java 2 SDK for your platform, install it, and make sure it is included in your environment's search path. Field "JDK Home" in the Options Dialog points by default to `$TGH$/jdk.` If JDK is not a part of installation, actual JDK location should be specified in this field. Otherwise, compile, make and run will not work.

> **Note:** Java (Javac) compiler will misbehave if the system temporary (TMP, TEMP) directory path contains spaces, as can often occur on Win32 systems. Together attempts to intercept such errors and try alternative paths (user's profile directory, system root) to store the necessary temporary files. However, to prevent problems, specify Together's temporary directory during installation and set this environment variable for your operating system. If the environment variable is not defined elsewhere, you may set it in Together.bat (for Win NT/9x) or Together.sh (for Unix).

Executing the compiler and make tools

The default Java compiler/maker can be executed from :

- Project, package or class menu in the project explorer
- Project, package or class menu in the class diagram
- Current file in the editor
- Buttons on the Builder pane
- Using keyboard shortcuts

Compiler and maker are pre-configured to redirect their output to the Messages tab of the Message Pane. In case of errors/warnings during compiling/making you can simply click on the appropriate line in the Message Pane and navigate directly to the line of source code that caused the error/warning. The Message Pane also displays executed commands and status of tools' execution.

Configuring the compile/make utilities

In general, the default compiler is pre-configured, and thus ready to work. However, if you need to reconfigure it, choose *Options* dialog, referring to command syntax for popular compilers for details.

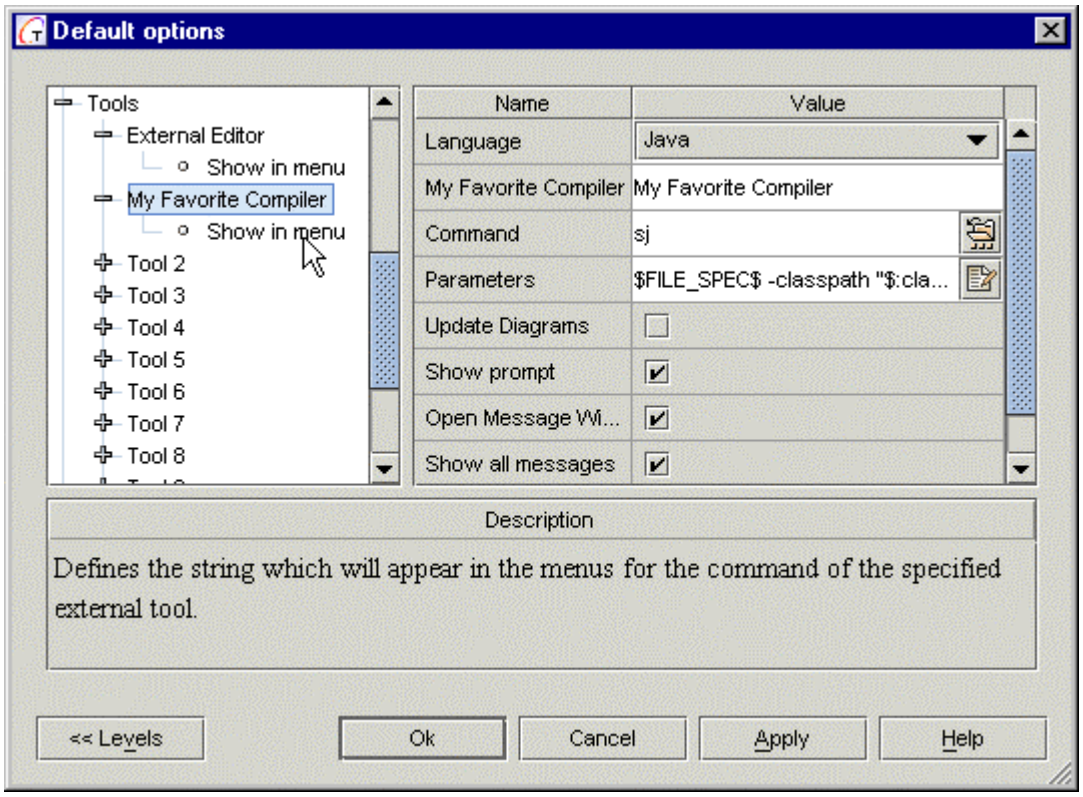
Using another Java compiler

You are free to use compiler at your own discretion. If you are not inclined to use the default compiler, you have to configure your favorite compile / make tool in the *Tools* node of the *Options* dialog.

Expand sub-tree at the selected *Tool#x* slot and configure command for the tool:

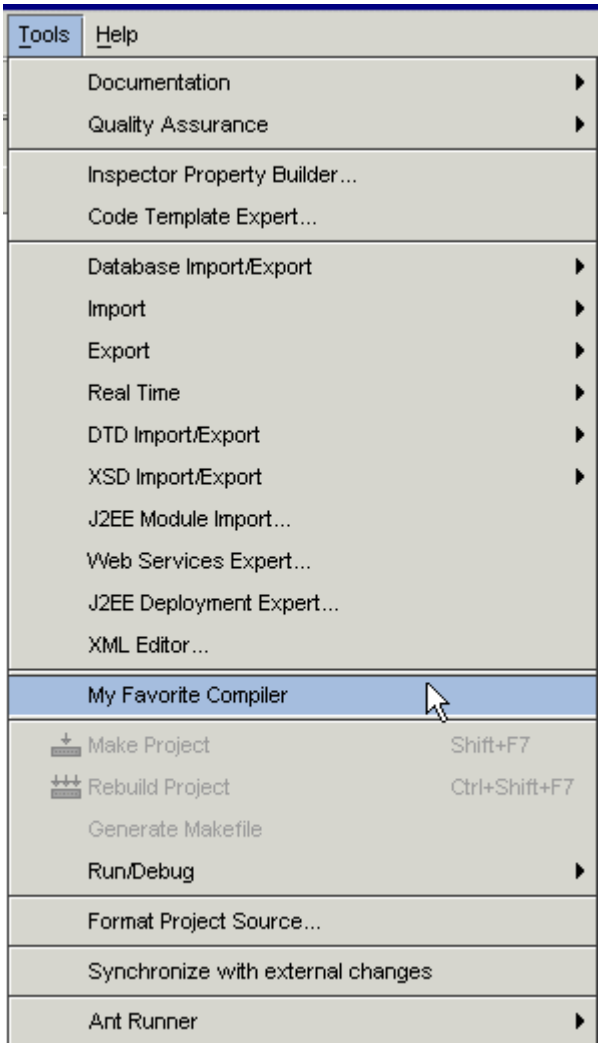
- Specify command name to be displayed in the menu, command line parameters, compile/make output, menu settings etc.
- Check required boxes in the "Show in menu" node to specify the menus where this command can be invoked
- Use information from the Description window of the Options dialog to create proper configuration.

Tip: To avoid overwriting the default compiler/make settings, use empty "Tool #x" slots.



Executing other compiler or make utility

Having specified your own compile and make commands, you can run them from any menu that you have checked.



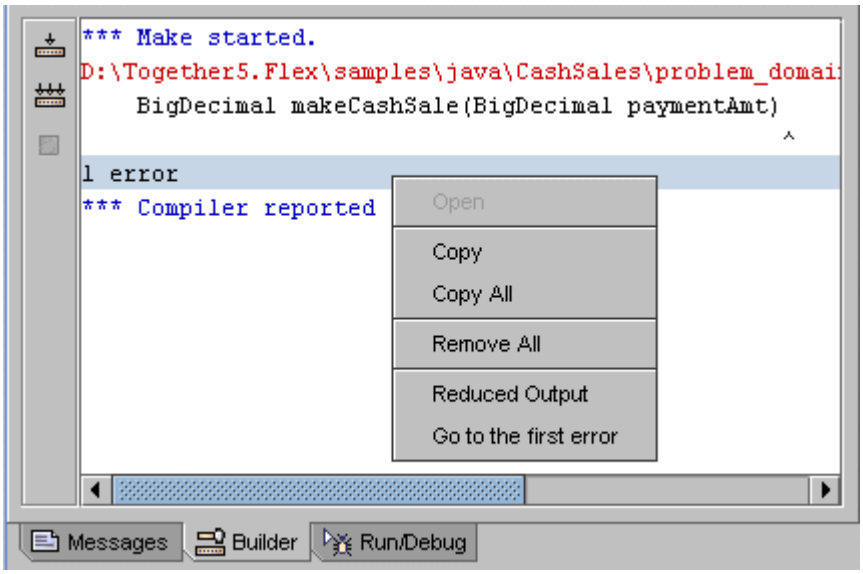
Using a C++ compiler

Together provides no default C++ compiler. Set up a compiler at your choice, following same procedure as described above (Configuring other Java compiler/make utilities). Refer also to External tools section under *Using Together with C++* for further details.

Compiler output

Compiler errors/warnings are presented in a navigable overview of the entire list of errors/warnings, thus reducing the compiler's default output. You can then expand the overview list to get the detail.

For example, if the compiler returns an error , use the right-click menu in the Builder's output pane and choose "Full Output" to see exact location of the error. You can reverse the display selecting "Reduced output" on the Builder's speedmenu.



See also

Using the Integrated Debugger
Guide to the Options dialog: Tools

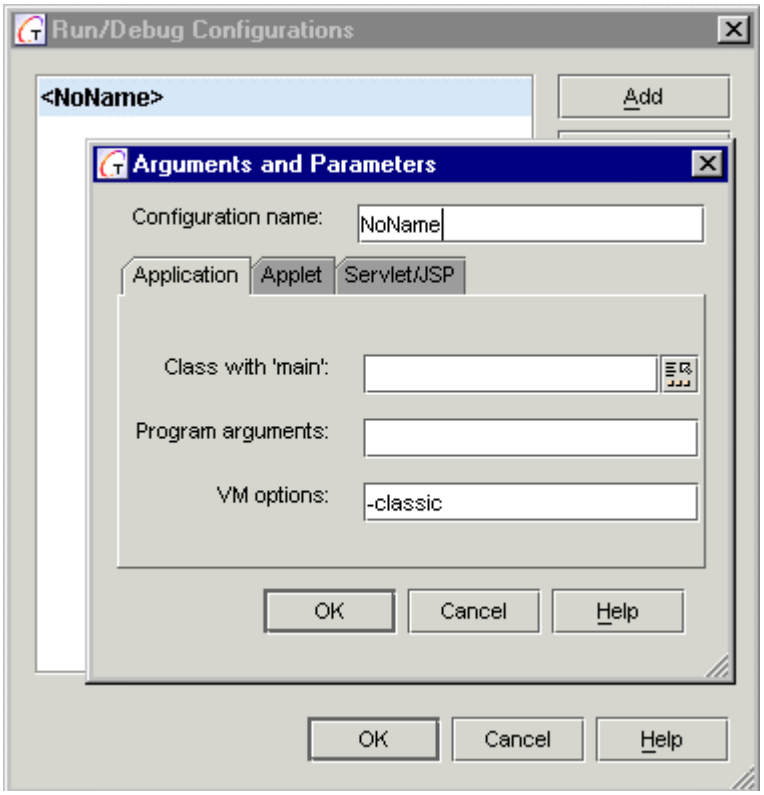
*(Current version as of this writing. For up to date information on supported/shipping JDK see system requirements posted at www.togethersoft.com.)

Run/Debug Configuration

In order to run your project, you have to choose Tools | Run/Debug | Run on the Main menu. Running a program requires specifying main class name, arguments and VM options. Large-scale projects may have numerous main classes and significant number of parameters. Together helps simplify the process by means of Run Configuration feature, which allows to enter various sets of runner parameters.

This how it's done... Choose Tools | Run/Debug | Run Configurations on the Main menu, or use keyboard shortcut CTRL+SHIFT+F5. Press Add button and enter the required arguments and parameters. You can add as many configuration as needed, and assign one of them as the default one. It is possible to create Run configurations for applications, applets and servlets.

Default configuration is highlighted in thick print.



When you further choose Run command from the Tools menu, the program runs with the configuration which is set as default.

Arguments and Parameters

Arguments and Parameters of a run configuration are specified in the Arguments and Parameters frame, which shows up on pressing Add or Edit buttons. The frame contains three tabbed pages: Application, Applet and Servlet/JSP.

Application

Class with 'main': enter the name of the main class in the text area, or press selection button to choose one of main classes, available within the current project.

Program arguments: text area to enter parameters. Multiple parameters are delimited with commas.

VM options: parameters for VM to be launched.

Applet

Class with applet: enter the name of the applet class in the text area, or press selection button to choose one of the applet classes, available within the current project.

Applet parameters: text area to enter parameters. Multiple parameters are delimited with commas.

VM options: parameters for VM to be launched.

Width and height: enter applet frame dimensions in pixels

Servlet / JSP

Start Page / Servlet: specify an entry for the web browser that starts on launching Tomcat. It can be jsp, html, shtml file, or servlet name from the current project.

Query String: enter Start page parameters, separated with &. This creates browser address in the form: http://hostname:port/startpage?p1name=p1val&p2name=p2val....

Context Parameters: enter parameters which can be sent to any JSP/Servlet by the web server, in the format "param1=val1" "param2=val2" ... These parameters are accessible from all servlets. To see them, you can add the following code to your servlet :

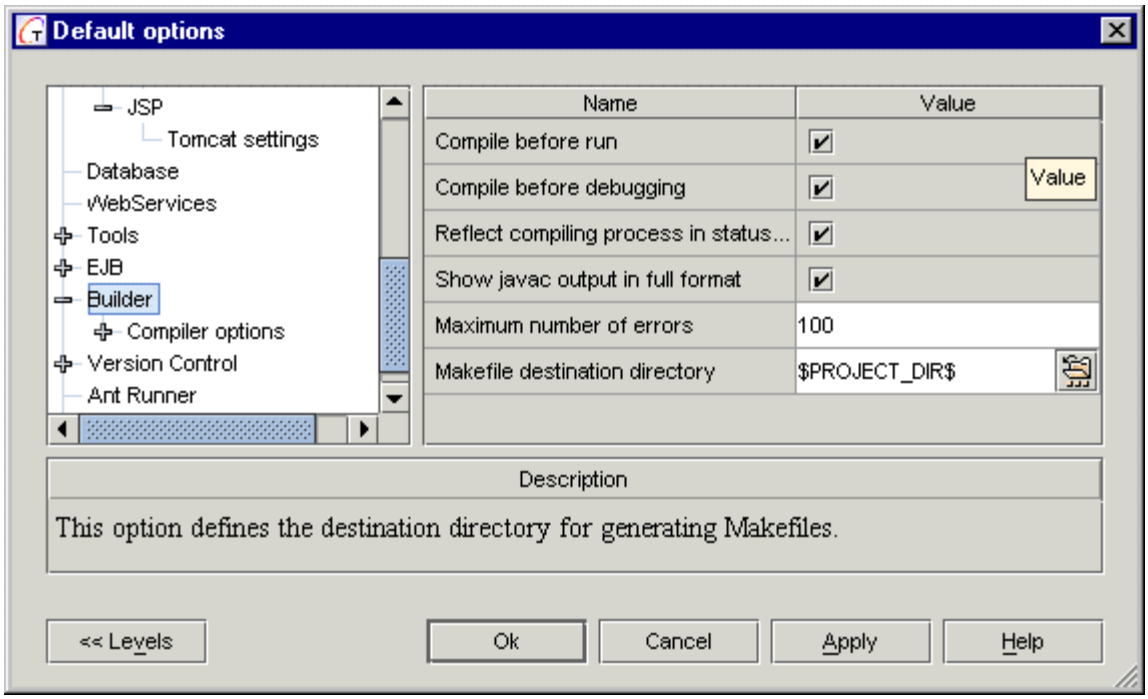
```
out.println("Context init parameters:");
ServletContext context = getServletContext();
Enumeration enum = context.getInitParameterNames();
while (enum.hasMoreElements()) {
String key = (String)enum.nextElement();
Object value = context.getInitParameter(key);
out.println(" " + key + " = " + value);
}
```

VM options: same as above

Makefile generation

Together provides the possibility to generate makefile for your projects. This feature is especially precious for large-scale projects. Having once created a Together project and generated a makefile for it, you can further install this project anywhere, independently from Together.

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



To generate a makefile select *Tools* | *Generate Makefile* from the Main menu. For large projects the process can be rather time-consuming. Finally, a messages shows up to inform you that the makefile was successfully generated.

Resulting makefile is editable. You can add and modify commands and parameters at your discretion.

Debugging

Using the Integrated Debugger

Full-function integrated Java debugger allows you to debug your projects right inside Together.

Debugger features

Breakpoints: Enables you to stop at any line of the source code including "Logging" and "Pass Count" features. Provides 5 kinds of breakpoints: line, exception, class, method and attribute breakpoints.
Command execution: "Run", "Pause, "Continue", "Stop" commands available. For debugging methods there are "Step over", "Step into", "Step out" and "Run to the end of method" facilities.
Watch: You can watch/modify expressions, variables, and class members.
Evaluate: Enables evaluating variables and expressions, and changing their values in course of run in Debugger.
Threads: Enables browsing the state, methods and variables of a thread.
Frames: You can work with the information of the current frame.
Skip classes: This feature enables you to specify classes that should be skipped. Shows the classes that are already loaded.
Remote process: Enables attaching to a remote process, by specifying address and transport.

There are several ways to access the debugger commands: using the Main menu: Tools | Run/Debug, using the Debugger toolbar (Can be accessed using Tools | Run/Debug | Show Run/Debug tab menu command), and the hotkey shortcuts.

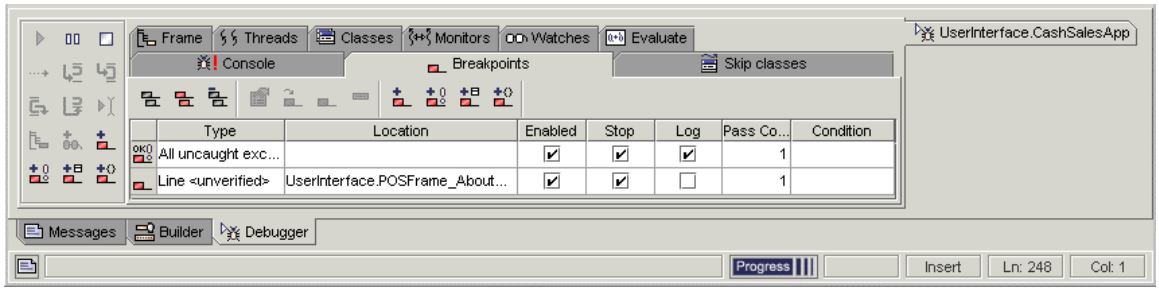
Starting a Debugger session

Tip: Before starting a debugging session be sure you've set up all the needed breakpoints. For more information see Working with breakpoints.

It is possible to start the debugging session in two ways. If you are going to use the current run configuration, choose *Run/Debug | Run in Debugger without showing the Parameters Dialog* on the *Tools* menu, or use the hotkey Shift+F9. If you need to specify the main class or modify run configuration, choose *Run/Debug | Run in Debugger with Parameters Dialog* (Ctrl+Shift+F9).

This starts your projects in debug mode and automatically shows Debugger Tab. Debugger Tab significantly speeds up access to the information you could need during the debugging process.

Debugger Tab



Debugger tab has its own elements:
Console tab: Displays Java console
Threads tab: Threads viewer. Displays in details all the running threads.
Classes tab: Displays a hierarchy of loaded classes.
Monitors tab: Displays synchronization monitors.
Watches tab: Watch window for class members/expressions.
Breakpoints tab: Displays list of breakpoints.
Skip classes tab: Contains list of "disabled for debug" classes.
Evaluate tab: This tab enables evaluating expressions and modifying values (including method executions) when the Debugger is in suspended state.
Frames tab: Frames viewer. Displays the frame stack of the current Thread.
Toolbar: Provides mouse-click access to the debugger commands, such as "Pause", "Resume", "Reset", "Step over", "Step into", etc.

Controlling program execution

When a debug session starts, you have to control the program execution, using the following set of control commands:

Command	Icon	Description	Hot Key
Pause		Pauses program and passes control to the debugger.	
Continue		Resumes program execution.	
Stop program		Terminates the program and debugging session.	Shift-F2
Run to cursor		Resumes program and breaks before cursor.	F4
Step over		Skips debugging of a method that is currently under cursor. The method executes and returns result.	Shift-F8
Step into		Forces debugging of a method that is currently under cursor. Debugger stops at the first line of this method.	Shift-F7
Step out		Forces current method execution and stops in the current method's caller at the next line after call.	
Run to end of method		Forces current method execution and stops before return.	
Toggle smart step		Debugger performs a "smart step". You can set up smart step using *Debugger* tab of the Options Dialog.	

See also

Using the editor

Breakpoints

Breakpoints provide the most powerful debugging facility, which makes it possible to break program execution at the specified place, in order to inspect variables, class members, etc.

Setting breakpoints

To set a line breakpoint, it is enough to select the desired line and press F5, select Toggle Breakpoint command on the speedmenu, or just click on left margin next to the this line. (See also: User's Guide: Using the Editor: Setting breakpoints). Alternatively, use appropriate Debugger Tab toolbar button.

Class, exception and method breakpoints are added by pressing the Debugger Tab toolbar buttons, or by appropriate speedmenu commands on the Breakpoints tab.

The attribute breakpoint has no icon on the Debugger toolbar and can be added from the Watches tab only. Select the desired watch, and choose Add Attribute Breakpoint command on its speedmenu. This adds breakpoint for the selected attribute to the Breakpoints tab.

Controlling breakpoints

Breakpoints tab of the Debugger provides a toolbar that enables full control of the breakpoints:

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

Modifying breakpoint properties

After the breakpoints are set, you can disable or enable them. To do this, check or uncheck *Enabled* flag for the appropriate breakpoint in the Breakpoints tab. If a breakpoint is enabled, it is accessible for modification.

To modify a line, class, method or exception breakpoint, choose Breakpoint Properties command on the speedmenu in the Editor pane, or in the Breakpoints tab of the Debugger. Alternatively, use the Debugger toolbar icon. This bring in a Breakpoint Properties dialog that allows to define if there should be a stop execution at this point, specify number of passes through this breakpoint before stop, conditions, and also whether this stop should be logged or not.

Besides the Breakpoint Properties dialog, it is possible to modify all type of breakpoints in the Debugger tab. The controls of the dialog are replicated in the table of the *Breakpoints* tab. Note that *Pass count* and *Condition* fields are editable in place. The flags *Stop Execution* and *Log Message* may not be disabled simultaneously. At least one of these flags must be set. Refer to the description of the Breakpoint properties dialog in the Context Help chapter of this manual for detailed description of the fields.

Attribute breakpoint has two properties that are modified in a slightly different way. These properties are*: Stop on Read* that allows to break application when an attribute is about to be read, and *Stop on Write* that allows to break application when an attribute is about to be written. By default both options are enabled. You can enable/disable these options, choosing Enable/Disable Stop on Read/Write on the speedmenu of the selected attribute breakpoint.

Examining data values at breakpoint

When staying at the breakpoint you can examine data using "Add watch" command, available from "Tools | Run/Debug" menu and from the Debugger Tab toolbar.

Attaching to a remote process

You can remotely debug Java programs with the integrated debugger. Start the external Java program to be remotely debugged (debugged per attach) in the following way:

```
java ... -Xdebug -Xnoagent -Djava.compiler=NONE --> -
Xrunjdwp:transport=dt_socket,address=8787,server=y,launch="%1\bin\win32\
display.bat %1"
```

If port 8787 is not convenient for you, you could allow the JVM to define the port:

```
-Xdebug -Xnoagent -Djava.compiler=NONE --> -
Xrunjdwp:transport=dt_socket,server=y,launch="%1\bin\win32\display.bat%1
"
```

In the latter case, the JVM prints the address. Note the address and enter value in the Attach to Remote Process dialog.

See also

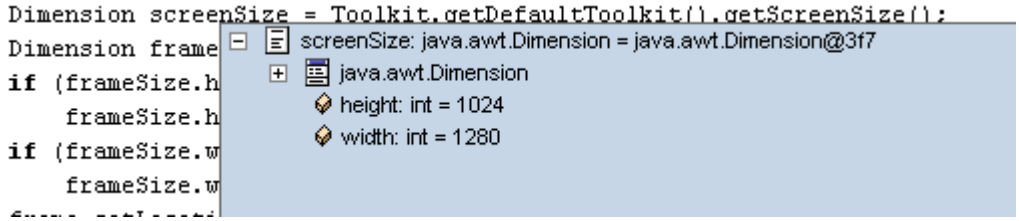
Using the debugger

Evaluating and Modifying Variables

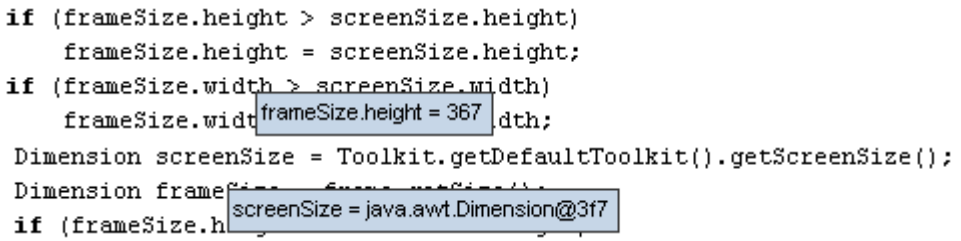
Together allows to access the variables in course of program execution. Set breakpoint in the desired location of your code and select Tools | Run in Debugger on the Editor speedmenu or on the main menu. This adds *Evaluate* tab to the Debugger pane and Evaluate/Modify command to the Editor speedmenu.

Displaying structured context

While your program runs in the Debugger, you can check the value of each object. To do this, you need only navigate the cursor to the desired object, and after a small delay the description of this object will show up. The form of presentation is defined by the *Show variable as tooltip* flag in the Run/Debug | Debugger node of the Options dialog. If the flag is set (by default), evaluation window shows the entire structure and value of an object. Expand the nodes to reveal the constituent properties.

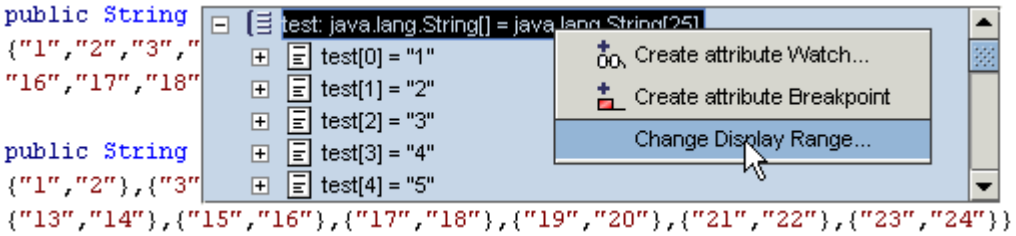


If this flag is cleared, evaluation window shows object's address or value only:



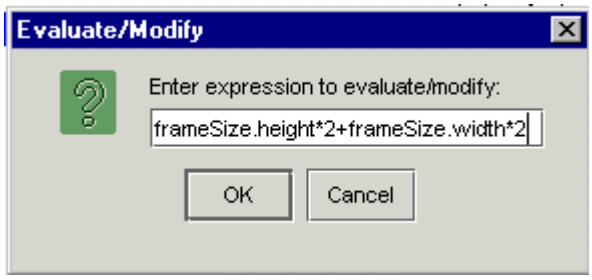
Evaluating arrays

When evaluating an array, you can specify the bounds of display range. To do this, right click on the array name in the evaluation window and choose *Change Display Range* command on the speedmenu, and enter starting and ending element numbers, or an asterisk to display the entire array.

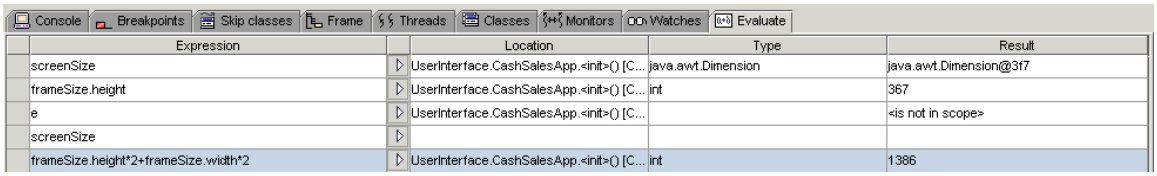


Evaluating and modifying objects

It is also possible to evaluate and modify objects in the Evaluate tab of the Debugger pane. To do this, select Evaluate command on the Editor's speedmenu and specify variable or expression to be evaluated/modified.

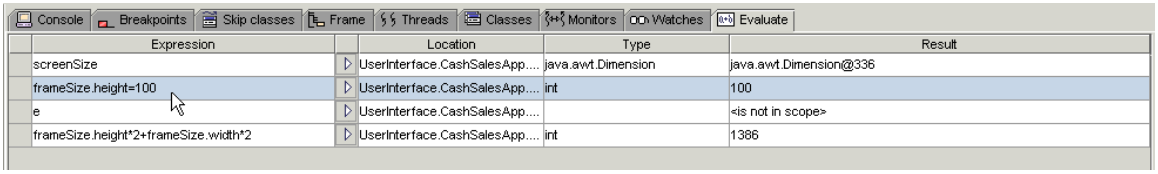


Appropriate entry adds to the Evaluate tab, showing location, type and value of the object in question.



Expression in the list are evaluated every time you choose Evaluate/Modify command on the speedmenu of the current row, or press the right-arrow icon in the second column.

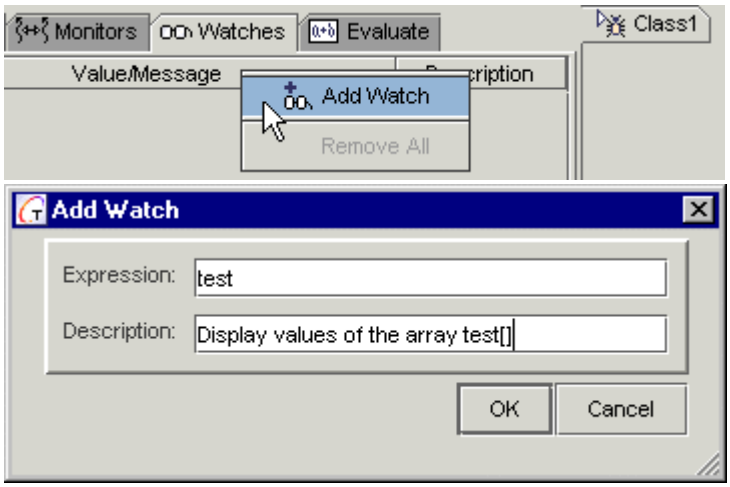
You can change the value of variables, typing directly in the *Expression* field of the tab. Click arrow icon for the changes to take effect.



Note, that you can only evaluate and modify objects within the current debugging context. If you try to check an object past the breakpoint, the Result field will report that the variable is out of range.

Watching Expressions

Watching class members, inspecting objects etc. is an important side of the debugging process. Select **Tools | Run in Debugger** on the Editor speedmenu or on the main menu. This will open Debugger tab with the full set of elements, and show "Add Watch" command on the Editor speedmenu. Now you can add watch on the *Watches* tab of the Debugger, or from the Editor speedmenu.



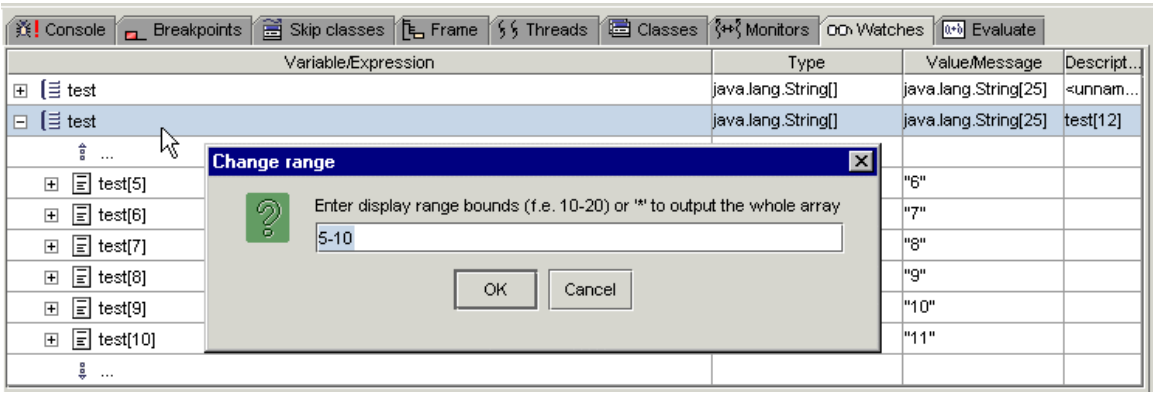
Specify the expression you would like to watch and its description (optionally). This expression displays on the "Watches" tab of the Debugger Tab. Add as many watches as needed.

Using watches

Right-click on the selected watch invokes its speedmenu that provides commands for adding, removing and modifying watches.

Change Display Range

If the expression being watched refers to an array, you can confine its range to the elements that are of interest for you. Speedmenu command *Change Display Range* serves this purpose. You can explicitly specify the numbers of the starting and ending array elements to be displayed in the watch. Entering an asterisk (*) displays the entire array.



Change Values

Having selected a watch in the *Watches* tab, you can change the value of the variable. The values are changed in-place, or by means of *Change Value* command on the watch speedmenu. Note that the string values must be entered in quotes. Otherwise any modifications will be ignored.

Change display format

For the integer variables, it is possible to toggle between decimal and hexadecimal representation. Use complementary commands *Show decimal value* / *Show hexadecimal value* on the watch speedmenu.

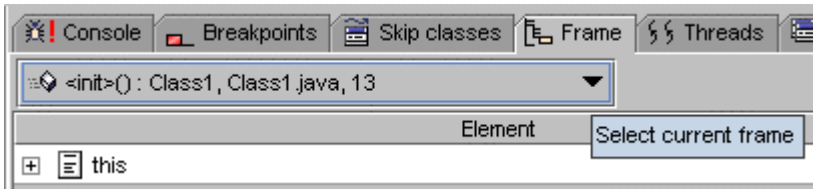
Similar modification features are available in *Evaluate, Threads* and *Frames* tabs. There you can also edit variables in-place and toggle between decimal and hex view for the integer values.

Using Threads and Frames

It is possible to observe the various threads in course of program execution. Using the two buttons on top of the *Threads* tab, you can display all current threads and frames  , display the current thread only  or display all threads  .

Frames tab represents a frame stack of the current thread being resumed. This tab displays information which is a subset of the *Threads* tab. This spares the user from unnecessary navigating to the *Threads* tab, and allows to operate with the current frame.

Combo box on top of the *Frames* tab allows to choose the location of the currently suspended thread.



Re-use Support

Working with Code Templates

Code Templates are used by the OneSource™ round-trip engineering engine 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, the default name is "Class1", and the default code generated for it is:

```
public class Class1 {
  }
```

By modifying the appropriate template, you can change the name to "New1" (or whatever you want). more importantly, you can also modify the default code, adding default attributes or operations. For example, you could change the default code for a Java class so that a default *addNew()* operation is always generated when you create new classes in diagrams:

```
public class Class1 {
   public void addNew() {
    }
  }
```

This may seem trivial at first glance, but consider the implications for things like Enterprise JavaBeans (EJBs). When you use the one-click EJB feature, you get default source code for an entity or session EJB (see figure below). By modifying the respective Code Templates, you can customize the default code for EJB classes, home and remote interfaces, etc. adding fields, properties, business methods... 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 that contain separate sub-folders 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 file `%Name%.*` (with extension specific for the selected language), and optional properties file. Link and member templates have the template name with the **.link* and **.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, flag that specifies whether this template will be displayed in the Choose Pattern dialog, and other information.

Possible properties are:

Properties	Description
defaultName	The name on the created object, that is used as a starting value. For example, `Class1` for the first generated class, `Class2` for the next one etc.
defaultType	Defines types of the attributes and return types of the operations.
hideInChooseList	If this property is present in the `*.properties` file, the template will be ignored by the Pattern Chooser panel.

Properties	Description
generatePrologueEpilogue	If `true`, 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 the operators and members, and specifies that this operator or member can occur in the generated class only once.
patternDescription	Contains brief description of this template in HTML format.
doNotKeepTag	Contains the tag name that should not be preserved when an object is replaced with another one. For example, `doNotKeepTag=link` means that the *link* tag should be omitted in a new link.

Textual pattern *Default class* is used every time a new class is created. Same refers to the interfaces, associations, aggregations, dependencies etc., whose names begin with "Default_". These templates never show up in the Pattern Chooser panel.

The templates make use of macros whose full list and descriptions are provided in the Template Macros. Each template type handles certain pre-defined macros:

Template type	Macros handled
Class	%Name%, %Class_Name%
Member	%Name%, %Type%, %Class_Name%
Link	%Name%, %Type%, %Dst%

The macros not supported by a certain template type will be ignored by the parser, and such template will become inapplicable. In addition, each template type handles unlimited number of the user-defined macros.

Default properties in templates

The template node for each modeling element contains two subnodes (see figure below). The first is for the default source code, the second is for 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, you open and edit the source code node; to change default values of properties you edit the properties node. (See Editing Code Templates below.)

Caveat

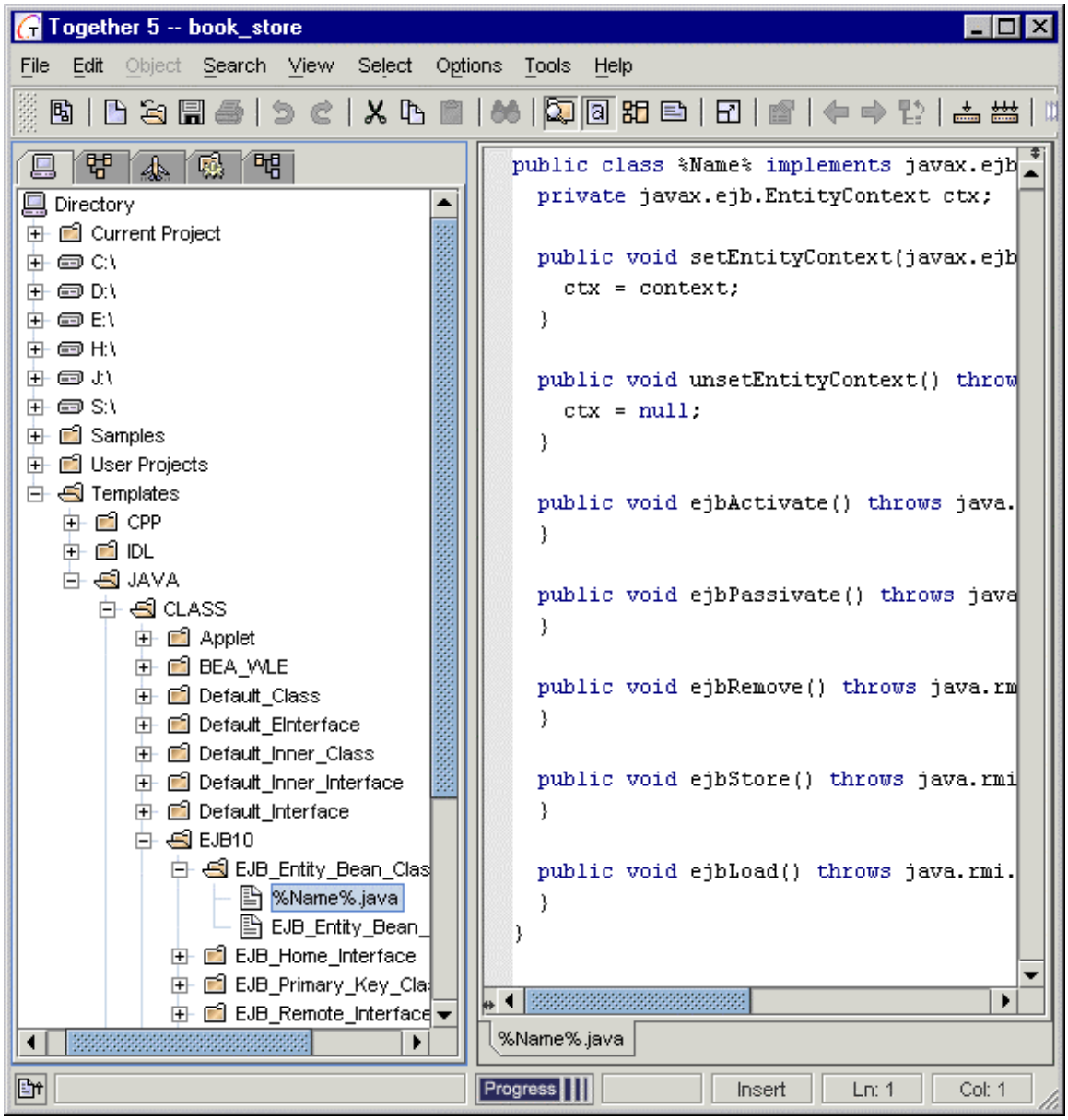
> NEVER play around the default templates. Improper modifications can result in improper functioning of Together, and make it impossible to create classes, interfaces etc. in the diagrams.

Browsing the available Templates

You can browse through the available Code Templates using the Directory tab  of the Explorer. (It doesn't matter whether a project is open or not.) To see the main template categories, expand the *Templates* node. The categories correspond to different programming languages and IDL variants.

Note that while all templates are available to view and edit, not all templates function in all products. For example, Java templates don't work in products that support only C++. Functionality depends upon your product license.

Each main category has subcategories for class templates, link templates, and member templates. Expand any node in the treeview to see the available templates. The figure below shows the structure of the Templates node in the Directory tab, and the default code for the selected element. This is the code generated for this element... in this case and entity EJB... whenever a new one is created in the visual model. You can edit this code (see next section) to change the default source generated for new instances of the modeling element. For example, for the EJB shown below you could add default declarations for finder or business methods.



Code templates in the Explorer, and the default code for an entity EJB

Editing Code templates

You can edit the default source code specified by a template, the default values of properties, or both using the Together Editor. You can open multiple source templates or properties in the editor at once, thus facilitating clipboard operations between templates. Changes to source code or properties are automatically saved when you leave the Editor. You can manually save changes using the File menu or Main toolbar.

Using template macros

You will notice in the default code delivered with Together that several "template macros" (former blueprint macros) are used. For example the default class code uses `%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 chapter Template Macros of this guide.

To open one template for editing:

1. Navigate to the template's node in the Directory tab of the Explorer and select it.
2. Right-click on the node and choose *Edit* on the speedmenu.

To open multiple templates for editing:

1. Navigate to a template node in the Directory tab of the Explorer and select it.
2. Right-click on the selected node and choose *Edit in New Tab* on the speedmenu.
3. Repeat 1 and 2 for additional templates as desired.

Applying source formatting

You can configure quite a number of source code formatting options in the Source Code page of the Options dialog... indenting, comment format, treatment of spaces, etc. 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 properties node) of the template in the Explorer.
2. Right-click and choose *Format Source* from the Speedmenu.

Using an external editor

You can configure Together to invoke any source code editor. If you prefer to edit templates using another editor you can easily do so.

To edit template in an external editor:

1. Check to see that you have configured the external editor (Options dialog, Tools page).
2. Right-click on the template node and choose *Tools | External Editor* on the speedmenu.

See also

Using the Editor
Guide to the Options pages: Tools page
Guide to the Options pages: Source Code page
Template macros

Custom Code Templates

Using Together you can create your own templates and groups of templates, either with 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.

Creating custom code templates

With an Expert

Together provides a handy way to create custom code templates using the Code Template Expert, which is invoked by the relevant command from the *Tools* menu, or diagram object speedmenu. Considering the selected language, the expert creates template and properties files according to the rules described above.

In the expert you can:

- specify template properties
- edit template contents, using appropriate template macros.

The newly created templates show up in the *Explorer* pane. Now you can use them to create classes, members and links by patterns.

There is a rigid dependency between the template type and contents of the inspector. For the classes and interfaces a single control with the class name appears in the inspector. The value of the class name replaces `%Name%` macro in the template.

For the Links the field "Name" is related to the `%Name%` macro, and the field "Link destination" is related to `%Dst%` macro.

For the Members (attributes or operations) there is the field "Name" for `%Name%` macro, and the field "Type" for `%Type%` macro.

If a certain macro is not included into 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.

> **WARNING:** The expert stands guard over the integrity of Together. Default templates used to create one-click diagram elements, whose names start with "Default", may not be deleted by all means. That's why, if you select any default template in the wizard tree, *Remove* button will be disabled. However, modification of the default templates is still possible.

Manually

If you so wish, you can modify the code templates by means of editing the relevant files in `Templates` folder. In this case you are fully responsible for the correctness of created templates.

According to the required template type (class, link or member) and language, create a folder in the appropriate location. The name of this folder should be the same as the name of template being created. Spaces in the folder name are not allowed 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`.

In this folder create file %Name%.h (for C++), %Name%.idl (for IDL) or %Name%.java (for Java), and optional properties file, whose name is the same as the template folder name.

Write your contents of these files. If you want your template to show up in the Pattern

Chooser panel, do not include `hideInChooseList=true` in the properties file.

> **Tip :** Even if you set this flag `false`, the Pattern Chooser panel still ignores the template. This line should be omitted.

The newly created templates show up in Together's Explorer pane with the next start. 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, click *New Folder* button in the *Code Template Expert* and 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 file `description.html` under the folder in reference, and write the necessary information. The description displays in the *Description* area of the Pattern Chooser upon restart of Together.

Displaying custom template names

As mentioned earlier, templates are stored in folders with appropriate names. However, the users might want to see more sensible names in the Pattern Chooser, with spaces, quotes and apostrophes. To do that, add the following line to the template's `*.properties` file:

```
patternDisplayName=new name
```

The new name immediately shows up in the Pattern Chooser, but the actual name still displays in the *Templates* node of the Explorer.

It is also possible to rename template folders in the Pattern Chooser. To do that, create file `folder_name.properties` in the upper-level directory. This file should contain the only line:

```
patternDisplayName=new name
```

Upon restart of Together, the new name shows up in the Pattern Chooser.

Example:

Group of templates Robustness resides in `$TGH$/templates/java/class`. To rename Robustness group into Robustness Diagram, create file `$TGH$/templates/java/class/Robustness.properties` and add line

```
patternDisplayName=Robustness Diagram
```

Restart Together, invoke *Choose Pattern* dialog and observe the new name in the Pattern Chooser panel.

User-defined macros

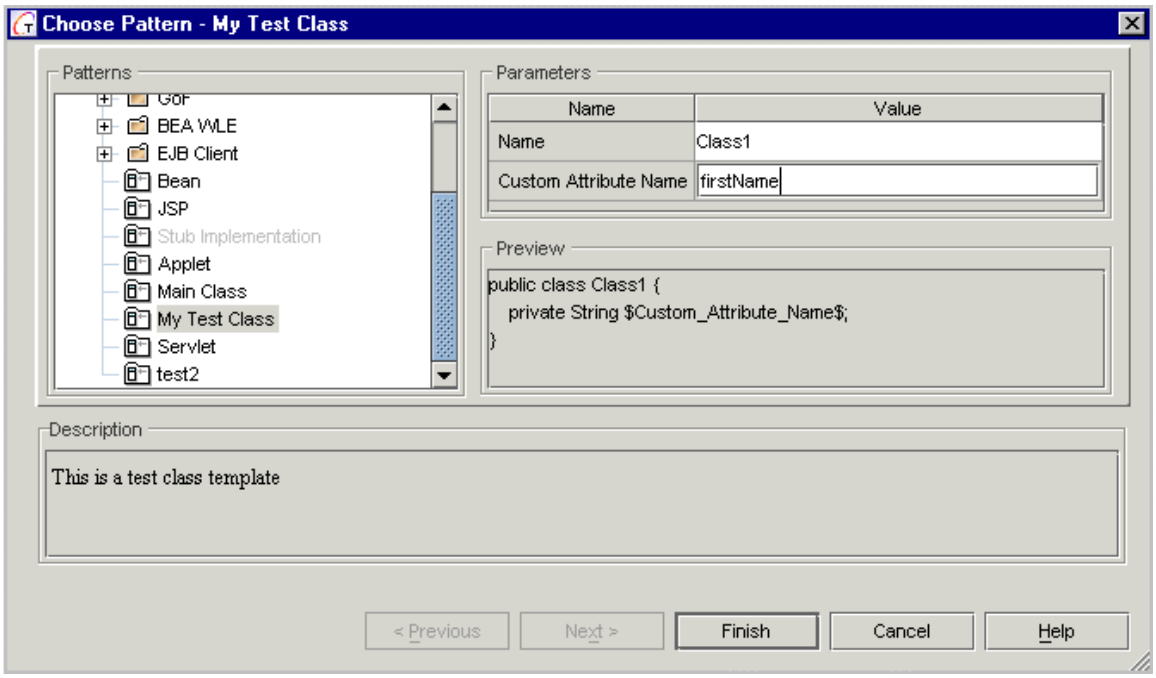
In addition to the standard macros, unlimited amount of the *user-defined macros* are allowed for all template types. The names of the user-defined macros follow the same syntax rules as the template folder names (spaces not allowed). Inspector provides special control for each macro with the name of this macro, and the underscores replaced with the spaces. *Example*: 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 do this, add line

`default.$Custom_Attribute_Name$=<defaultValue>`

to the properties file of the appropriate template folder. The macros without pre-defined default values are initialized with an empty string.



Creating Templates from Diagram Elements

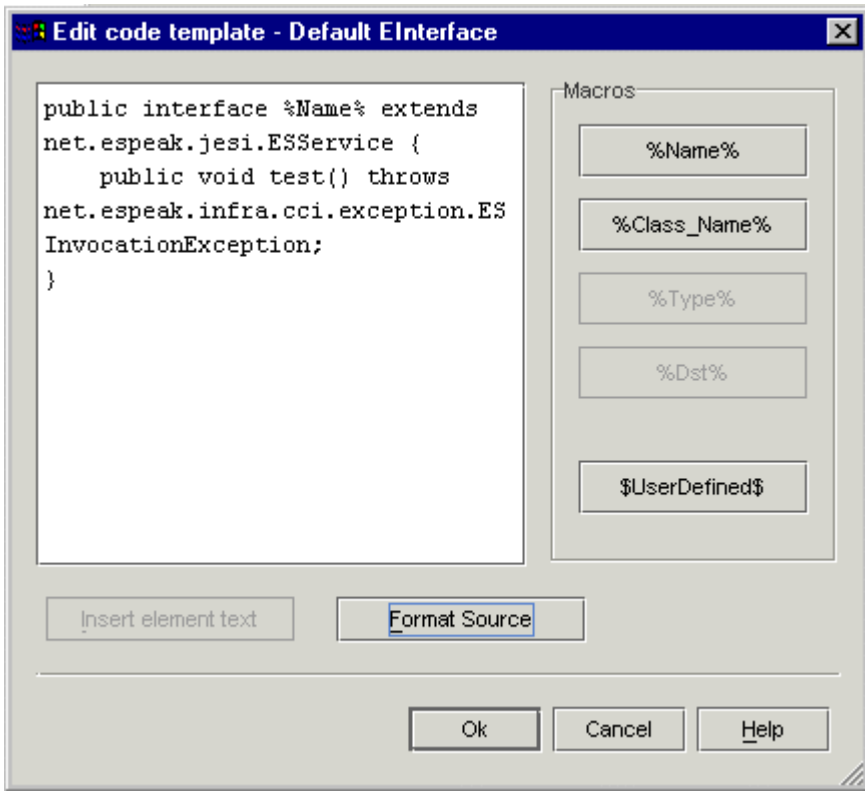
You can create a template based on your existing classes, links or members. Create a class, populate it with the necessary attributes and operations, and make a template from it.

To do this, select an element that will be a source for a new template, and choose Edit Code Template command on the speedmenu. This brings in Code Template Expert.

Code Template Expert displays the second page, the language and category of the future template being automatically assigned based on the selected element.

The treeview of the Templates folder shows only those templates that correspond to the current category of elements. Press *New Template* button and specify template name.

Click on the created template name and press Next to proceed with the settings. On this page of the Expert, press Edit Code Template button. This will bring in the template editor. Press *Insert element text* button to enter the source code of the selected diagram element. Edit the source code, using template macros, format source if needed, and press OK to complete.



The newly created template appears in the appropriate location of the Templates folder.

Patterns

Together delivers a unique capability to extend its functionality externally by using modules and patterns. Modules are discussed in Developing Extension Modules. This topic explains how to use the "pre-fab" patterns included with *Together*, and discusses the basic structure of patterns and points you to information on how to deploy your own patterns.

About patterns

Together patterns are public Java classes that implement `com.togethersoft.openapi.sci.pattern.SciPattern` interface.

Patterns are used to create frequently used elements, to modify existing elements, or to implement useful source code constructions or solutions in your model.

Among the patterns provided with *Together* there are the Coad Components, GoF patterns including Visitor, Observer, Singleton, a set of EJB classes for various specifications, a set of TagLib patterns etc.

Some patterns work only with a specific language. Enterprise Java Bean patterns, for example, can work only with the Java language. GoF patterns are currently available as Universal. Other patterns can be applied to any language - `pattern.UNIVERSAL.MEMBER.Stub_implementation` pattern for example. Together determines the target language for a pattern automatically, and identifies its location in the installation. See details below.

Behavior of a pattern is defined by its properties set. The *`getProperties`* method returns a `com.togethersoft.util.propertyMap.PropertyMap` instance containing a set of all the properties for the pattern.

The *SciPattern* interface defines the methods that you should implement in your pattern:

`apply` makes the pattern perform desired actions

`canApply` checks whether the pattern can be applied to the target object (objects) with the current values of pattern's properties.

`prepare` checks if it is possible to apply this pattern to the target object (objects) at all, and makes some startup preparations for the pattern. It returns `true` if everything is okay, and `false` if the pattern cannot be applied to the target object (objects) at all.

`getProperties` returns a `PropertyMap` instance containing a set of all the properties for the pattern.

Other information about pattern-related interfaces and pattern properties can be found in the main API documentation (`/doc/api` folder in your Together installation).

Using Patterns

Creating classes and links by patterns

Together makes it easy for you to apply patterns when creating classes or links. To create classes or links during modeling, you can use the following icons on the Class Diagram Toolbar:

Tool-tip	Icon	Description
Class by Pattern		Create a new class using selected pattern to define how code is generated
Link by Pattern		Create a relationship link using selected pattern to define how code is generated

Both icons launch the Choose Pattern dialog displaying the available patterns for the respective operation. You can access Together's predefined patterns plus any that you implement yourself. The dialog has an "expert" format.

To create a class or link from a pattern:

1. Click the appropriate toolbar icon.
2. If creating a class, click on the Diagram pane to display the Choose pattern dialog for classes. If creating a link, drag it from the source class to the destination and drop it to display the Choose pattern dialog for links.
3. Select the pattern you want for the new class or link.
4. If the Next button is enabled, there are properties or parameters for the pattern. Click Next to display them. (If you know the pattern and want to accept the defaults, click Finish).
5. Set properties or parameters as desired and click Finish.

Note: Users of the Microsoft JVM should note that `java.rmi` classes, required for compilation of the EJB pattern, are not present in this JVM.

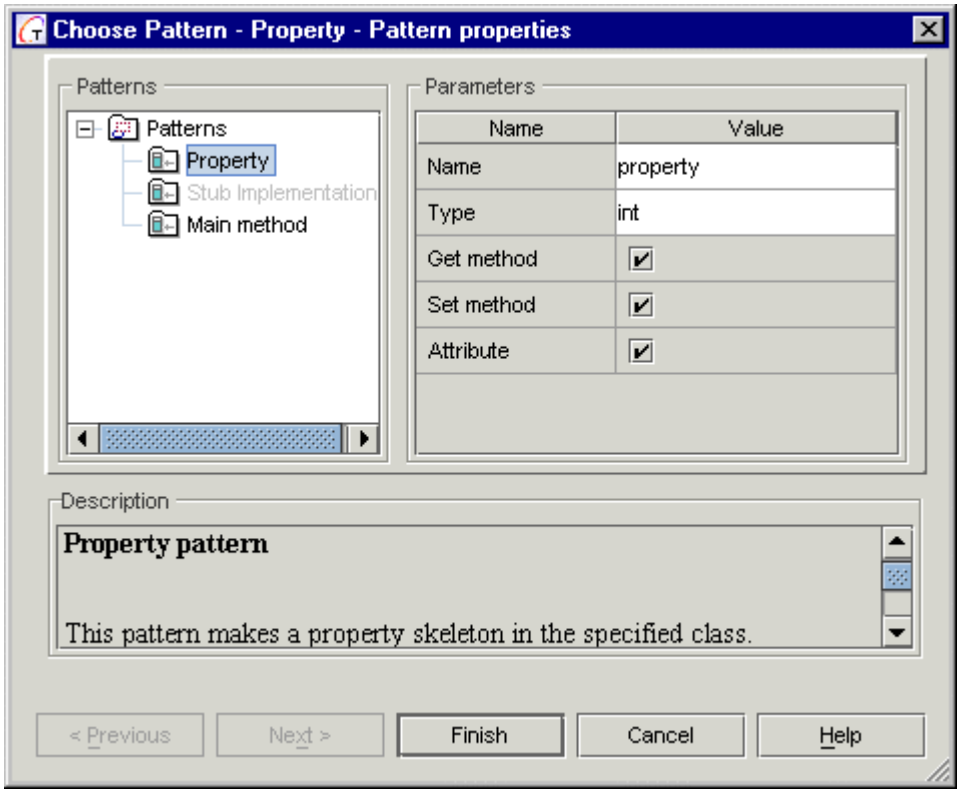
Caveat

The result of applying a pattern to a class depends on the way it was invoked. If you are going to create a new class by pattern, make sure you click on the Diagram pane, rather than on a class shape. In the latter case, the class where you click is regarded as a container class, and the newly created class by pattern is added as an inner class.

Invoking Choose Pattern command from the class speedmenu enables refactoring of the class according to the selected pattern.

Creating members by pattern

There is no toolbar button for this operation. Invoke class speedmenu and select New | Member by Pattern. Choose Pattern dialog shows up, where you can select the desired property pattern. Enter property name and type, and check the boxes for accessor methods and attributes if necessary.



Choosing a pattern for a Member

You can also apply patterns to control the way implementation code is generated for an attribute or operation.

To apply a pattern to a member:

1. Select an attribute or operation and select Choose Pattern from its speedmenu. The Choose Pattern dialog is opened displaying available patterns for members
2. Select the pattern you want for the member. The resulting code displays in the Preview box.
3. 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).
4. Set properties or parameters as desired and click Finish.

The dialog's "Member" node displays patterns that can make the selected member an attribute, an operation, or even regenerate the code for the property with get and set method. The "Link" node displays patterns that make the selected member a link.

> **Note:** The Choose Pattern command is enabled only if the selected element satisfies certain criteria that make possible regeneration without losing significant code. Thus, if an operation already has some code in its body, the command is disabled.

Refactoring with patterns

You can also refactor existing classes or links replacing an existing pattern with a better one. You can do this singly, or to multiple selected elements.

To refactor classes or links:

1. Select the element(s) in the Diagram pane. (E.g. select multiple classes).
2. Select **Choose Pattern** on the speedmenu of the selected element (or one in a group) 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 speedmenus of Class diagram elements in the Explorer.

Developing and deploying your own patterns

You can develop your own patterns and reuse them in Together. Patterns are implemented using the SCI API. To develop your own patterns you will need to study the documentation for the Together API. (For more information on the API documentation see Together Open API.) You can also study examples of mature patterns in the `$TOGETHER_HOME$\modules\com\togethersoft\modules\patterns` directory in your installation. To deploy your patterns, you must place them in the proper package under this directory. For more information and technical specifications for deployment, see the file `$TOGETHER_HOME$\modules\com\togethersoft\modules\patterns\patternsReadMe.html`.

Java Beans

Java Beans provide a powerful means of reusability. Together enables creating Java Beans from classes on the Class diagram.

To make the beans visible in diagrams, open the Options dialog, expand the *View management* node and check the option *Recognize Java Beans* on the page *Java Beans / C++ Properties*. When this option is selected, the classes with specific bean properties display as Java Bean icons in diagrams.

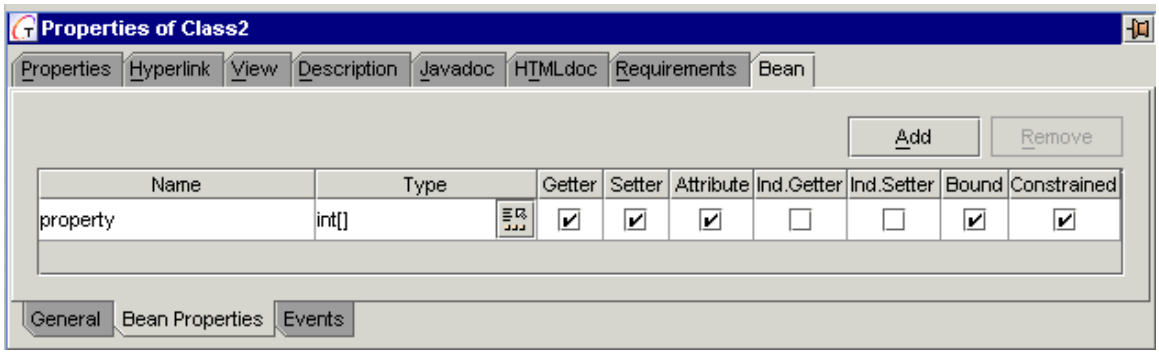
Creating a Java Bean from a Class

If you want to refactor a class into a Java Bean, open the Object Inspector and select the *Bean* page. This page contains *General, Bean Properties* and *Events* tabs.

On the *General* tab you can check the *BeanInfo* option to generate the `BeanInfo` class, and choose *Serializable* to save and restore the bean's state. Alternatively, providing persistence is full responsibility of the class itself.

On the *Bean Properties* tab you can create or delete properties using the *Add/Remove* buttons. Checking appropriate flags for the selected property adds accessor methods and indexed getter/setter methods to the bean. In addition, it is possible to create bound and constrained properties.

The *Events* tab is designed for handling the bean event sets: choose between multicast or unicast modes and create fire methods.



As soon as a bean property is added, the class involved displays as a bean icon.

Recognizing Bean Properties

The *Recognizing Java Beans* option is applicable to the entire project. However, it is possible to switch this possibility off for certain classes, or for particular methods of a class. To do this, add the following comment to the source code next to the import statements:

```
/** @notBean */
```

In the same way, the comment `/** @notProperty */` cancels recognition of a method as a bean property.

Bound and Constrained events

Java Beans may have bound and constrained events, which are not accessible for modification from the bean Inspector. If bound or constrained options are selected for a certain property, *propertyChange* and *vetoableChange* event sets add to the bean. However, these methods are not displayed in the bean icon and show read-only in the inspector (the Remove button is disabled, and *Unicast/Fire Methods* flags are not available).

On the other hand, you can easily delete these methods from the source code in the Editor pane. In this case, the content of the object Inspector changes appropriately.

Documentation Generation

Generating Project Documentation

Together delivers a wide variety of documentation generation options. Regardless of the option you choose, your system documentation stays accurate and up to date.

In this chapter you will find detailed information about DocGen commands, features of the Documentation Template Designer, and creating multi-frame documentation. In addition, useful information about the internal variables and functions is provided in the sections under DocGen and DocDesigner Reference in the Table of Contents.

Documentation Generation commands

Doc generation commands are available on the Tools menu:

- Generate HTML
- Print Documentation
- Generate using Template
- Design Template

Generate HTML documentation

HTML doc generation produces JavaDoc (tm) compatible output. The resulting documentation is net-ready, complete and up-to-date. It is quick and easy to produce and update. Selecting this command on the Tools | Documentation menu invokes a dialog window. See Generate HTML topic for description of controls.

Print Documentation

This command displays Print Documentation dialog, that allows to select the range of documents for printout, and the printout destination. You can choose between a printer and PDF file.

Generate Using Template

Use this option to generate documentation based on an existing or specially designed template. This command invokes Generate Documentation by Template dialog. It is possible to choose the desired output format. In particular, generation by template allows to produce documentation in RTF format. Doc Generation and Design module provides mature templates for various diagram types, templates for multi-frame documentation etc. You can find them under `modules\gendoc\templates` folder of your installation.

Design Template

Together provides a powerful and flexible tool to create the most sophisticated documentation templates that can be further used for doc generation. Refer to the template topics for detailed explanations and examples.

Overview of GenDoc concepts

This topic outlines the main notions of the documentation generator and designer. Template Designer allows to create user-defined templates, which can be used to generate future documentation. Template Designer produces a textual file *.tpl with html-type structure. This file contains special tags describing all template parts and elements.

Zones and Areas

GenDoc divides a generated document into 5 major zones:

Document Header
Page Header
Details Zone
Page Footer
Document Footer

The Details Zone represents all the document data and breaks it up into a sequence of document areas. When GenDoc generates a document, it produces a flow of areas and writes them sequentially into the document's stream.

Report Header/Footer are the areas created only once per document. Page Header/Footer are the areas created once per page.

Every document area in the generated document always has a prototype definition in the GenDoc template, which is called the template area. Each template area describes the data to be placed in the particular area of the report.

When GenDoc executes a template, it sets a flow of template areas by which it produces a flow of appropriate areas in the generated document. All areas in the template are organized into template sections.

Template structure

The GenDoc template contains a set of sections arranged in a particular order. Each section plays the role of a specific command to the GenDoc's processor. Thus, when a template is executed, GenDoc acts like a computer processor, but instead of machinery code commands, it "executes" the template's sections that constitute section flow.

All sections in the template are organized in a number of ordered sequences, or *section scopes*. The Detail Zone of the template represents the *root section scope*. Sections of some types may contain their own section scopes. Thus, the whole template may be represented as a tree of nested section scopes. You can see this tree in the left panel of the GenDoc Template Designer.

When processing a template at any particular moment, the GenDoc processor executes the section scopes and consequently visits every section. It starts from the root section scope of the template. If a compound section is encountered , GenDoc saves the current position in the current scope, enters the section's scope and after completing, returns to the pervious one.

Not all visited sections are necessarily executed. Some sections may have a special condition and can be skipped if the condition is not true. In addition, for any template section you can define a special *enabling condition* which can dynamically switch this section on/off. Thus, it is possible that no document areas are generated when section scope is processed. This situation is called a *section scope without output*.

Section types

Sections differ in destination and their ability to contain other sections. Nested sections are inserted in the current section and are located on a lower level than the current one. Sibling sections are parallel to the current section. They are inserted on the same level as the current section.

Static Section

Static Section is the simplest section type. It is intended for plain information output and represents an atomic unit of output in the Template Designer.

Element Iteration Section

Element Iteration Section is similar to a loop in a programming language. It provides traversing through all elements of a certain type. Static sections are used inside an element iteration to produce output. The element iteration section can contain static sections, folder sections, element property iterators, other element iterators and stock section calls.

Property Iteration Section

Property Iteration Section is used to iterate through the current model element's RWI-properties and display their values. Along with the current model element, there is a *current RWI-property* which has a definite value inside the Property Iterator's section scope. Property Iteration Section can only be used inside an element iterator and can contain stock sections, folder sections and stock section calls. It cannot contain other element property iterators of element iterators.

Actually, these section types are quite sufficient for most needs. However, to optimize the template structure template design process, GenDoc introduces additional section types:

Folder Section

Folder Section structure is similar to that of Iteration Section. It contains a section scope and may have header and footer areas. However, it doesn't change the current element inside its section scope. When GenDoc enters a Folder Section, it always passes through the scope only once. The Folder Section can be used to group several consequent sections into a separate section scope. This allows to define a common enable condition for the entire scope, and to insert header/footer areas if the internal section scope produces output.

Call to Stock Section

In order to optimize templates, some frequently used constructions (designed as Static, Iteration or Folder Sections) can be defined only once and placed into the template's stock. Such sections are called *Stock Sections*.

Once defined, a Stock Section can be called from different places of the template. It behaves like a procedure in conventional programming languages. To create a *call to a Stock Section*, the Call to Stock Section should be inserted in a section scope and adjusted to the particular Stock Section being called.

When GenDoc processes the template, the effect of this call is the same as if the called Stock Section were present just at the position where it is called from. Stock Section allows calls to itself inside its body. This feature enables a unique possibility to generate output for model elements with recursive structure (like States on State Diagram, or Inner Classes).

Call to Template

This kind of section allows to start a separate generator for a different template without terminating the current one. The root element passed to the generator is the current element of the calling template. The new generator can be started in two different modes:

- To generate a separate document. This feature is especially important for generating multi-frame HTML documentation consisting of separate HTML documents, extensively linked together.
- To generate output into the common stream of the calling template. In such a case the called template behaves like a Stock Section. (This feature is not implemented yet.)

Both features can be used simultaneously for the same called templates. This makes it possible to construct a library of templates for generating documentation for particular model elements (Class, Actor, UseCase, etc). Calling pre-designed library templates, you can quickly construct templates for more general reports (like "ProjectReport", "ClassReport", etc) intended for both multi-frame HTML output and flat RTF printable documentation.

Controls

Controls are template elements that perform actual information output. These elements can only be included in the static sections.

Label

Label represents a permanent text. For example, if you want "Class" text to be displayed before the name of each class, use a label control.

Image

This control is used to insert images in the reports. There are two possible types of images: static images (*.gif, *.jpg or other formats), and Diagrams. The latter type is specially designed to insert the current diagram in the reports.

Panel

Container element that can include other controls.

Data Control

This is the most important control type because it contains information you need to see in a report: documentation, name, stereotype, version, package, author etc. There may be two different types of data source for this control:

Element property - In this case, a name of a specific element property should be set for the control. When GenDoc executes this control it requests the "current" model element for the value of property with the specified name.

GenDoc variable - Each variable represents specific GenDoc internal information. To connect the data control with a specific GenDoc variable, the "variable" data source should be selected and the appropriate variable name should be chosen from the list of the variables available in this place.

Note: The information you can insert in the Data Control always depends on the Element Iteration area you are working in. For example, you can't have stereotypes in a Page Header's data field.

Formula

Formula control is the most powerful information output facility. Using formulae makes it possible to display data in a convenient way. For example, declaration of C++ classes and members is displayed in the java format. However, a specially constructed formula can show a correct C++ declaration.

Refer to the topic DocGen functions in formulae expressions for detailed information.

Meta Model

Metamodel describes how diagrams and elements are organized on the RWI level:

- which ShapeTypes correspond to which elements
- which elements can be contained in a particular model element
- which properties an element can have

Metamodel for GenDoc is represented by the configuration file `MetaModel.mm` under `modules\com\togethersoft\modules\gendoc\templates` folder of your installation. This file contains a tree of all possible elements and their properties. The root of this tree is a Model that contains so-called metatypes. Each metatype has a special description arranged according to the following scheme:

```
<metatype>
name=PACKAGE -->> metatype name
extends=ELEMENT -->> parent metatype name
rwi_entity=package -->> related RWI element
full_name="Package" -->> name displayed in the Template Designer
properties = -->> properties available for this metatype
    { $fullName; -->> RWI property
    %package; -->> GenDoc property
    stereotype; -->> other properties
    alias }
contained_metatypes = { PACKAGE; -->> possible child metatypes
    CLASS;
    INTERFACE;
    ASSOCIATION_LINK;
    DEPENDENCY_LINK;
    CLASS_DIAGRAM;
    STATE_DIAGRAM;
    INTERACTION_DIAGRAM;
    ACTIVITY_DIAGRAM;
    COMPONENT_DIAGRAM;
    DEPLOYMENT_DIAGRAM;
    USECASE_DIAGRAM;
    BUSINESS_PROCESS_DIAGRAM;
    ER_DIAGRAM }
</metatype>
```

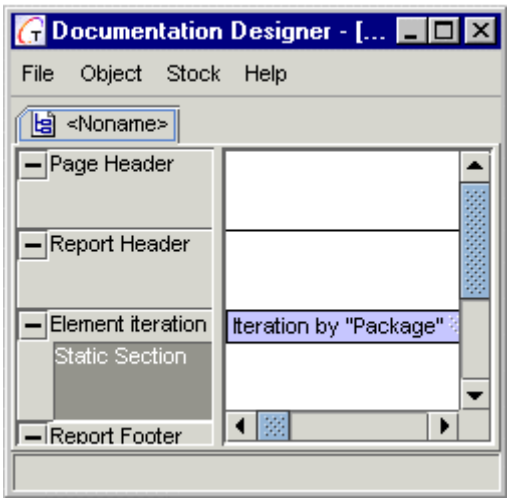
Using the Documentation Template Designer

Main menu

Documentation Template Designer's main menu contains four commands.

File provides usual file operations (creating new templates, opening or re-opening existing templates, saving changes). Options command invokes Templates Options dialog, which allows to define template settings.

Object is only enabled when en element iteration section is selected in the scope pane and provides operation with the current section. Properties command displays the properties of the currently selected section. Commands Insert Sibling / Nested section allow to insert section parallel or lower level section in relation to the currently selected one. Besides that, you can delete, copy and move the current section, and add header or footer to the section for the sake of better readability.



Stock command provides operations with stock sections.

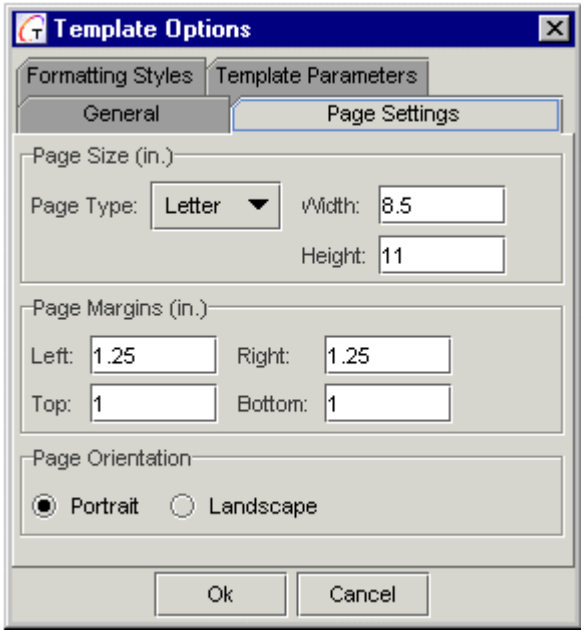
Help shows up this topic.

To start Documentation Template Designer, choose Tools | Documentation | Design Template on the main menu, or press Design button in the Documentation Generator. You can create a new template (File | New) or modify an existing one (File | Open).

Note: If you launch the Template Designer by the Design button, the project must be already opened. Template Designer will automatically open the project specified as default in the Template name field.

Template Settings

Creation of a template starts with setting up the template parameters. File | Options command invokes a dialog with a set of tabbed pages.

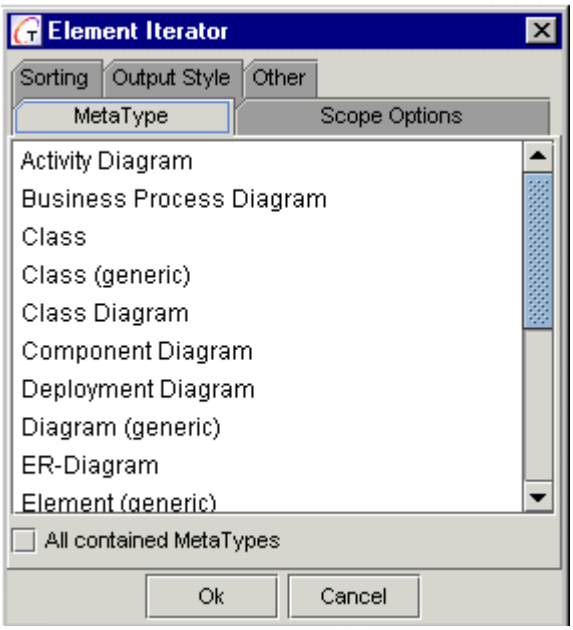


General tab allows to specify template name, select root object, and toggle the usage of headers and footers. On the *Page Settings* tab you can specify paper size and type, page orientation, and margins. *Formatting Styles* tab serves to define own styles. The notion of styles is similar to that of MS Word. This involves font style, size and color, paragraph settings, and border type.

Template elements

Element Iteration

The next step is to determine the sections of the new template. Properties command on the section speedmenu displays section properties dialog.



The most important is the *MetaType* tab, which enumerates all available metatypes. The metatypes included in this list are defined in the Metamodel file. To provide the most general iteration, it is advisable to select iteration by Package, or Diagram (generic) as the main section.

Sorting and *Scope Options* tabs allow to set up sorting and filtering modes. The most common way suggests to sort elements by type, and within a certain type sort them by name. The elements are always sorted in ascending order.

On the *Output Style* tab you can define the way of presenting information in the resulting report. Possible options are: paragraph, text or table.

Paragraph : new paragraph is created for each new element

Delimited Text Flow: all elements are included in the same paragraph. In this case delimiter type should be specified.

Table: each element occupies a separate row in the table. The title row is located in the Header section.

The tab *Other* is also very important. Proper setting of the left indent spares you from further headaches about indents for controls. The left indent is calculated in relation to the container section indent, rather than to the physical paper border. Enabling condition is a type of filter that defines if an iteration will be displayed or skipped. If a whole section needs to be skipped, check the flag *Disabled*.

Element Property Iteration

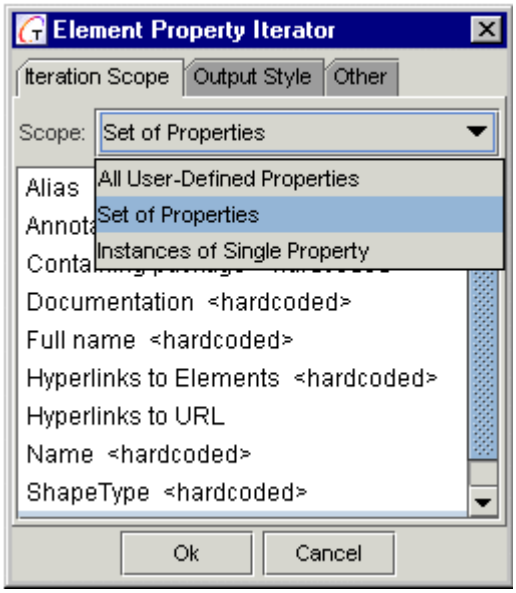
Having created an element iteration section, we can organize iteration by element properties.

The tab *Iteration Scope* allows to select the range of iteration from the dropdown list. Possible options are:

Set of Properties is the most commonly used mode. It is possible to select one or several properties to be iterated.

All User-Defined Properties mode is used for the properties that are not described in the metamodel. If this mode is selected, the dialog shows two checkboxes. The flag *Exclude already iterated properties* allows to omit properties that were already iterated for the current element. The flag *Iterate only unknown properties* includes those properties only that were not included in the metamodel.

Instances of a Single Property is a very useful tool for the properties that can have multiple values, for example, @see or @author.

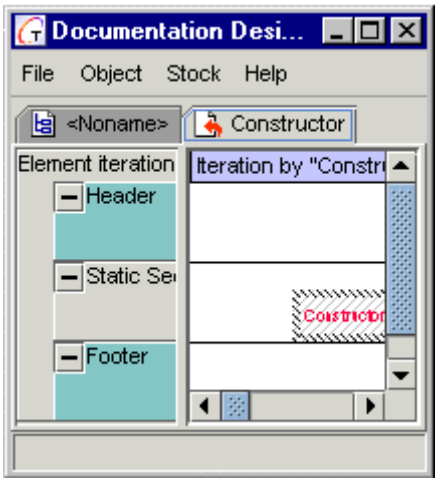


Using Stock Sections

As mentioned in the Documentation Designer Overview, stock sections help present the information that is repeated for various elements. *Stock* command of the Template Designer main menu provides operations with the stock sections.

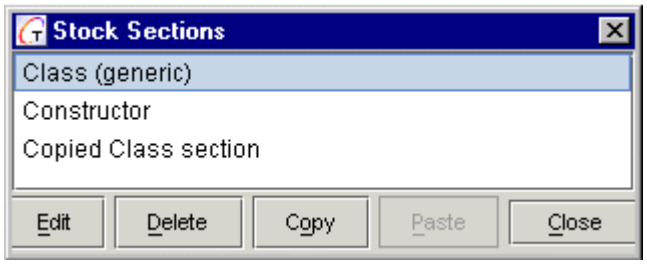
Create

To create a stock section, choose Stock | New on the Template Designer main menu. It is possible to create an element iterator, a folder section, or to insert a copy of a stock section from clipboard. Each new stock section adds a tab to the Documentation Designer window, and joins the list of available stock sections.



Edit / Delete / Copy / Paste

Stock | Edit command on the Template Designer main menu displays the list of existing stock sections.



Edit button displays selected stock section in its tab. *Delete* button removes selected stock sections from the list, and deletes their tabs from the Template Designer window.

Copy / *Paste* operations can significantly speed up the process, when you have to create similar sections. Select the source stock sections from the list and press Copy. Thus, they are placed to the clipboard. Paste adds copies to the list of stock sections with automatically generated names (for example, Class - Class1).

Call

Once the stock section is created, it can be called from the template. To do so, right-click on the section where the call should be placed, and choose Insert Nested / Sibling Section | Call to Stock Section on the speedmenu.

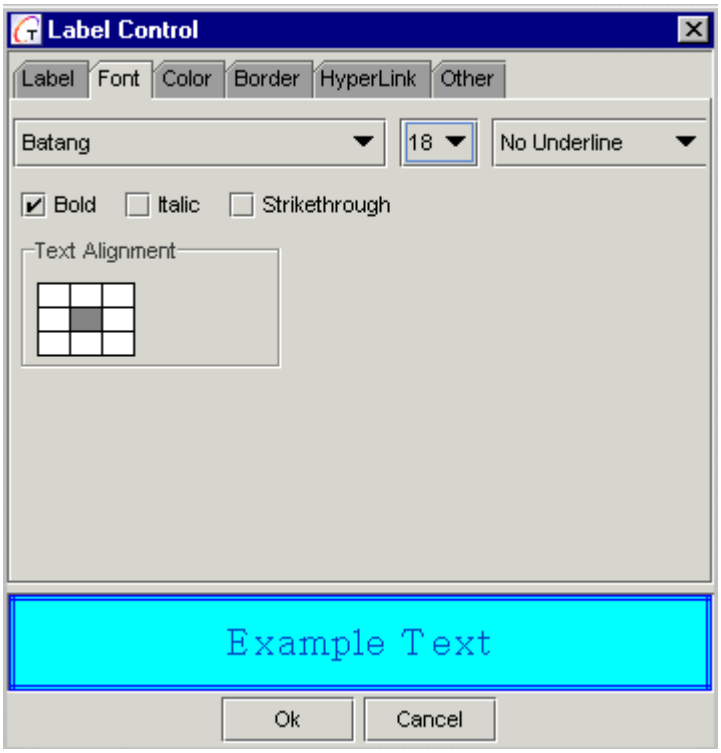
Show / Hide

There is no need to show all stock sections at once in the Template Designer Window. Right-click on the stock section tab name and select Close to hide this tab. To display a stock section, right-click on the call to the desired stock section and select *Show Stock Section* on the speedmenu.

Inserting and Formatting Data

Actual data output is performed by the Static sections where certain controls are inserted. To insert data, right click on the background of a static section, header or footer and choose *Insert Control* on the speedmenu. This command lets you create an element exactly at the position where the box appears.

Insert Control command shows up a dialog with a number of tabs, depending on the control type, that enable flexible formatting of the field. You can also right click on a data field and choose Properties command on the speedmenu.



Font

Select font family, style and size, and text alignment. All settings are applied to the entire text in the field.

> **Note**: After you finished, click Ok to apply formatting to the field

Color

Pick the desired color and using the left/right mouse button, change the text color/fill with selected color.

> **Note:** If the background color in the text window is hatched, it means that background color of this field is transparent. To specify a color, first uncheck the Transparent option.

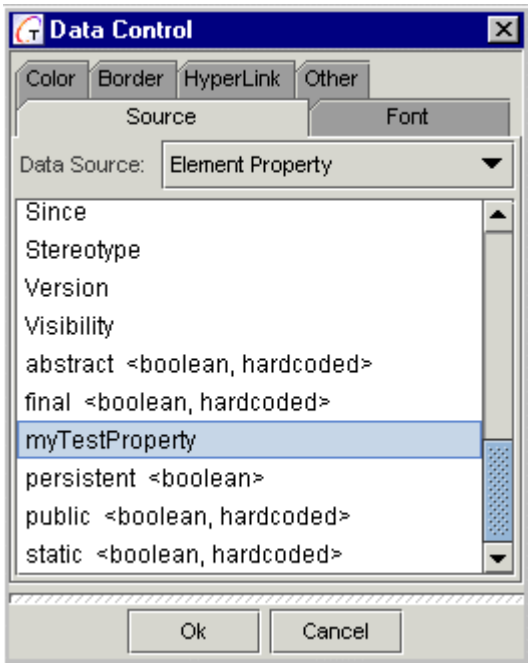
Border

Here you can set the border color and thickness of the label border lines one-by-one or all together. This features will help you when you want to highlight an element which is important for you.

> **Note:** Use the "Preview box", which reflects your selections, to synchronize all changes you make with the typed text.

Displaying Custom Properties in Generated Documentation

The information included in the generated documentation may cover object properties. One possible way to include custom properties in the generated output requires to add these properties to the metamodel of an object. Thus properties become visible in *Source* tab of the Data Control dialog. In case of need, you can use regular procedure of inserting data to iterate by certain properties.



For example, you can add a property as described in Customizing Properties Inspector section, add same property to the appropriate section of the MetaModel.mm file, and see this property in the list of available properties.

The other way doesn't require any hacking to the metamodel, but instead suggests using DG functions in formulae expressions. The function `getProperty(name)` allows to obtain any property, where it is included in the metamodel or not.

Tips and Tricks

Moving sections up/down

Every section is being inserted according to the place of your clicks and right-clicks in the Scope pane. If the current order of sections doesn't fit your requirements you may delete the section and insert it again into another place or use "Move up/down" options. These actions are available on the speedmenu of the target section

Setting attributes

To set attributes like type, color, label, etc., of an element, choose Properties on the element's speedmenu. Even if you select more than one element can set the properties of only one element.

Aligning elements

If you want to align a set of elements, e.g. make them centered or left-aligned, etc. select the elements you want to align, and choose one of the alignment *options: Left Side, Right Side, Top Side, Bottom Side*. There are three more options in this menu: *Make same width, Make same height* and *Make same size*. Use them for a more confined form of the zone.

Enabling Conditions

This option is used when the user wants to hide a section (it may be done by checking "Hidden" box) for one session only. For example: in your created template, right-click on the first *Static section* of the *Element Iteration* section, and select *Properties*. In the *Enabling Condition* field you'll see the value `< ! $fullName='' >`. It means, that the root package name (in this situation) will be not printed, though for all other packages (except "root"), this option will be ignored.

Resizing fields

To re-size a zone/area, position the cursor over its split bar. After the cursor takes the shape of an arrow, click the left button and drag and drop the split bar to its desired position.

Collapse/Extend zones and sections

After you finished working with a zone, for a better navigation in the left pane, Collapse (click on sign "+") or Extend (click on sign "-") located in the upper-left cornet of each zone or section.

Multiple items selection

To select more than one item (labels, data controls, images) from pop-up lists or sections, keep "Ctrl" key pressed.

Note: Sometime, you'll need to have Scope option set to "Set of properties".

How To Create Custom Documentation Template

Start Template Designer, create a new file and save it as `ecm.tpl`. This name displays on the template tabbed page.

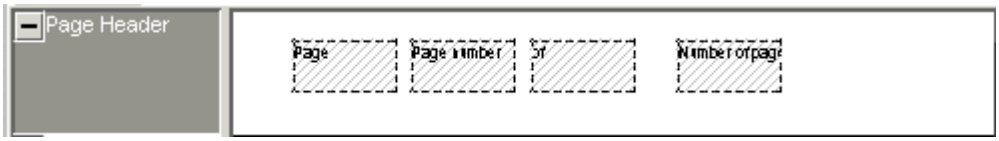
Page Header (first zone)

Right click on the white background of the Header to add elements. Select Insert Control | Label and enter "Page". This adds *Page* field, which will appear within the header zone.

Select Insert Control | Data Control. Make sure that Data Source is set to Document Field. Choose *Page Number* to prints numbered pages.

Repeat the above steps to add "of" Label and a Data Control field *Number of pages* .

> **Note:** You can insert Page Number control in either the Page Header or the Page Footer zone, but not in any other zone.



Save Changes. Don't forget use File | Save command of the main menu when you are finished making adjustments to your template. Otherwise all changes will be lost.

Report Header (second zone)

Add a Label containing a Project Name. In the Color tab of the Properties dialog choose background and text colors.

Select Insert Control | Data Control, making sure that Data Source is set to Generator's variable. Choose *Project name* and *Current date/time*.

Element Iteration (third zone)

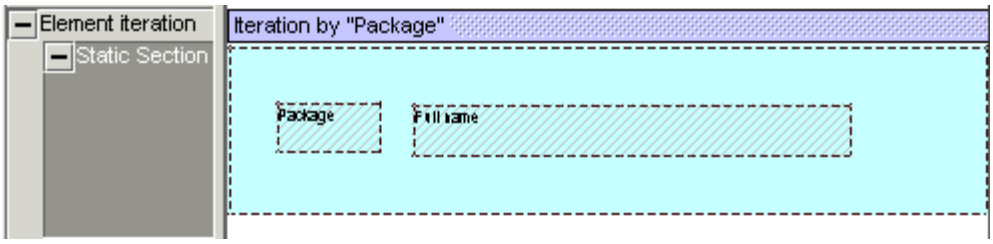
Right click on the Element Iteration zone of the Scope pane and choose Package in the Properties | Metatype. This creates an Iteration by Package, which means that all documentation will be built by packages. Number of iterations will be equal to the number of existing packages.

> **Note:** when creating a new template, "Iteration by Package" is set by default.

In the Static Section field, create a panel (Insert Control | Panel). After re-sizing the panel window and setting background color, right click within the panel, add a Label and enter the name "Package". Now your report will write the word "package" for every encountered package.

To make the report write each package name, add Data Control, select *Element Property* as the Data Source, and choose *Full Name* from the list of element properties. You can specify a Default text, in case a package has no name, or you can leave this field empty. Click Ok, re-size, then choose the position within panel when you want the data will appear. The report will write a full name for each package.

> **Note:** You will not see the root package, so this will be the first shown and without name.



Creating Property iteration section

Insert a Sibling Section for the Static Section. To do so, right click on the Static section in the Scope pane and select Insert Sibling Section | Element Property Iterator. Choose *Documentation* in the Iteration Scope tab of the Properties. This name will appear in the beginning of this section.

To add data, right click on the Static section field and select Insert Control | Data Control | Source | Documentation.

Creating Folder section

It is advisable to create a Folder Section, which will be used to include brief contents of project diagrams in your report .

To create a Folder Section, right click on the Property Iteration and select Insert Sibling Section | Folder Section. This section goes next to the Property Iteration section.

> **Note:** If you decide to create the Folder Section through Element Iteration, the section will be created behind all other sections.

Insert all sections for which you want to have brief information. All subsequent iterations should occur within the Folder Section. Hence, you can add each element iteration in two ways: by inserting a Sibling Section for the Static Section, or by inserting a Nested Section under the Folder Section. It strongly recommended to put diagrams first, before any other elements.

Suppose you created a Class Diagram first. First of all, insert a Header to delimit one element iteration from another. Click on Element Iteration in the Scope pane and select Add Header. In Header's field, insert a Label called "Class Diagram". Next, right click on the Static section's field in the Area pane and insert an image.

> **Note:** Make sure that Image type is static.

Same way add a Label called "Diagram" and a Data Control for it (Name in the Data Source | Element Property by default). For a better design, arrange element positions in this field.

Perform same steps for State, Interaction, Activity, Component, Deployment, Use Case, Business Process diagrams, and also with Subpackages, Classes, Interfaces.

> **Note:** For subpackage, class and interface, use Full Name instead Name in Data Control settings.

Delete the first Static Section within Folder Section since it's empty. For this, click on it in the Scope pane and select Delete.

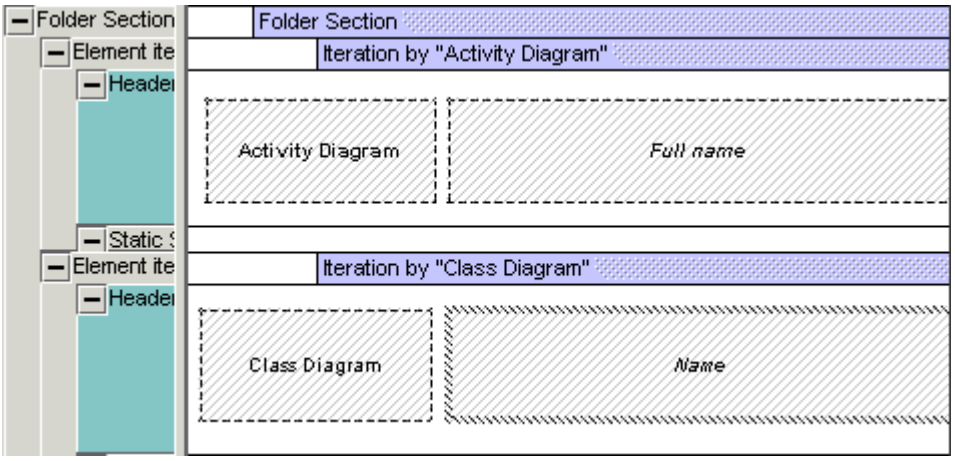
Add more Element Iterations in the Folder Section. Use Cut, Copy and Paste for increasing speed.

Iteration by Class Diagram

Within the first Element Iteration (Iteration by "Package"), after Folder Section, you have to create Element Iteration sections for each diagram and diagram element. These fields will display information about classes, attributes, and interfaces, depending of the diagram type. Our example goes as far as only the class diagram.

Relative to the last created section (Folder Section), create a new Element Iteration section (Insert Sibling Section | Element Iteration) for Class Diagram. First, add a Header section with a Label called "Class Diagrams".

Same way as for the Class Diagram in Folder Section, insert a static image, a Label called "Class Diagram" and a Data Control called "Name". Use the copy / paste options.



Creating "Diagram description" stock section

Next, you have to describe the diagram e.g. to parse it. For that, you can write a Stock section, actually the best way to insert data into template. A stock is used when you have some diagrams with similar configuration, so you don't have to design them twice. It's enough to call a Stock section containing the description.

Select Stock | New | Folder Section from the Main menu and enter "Diagram description" in the New Stock Section dialog. In the Area pane right-click on the Folder Section and select Properties. You'll see a pick-list of all possible elements used in Together where you should choose Diagram (generic), as the most common description of a diagram. New tab "Diagram Description" adds to the Template Designer window.

In the first Stock static section, insert an image with "Diagram" as Type. This will add the class diagram image into the section.

Under the first Static section, add a Sibling Section | Element Property Iterator. Right-click on the split bar and choose Properties | Documentation to select a name for this area. For data, click with the right mouse button on this area, and select Insert Control | Data Control | Documentation. Re-size.

To obtain more information about parsed diagram, you have to insert another Element Property Iterator section, which will contain your specified data. Suppose your area name is "Stereotype, Alias".

> **Note:** To select more than one element in Element Property Iterator pop-up dialog, make sure that Scope option is set to "Set of properties". The multi-select is activated by pressing "Ctrl".

For data, you have to add two corresponding fields. First, make sure that in Data Label pop-up window, Data Source option is set to "Variable". Click on "Full name of current property" to select first field, and "Value for current property" for the second. Re-size.

Add an empty Footer to better separate the stock section.

Return to the *ecm.tpl* tab and right-click on last Static Section of the Scope pane to insert a Sibling Section | Call to Stock Section. Complete pick-list of all available Stock sections shows up. Select the last created one and click Ok.

Creating "Diagram Contents Summary" folder section

Now you may want to insert a brief content of the class diagram. To add a new Folder section within this diagram, follow the same steps as for the whole project. For example, let's create the first element iteration. Name it "Iteration by package reference". First, create a Header, and insert into it a Label, named "Package nodes". Then, create a Static section and as Data control, select "Full name" of the element property.

Continue with Class, Interface, Object, Actor and Use Case. Now you have a brief description of all elements of the Class diagram so you can describe each of them, one-by-one. This is how it's done:

Creating "Iteration by "Package reference""

Create a new Element iteration section. For that, right-click on last created Folder section (in the Scope pane) and choose Insert sibling section | Element iterator | Package reference. In this section, you'll be able to describe every founded package in a Class diagram.

1. Add a Header for this new created section containing one Label Control named: "Package Node Detail". Recommendation: For a better design, choose a color and fill it. Re-size.
2. In the Static section, insert a Label Control (right-click on its area and select Insert control | Label control). Call it: "package". Then, add the corresponding Data control by selecting "Full name" in the Element Property box. Recommendation: for a better design, insert a small image at the beginning of the area.
3. In the Scope pane, click on Static section and insert one Property iteration section. This will be used for describe all package documentation so in this section area, click on Property iteration, select "Properties" and choose "Documentation". Then, insert a Data control (). Set Data source option to "Variable" and choose "Value of current property".
4. Create one more Property iteration section. In the Element properties pop-up window, select all items except Documentation, Name and Full name. Note: Make sure that Scope option is set to "Set of properties".
5. Then, add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".
6. In the Class diagram, packages may be connected with other elements through relationships. So, you need to describe all possible links (related to packages) too. Very often, those links you'll have to use in other diagrams, that's why it makes sense to create for them Stock sections. This is the optimal method, though you can choose the slowly one: Click on one Property iteration section, select Insert sibling section | Element iterator and one from those three links will appear.

Note: You have to perform these steps every time you have to add a link description. Below, we'll continue with the first method.

First, you have to create a Link description stock section which describes the basic features of a link: destination and source. Using that, you'll be able to create another stock section, for all links just by adding theirs particular characteristics to the basic ones.

Creating "Link description" stock section

1. Select Stock | New | Folder section. Enter "Link description" in the pop-up dialog.
2. In the Area pane, right-click on Folder section and select "Properties". From the pop-up list, choose "Link (generic)". Note: "generic" means, that this is the basic description for a link.
3. Right-click on section area and insert two Data controls: "Shape type of link destination" and "Name of link destination"; Note: make sure that Data source option is set to Element property.

Creating "Generalization links" stock section

1. Select Stock | New | Element iterator | Generalization link.
2. Add a Header and within it a Label. Name it : "Generalization link".
3. Insert a Call to stock section. Choose "Link description" from the pop-up box.
4. Add a Property iteration section. Select "Documentation" from the pop-up box. Then, add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

Creating "Association links" stock section

1. Select Stock | New | Element iterator | Association link.
2. Add a Header and within it a Label. Name it : "Association link".
3. Insert a "Call to the stock section". Choose "Link description" from the pop-up box.
4. Add a Property iteration section. Select all items from the pop-up dialog except these ones: "FullName of link destination/source", "Name of link destination/source", "Shape Type of link destination/source". Note: Make sure that Scope option is set to "Set of properties".
5. Add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

Creating "Dependency links" stock section

1. Select Stock | New | Element iterator | Dependency link.
2. Add a Header and within it a Label. Name it : "Dependency link".
3. Insert a "Call to stock section". Choose "Link description" from the pop-up box.
4. Add a Property iteration section. Select next all items from the pop-up: "Client Role", "Documentation", "Stereotype", "Supplier Role". Note: Make sure that Scope option is set to "Set of properties".
5. Add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

Creating "Iteration by "Class""

1. This section will be used for describing any founded class in a Class diagram. There are only three sub-sections used here:

2. A Header: within it, insert a Label Control and name it: "Class Node Detail";

3. A Static section: first, add a Label Control, named "Class". Then, insert a Data control. Select "Full name" from pop-up window.

Note: make sure that Data Source option is set to "Element property".

Creating "Class Node Description" stock section

1. Select Stock | New | Folder section. Enter "Class node Description" in the pop-up dialog.

2. In the Area pane, right-click on Folder section and select "Properties". From the pop-up list, choose "Class".

3. Right-click on Folder section in the left pane and select Insert Nested Section | Element property iterator. Then right-click in the Area pane on the created Property iteration and select Properties | Documentation. Next, right-click on this section area, and add a Data Control named "Value of current property".

Note: Make sure that Data Source option is set to "Variable".

4. Add one more Property iteration section. Click on it in the Area pane, and choose next items from "Properties": "Author", "Author's URL", "Deprecated", "Note", "See Also", "Since" and "Version".

Note: Make sure that Scope option is set to "Set of properties".

5. Add two Data controls: "Full name of current property", and "Value of current property".

Note: Make sure that Data Source option is set to "Variable".

6. In the Scope pane, delete the empty Static section, created by default by the Stock section.

7. Now you have to add all possible links for a class. Since you have created Stock sections for all of them , it will be easy to insert them. Just right-click in the Scope pane on the Folder section, and select Insert Nested Section | Call to Stock Section. Follow these steps, when adding "Generalization links", "Association Links", "Dependency Links" and "Implementation Links" stock sections.

Note: To create "Implementation Links" stock section see "Creating "Generalization links" stock section".

Creating "Iteration by "Interface""

This section is similar to "Iteration by Class" except for the Label name from the Header, which is "Interface Node Detail".

Creating "Iteration by "Object (of Class Diagram)""

1. In the Scope pane, right-click on the last created "Element Iterator" section, and select Insert Sibling Section | Element Iterator | Object (of Class diagram). Note: Notice that a Static section was created by default.

2. Add a Header with a Label Control within it. Name this Label: "Object Detail".

Creating "Object Description" stock section

1. On the Main Menu choose Stock | New | Folder section and enter "Object Description" .

2. Open Folder section properties, and from the pop-up list select "Object (generic)". Note: This is the most common description for a object.

3. In the created Static section, add an "Object" Label Control and "Name" Data Control. Note: When inserting Data Control, make sure that Data Source option is set to Element Property.

4. Right-click in the Scope pane and insert a Property Iteration section. From its Properties select "Documentation". Then, in its Static section, insert a Data Control. From the pop-up window, choose "Value of current property". Note: Make sure that Data Source option is set to "Variable".

5. Create one more Property Iteration section. From its Properties pop-up window, select all items except "Documentation" and "Name". Note: Make sure that Scope option is set to "Set of properties".

6. In the Static section, add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

7. Add a Footer.

8. Add a "Call to Stock Section". Choose "Object Description".

9. Repeat this step to add two more calls for stock sections: "Association Links" and "Dependency Links". Note: they already have been created.

10. Delete the empty Static section which was created by default (see above).

Creating "Iteration by "Actor""

1. In the Scope pane, right-click on the last created "Element Iterator" section, and select Insert Sibling Section | Element Iterator | Actor. Note: Notice that a Static section was created by default.

2. Add a Header with a Label Control within it. Name this Label: "Actor Detail".

Creating "Actor Description" stock section

1. On the Main Menu choose Stock | New | Folder section and specify "Actor Description".

2. Select "Actor" from the Folder section properties.

3. In the created Static section, add an "Actor" Label Control and "Name" Data Control.

Note: When inserting Data Control, make sure that Data Source option is set to Element Property.

4. Right-click on the Scope pane and insert a Property Iteration section. From its "Properties" select "Documentation". Then, in its Static section, insert a Data Control. From the pop-up window, choose "Value of current property". Note: Make sure that Data Source option is set to "Variable".

5. Create one more Property Iteration section. From its Properties pop-up window, select all items except "Documentation" and "Name". Note: Make sure that Scope option is set to "Set of properties".

6. In the Static section, add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

7. Add a Footer.

8. Add two calls to Stock sections: "Generalization Links" and "Communicates Links".

9. Add calls to next Stock section: "Extends Links" and "Includes Links".

Creating "Extends Links" stock section

1. Select Stock | New | Element Iterator from the Main Menu. In Title field, enter "Extends Links", and choose "Extends Link" from the pop-up list.

2. Add a Header with one Label Control. Enter "Extends links" in the Label text field.

3. Add a call to the "Link Description" Stock section.

4. Add a Property Iteration section. Click on it in the Area pane and next two items: "Label" and "Documentation". Note: Make sure that Scope option is set to "Set of properties".

5. In the Property Iteration's static section, add two Data controls: "Full name of current property", and the second one: "Value of current property". Note: Make sure that Data Source option is set to "Variable".

Creating "Includes Links" stock section

Follow the same steps as for creating "Extends Links" Stock section.

1. Add a call to "Association Links" stock section.

2. Add a "Call to Stock Section". Choose "Actor Description".

3. Delete the empty Static section, which was created by default (see above).

Creating "Iteration by "UseCase""

1. In the Scope pane, right-click on the last created "Element Iterator" section, and select Insert Sibling Section | Element Iterator | UseCase. **Note**: Notice that a static section was created by default.

2. Add a Header with a Label Control within it. Name this Label: "UseCase Detail".

Creating "UseCase Description" stock section

1. See "Creating "Actor Description" stock section". Replace "Actor" with "UseCase" where this is necessary.

2. Add a "Call to Stock Section". Choose "UseCase Description".

3. Hide Page footer and Report footer by unchecking them in Setting Resource options box.

Creating Multi-Frame HTML Documentation

Multi-Frame Documentation Basics

Multi-frame documentation normally contains:

1. Frameset file, i.e. an HTML file with only a description of the frame windows structure. Normally, this file is the starting point for viewing the documentation.
2. Set of HTML documents with actual information, designed to be viewed in the appropriate frame windows.
3. Hypertext links, some of which are adjusted for loading the referred HTML-document into the corresponding frame window

Generating multi-frame documentation involves creating a set of templates. The major one is the FrameSet Template. This is the starting point from which GenDoc begins to generate multi-frame documentation.

FrameSet Template consists of two important parts:

FrameSet Structure definition;
Template body which looks like an ordinary Document Template, but unlike the former doesn't produce any output file. Instead, it contains calls to the Document Templates, which are template set members, who produce the informative HTML documents.

Every Document Template is designed for a particular type of model elements (i.e. MetaType) for which it can produce specific documentation. For example, it can be a template specially designed for a Class or UseCase or a State on the States Diagram.

In brief, the process of multi-frame doc generation involves the following steps:

1. First, GenDoc starts with FrameSet Template (File | New | Frameset Template).
2. Next, according to the template body, GenDoc traverses the model and calls appropriate Document Template for each encountered model element. Thus, GenDoc produces separate document files for the model element.
3. Finally, when executing FrameSet Template body is completed, GenDoc produces special HTML-file with the frameset definition, that corresponds to the FrameSet Structure specified in FrameSet Template. This HTML-file has same name as the FrameSet Template and should be started whenever you wish to view generated documentation.

Creating Hypertext Links

Multi-frame documentation cannot work without hypertext links. Every hypertext link consists of two parts: *link reference* and *link destination* (or target). The link reference is associated with a particular text in the HTML document, whereas the link destination is a location in the HTML document.

Both parts have appropriate definitions in GenDoc template:

Link reference is described in the *HyperLink* tab of the template control and produces the link reference's text.
Link destination is described in *Hypertext Target* tab of the template area, and defines the document area you wish to reference by this link.

GenDoc provides two ways to bind both definitions to produce a hypertext link.

One obvious way is to assign a particular name to the link target. Then, when defining the link reference, you can specify this name (and the document file name, if the reference and

target are in different documents). This is the universal method for creating hypertext links.

However, this approach engenders some complications, since one reference / target definition produces a lot of actual hypertext links. Hence, you have to make sure that the name of generated link target is unique for every associated part of the documentation. Also, normally few Document Templates produce lots of HTML files with the names generated according to the model elements described by the document. So, when defining a link reference in a template, in most cases you must provide an expression calculating destination file name.

To avoid these complications GenDoc provides a different approach. In most cases a hypertext link is based on a particular model element. The link connects element's full documentation and the reference to it from the place where this element is mentioned. For example, documentation that describes a link on a diagram, mentions the diagram elements this link connects, and refers to complete documentation for these elements.

To make use of this idea, let us introduce the notion of *Element's Main Documentation*. Normally, this is an HTML document (or a part of some HTML file) with complete information about a model element.

To specify location of the element's main documentation, select "Start of the current element's specific documentation" check-box on the *Hypertext Target* tab of the template area properties dialog. (Later it will be explained, why this check-box designates element's "specific" documentation rather than "main".) It means that when GenDoc passes this template area and generates appropriate document area, it inserts a hypertext target tag, the name of the target being automatically generated. Then, this hypertext target's name, together with the name of the document file, is remembered as the Main Documentation destination for the current model element.

Once the location of element's main documentation is defined, you can specify hypertext references to it for any template control in the same template or in a different one:

1. Go to the *HyperLink* tab on the control's Properties dialog
2. Choose *Link to Element's specific Doc* radio-button
3. Specify *Expression for RWI-element* in the Link Settings.

The expression should return the RWI-element of the model element whose Main Documentation you make hypertext reference to. If your control belongs to the template section iterating through the model elements and you wish to create hypertext links to their main documentations, you can use a simple expression `getDGRwiElement("curElement")`, which returns the current iterated element. Expressions can be rather complicated, for example:

```
findElement(getDGRwiProperty("curPropertyInstance")-
getSubproperty("$referencedElement")).
```

FrameSet Template

As it was explained earlier the FrameSet Template is the major template of the Multi-Frame Documentation Template Set. It is intended to define the structure of frame windows for viewing documentation and to spawn Documentation Templates, which create HTML documents to be viewed in frame windows

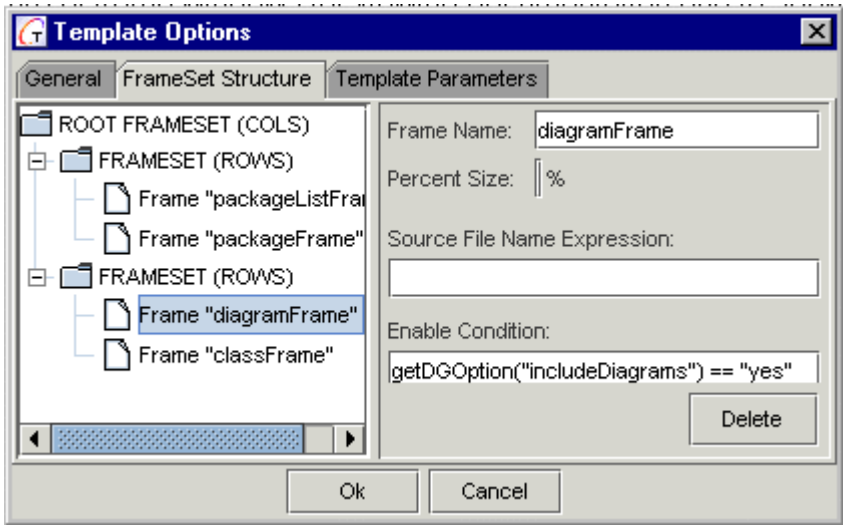
To create a FrameSet Template, choose File | New | New Frameset Template. Once created, the FrameSet Template cannot be changed into a Document Template or vice versa.

Defining FrameSet Structure

Having created the FrameSet Template, go to Template Options Dialog and choose *FrameSet Structure* tab. The FrameSet is represented as tree on the left panel. This tree can contain two kinds of nodes: FRAMESET nodes and Frame nodes.

FRAMESET node represents a container window for child windows. The child windows can be arranged in columns or rows, which is specified in Layout property of the FRAMESET node. For every child window the window size is assigned. The size is defined as percent of the parent window's space occupied by the child window.

Frame node represents a particular frame window where some HTML document can be displayed.



Frame name has to be assigned for each Frame node. This name can be specified in the reference target frame property, when defining hyper-links. Thus, the referenced document will be loaded in the frame window assigned for that reference.

Source File Name Expression can contain a name of HTML document to be initially loaded in the frame.

Enable Condition expression makes a node to be skipped or included when the resulting FrameSet HTML file is generated. This feature allows to configure frame windows structure dynamically, depending on parameters the passed to the doc generator. For example, the FrameSet considered here allows to switch on/off the frame displaying a diagram chart depending on the option "Include Diagrams" in the GenDoc launching dialog.

Linking document templates with "Call to Template" Sections

Having created FrameSet Template and defined FrameSet Structure, the next step is to describe how the actual informative documents should be created.

To do this, you have to create the template body. Template body is organized in the same way as one in an ordinary Document Template. It can contain any number of Iteration Sections (i.e. Element Iterators and Property Iterators), Folder Sections and Stock Section Calls. Prohibited are Static Section, and header / footers for Folder Sections and iterators. This limitation stems from the fact that FrameSet template's body produces no output file. Instead of Static Sections, Call to Template sections are inserted.

When GenDoc processes the template and meets a *Call to Template* section, it suspends the current template execution, loads the linked template, and processes it, which produces separate HTML document. The root element for the called template will be the current model element of the calling template. After processing of the linked template is finished, execution of the calling one resumes.

Thus, Multi-frame Documentation Template Set is organized so that the FrameSet Template describes traversing the model and visiting necessary elements, whereas the other templates are design for particular model elements.

The other possible way is to call document templates from the other document templates. In this case, the FrameSet Template will have rather simple body, consisting just a few calls to sophisticated templates that perform all other calls. This approach is possible and perhaps works faster, but the template set looks more intricate and less clear.

Important features for designing multi-frame documentation

Name of the generated document

Name of the generated document should be specified in *Output File Name Expression* property. (Please notice, this expression should return pure document name rather than the file path.) If a particular call of a document template is to be iterated many times and should produce multiple documents, the output document name should be derived from the properties of the current model element, which is valid in course of this section processing. For example, you can use expression `getProperty("$name")`.

If the *Output File Name Expression* is not specified, the name of the called template will be used for the generated document.

Output directory

Output Directory Expression property defines path to the destination directory for the generated document. This path is always relative and should be defined according to the following rules:

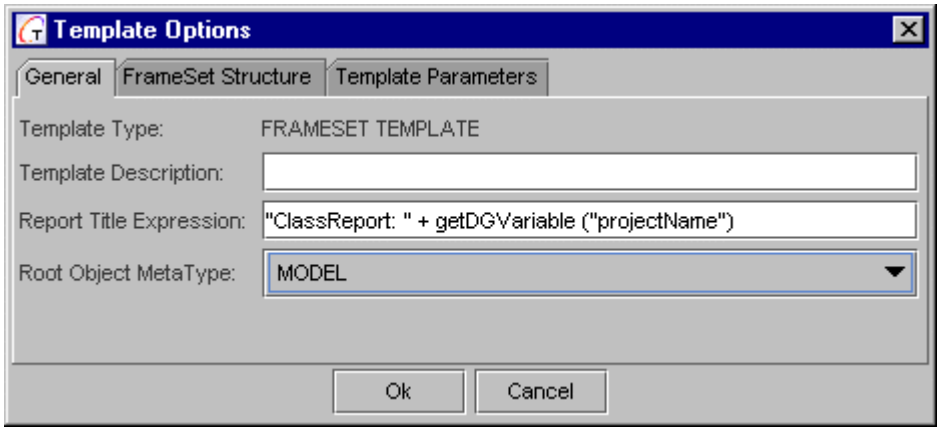
1. If the calling template is a FrameSet Template, the path is relative to the destination directory for the whole documentation
2. If the calling template is a Document Template, then the path is relative to the location of the document being currently generated by the calling template
3. The name-separator character used in the path should always be the right slash "/"
   All intermediate subfolders in the path will be created if not existed before

For example, suppose you design some documentation to be generated by a model and would like to maintain the documentation directory structure according to the structure of packages in the model. Then, you can create a template designed for a particular package, organize iteration by packages within FrameSet Template and call the package template with the path expression `replace(getProperty("$fullName"), ".", "/")`

Sample Multi-Frame Documentation Template

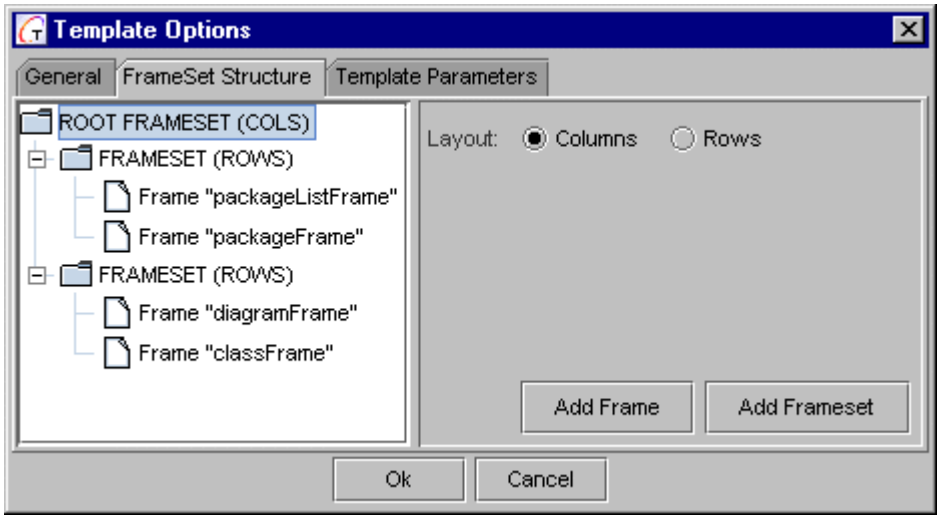
Let us study template structure of a sample documentation that presents the list of all packages and classes in a model, and detailed information about each package and class. The browser window should be divided into four frames. The upper row contains two frames with the list of packages and a diagram of the selected package. The lower row contains frames with the list of classes and detailed information for the selected class. You can find the sample under gendoc\Templates folder of your installation.

1. Start GenDoc Template Designer and load `ClassReportFrames.tpl` template.
2. Choose File | Options to invoke Template Options dialog. On the *General* tab of the dialog the root element meta-type of the template is defined as the entire model.

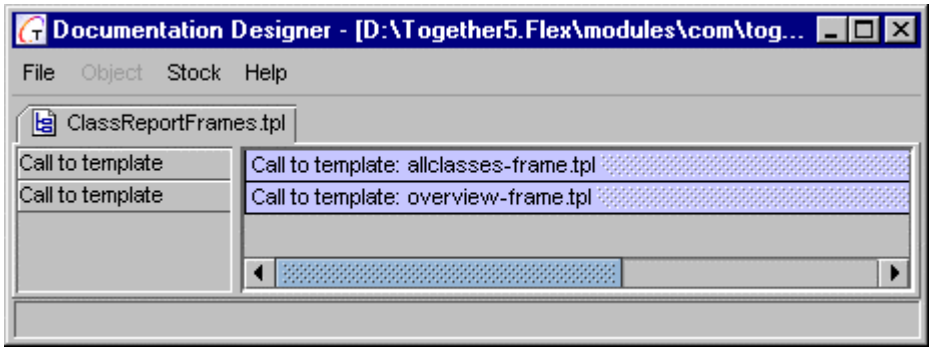


3. Select *FrameSet Structure* tab. The Frameset Structure defines the following frame windows:

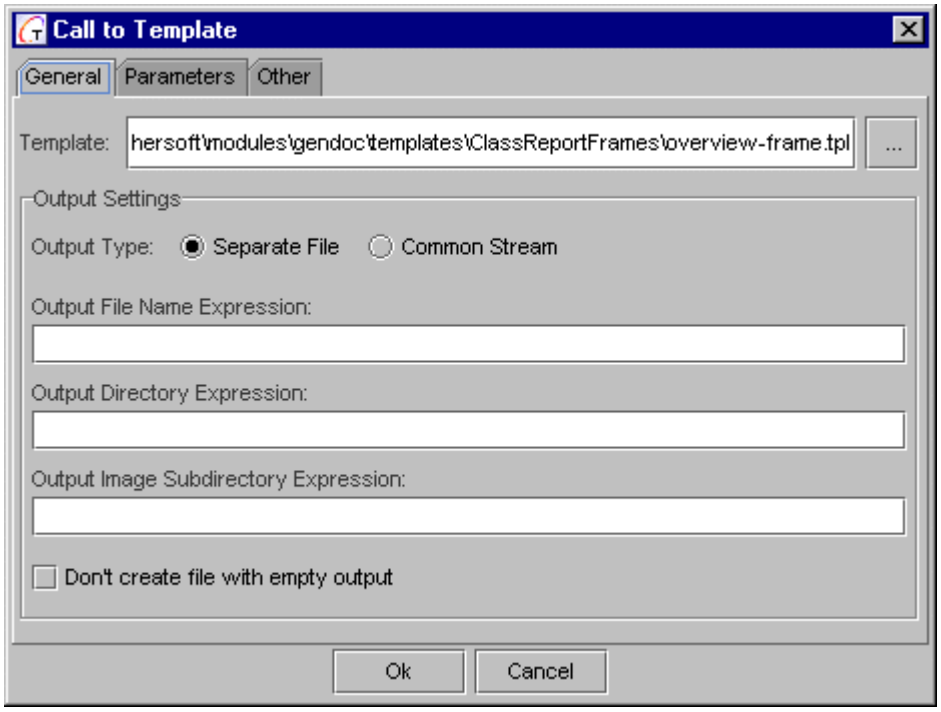
packageListFrame displays package index
packageFrame displays detailed information about selected packages
diagramFrame displays diagram for the selected package
classFrame displays detailed class information



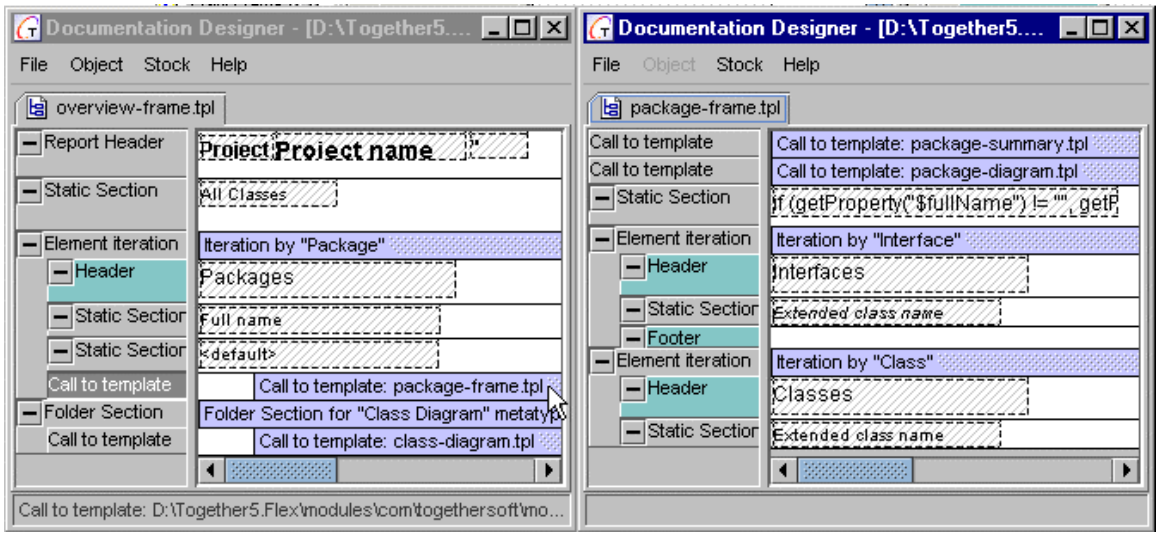
4. Root section scope of the template body consists of 2 calls to templates: `overview-summary.tpl, allclasses-frame.tpl`.



5. Right click on the Call to template: `overview-frame.tpl` and choose Properties on the speedmenu to see template settings.

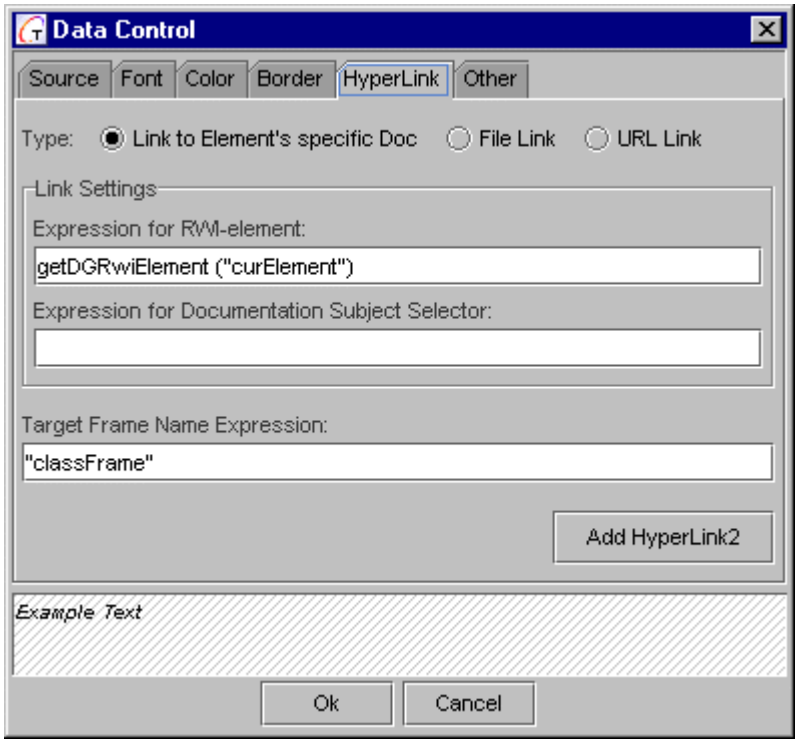


6. Choose *Open template* on the speedmenu. This opens `overview-frame.tpl` file in a new template window. This template also contains calls to other templates, each of them being available in the separate frame:



7. We can now have a look at the `package-frame.tpl` template. It contains two iteration sections: iteration by interface and iteration by class. Data control *Extended class name* produces class names with references to the relevant documentation for each class.

8. To see how the hyper-references are defined, choose Properties on the control speedmenu and choose *HyperLink* tab.



Type of the link is specified as *Link to Element's specific Doc*, which means that we refer to the documentation associated with the particular model element (the Element's Main Documentation in our case).

Expression for RWI-element field specifies which model element is considered. In our case it is the current model element of the iterator.

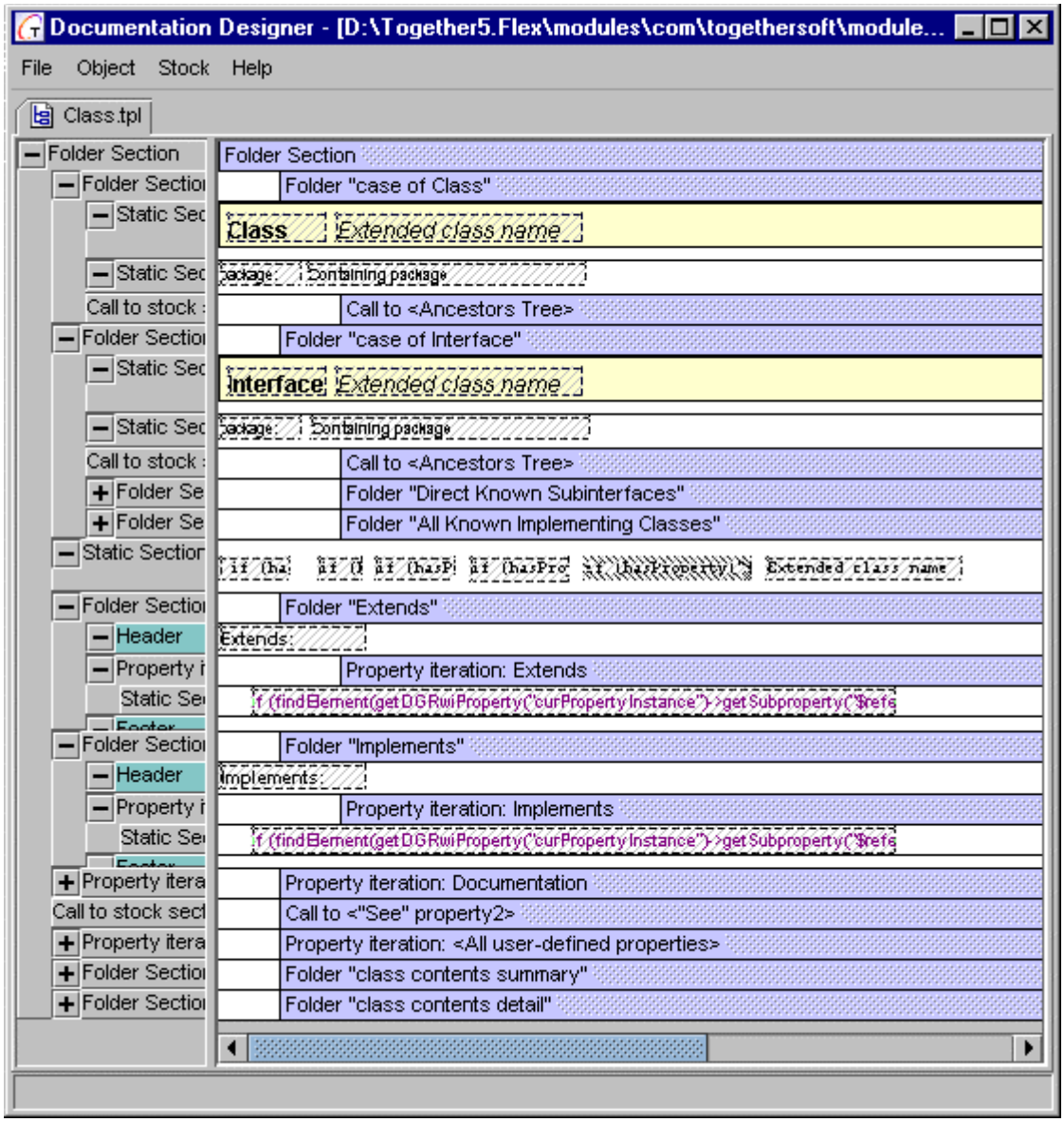
Expression for Documentation Subject Selector field is empty. It means that we refer to the Element's Main Documentation but not any other types of documents associated with that model element (see Referencing to element's "specific" documentation for more details).

Target Frame Name Expression field specifies into which frame window the document should be loaded when reference is clicked (see Assigning target frame to a Hyper-Reference for more details).

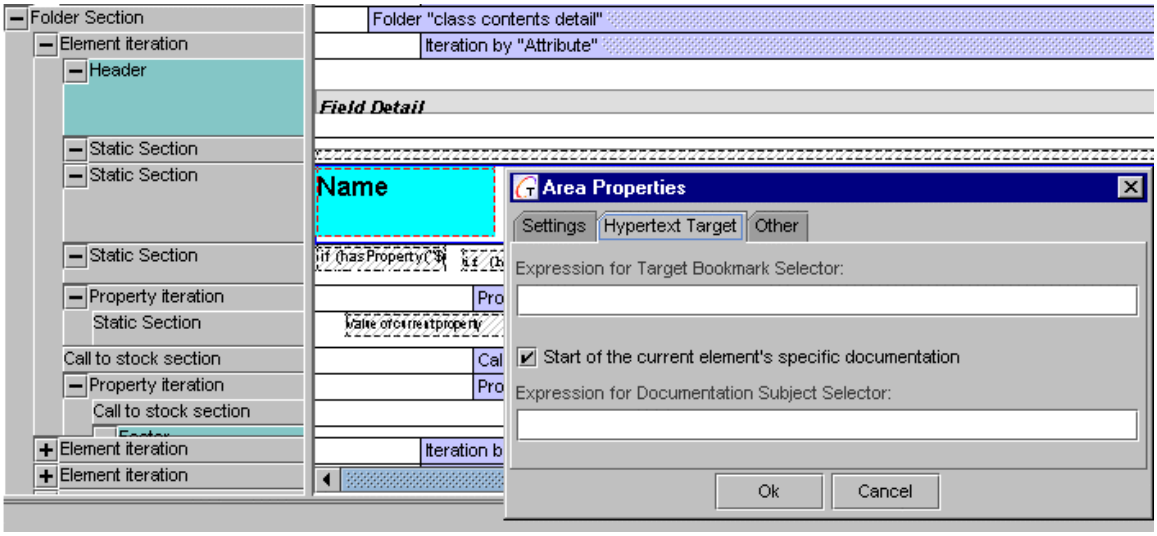
9. To study the structure of `Class.tpl` template, you have to open template `AllClasses.tpl`, and invoke *Open Template* command on the Call to template: Class.tpl section.

10. Choose File | Options to invoke Template Options Dialog. *Root Object MetaType* combobox on the *General* tab is set to "Class (generic)" meta-type, which means that template is designed to generate output file for the elements pertaining to the "Class (generic)" meta-type only.

11. The structure of this template is rather complicated. However, now we are concerned about the notion of Element's Main Documentation as it is applied here. Main documentation of a class is a file generated by `Class.tpl` template. Thus, a certain template area should be marked as the starting point of the Main Documentation.



Expand Folder section "Class Contents detail", invoke Area Properties dialog on the Static Section "Name" and choose *Hypertext Target* tab. The checkbox "Start of the current element's specific documentation" is selected. It means that this point is the target of hyper-references defined in the `ClassReportFrames.tpl` template.



Creating Hypertext Links (advanced)

Assigning target frame to a Hyper-Reference

By default, if no target frame for a hyper-reference is defined, the referenced document is loaded into the same frame window, where the current document is displayed. It is possible to change this behavior and direct the referenced document to a different frame. To do this, specify the *Target Frame Name Expression* property in the hyper-reference definition. The expression should return the name of one of the frame windows defined in the FrameSet Structure.

Referring to element's "specific" documentation

In the discussion above it was explained how to simplify creating hypertext links using the notion of element's **Main Documentation.** The useful advantage of this approach is that GenDoc makes by itself all necessary calculations and markups for the hyper-references defined in such a way.

However, sometimes along with the main documentation, it might be necessary to provide hyper-references to some 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 may be a different HTML document with only a list of all classes in this package. This special document can be displayed in a separate frame window as an index for this package. When the package is selected on a diagram (or in some more general index), the file containing package's index should be loaded and displayed in the appropriate frame.

To provide hyper-references to different documentation locations generated by the same model element, you can mark each location with the appropriate *Documentation Subject Selector*. All steps are the same as when you define location of element's main documentation: Invoke properties dialog for the template area starting the location. Then, choose *Hypertext Target* tab, select *Start of the current element's specific documentation* check-box, and enter an expression calculating the documentation subject selector.

After this, you can define hyper-references to such specific element's docs same way you do it for element's main documentation, but in addition, in the *Link Settings* you should specify *Expression for Documentation Subject Selector*, that calculates the subject selector of the element's specific documentation.

Thus, in our previous example, to make hyper-references to the package's index document we should mark the first area of the template for this document as the "Start of the current element's specific documentation" with the subject selector, say, "packageIndex", and then, provide this subject selector when we define hyper-references to this document.

> **Note:** It can seem that Main Documentation of an element is but an element's specific documentation with empty subject selector. This is not quite so, because only element's Main Documentation can be referenced from JavaDoc hyper-references (e.g. @link JavaDoc tag) imbedded in the text returned by some RWI-properties. You cannot specify subject selector for these references!

Hyper-Links from Image Elements of a Diagram

To include an image of a diagram in the generated document, put Image Control (image type = diagram) in the appropriate template area. Template area should be located so that the current model element represented one of the diagrams contained in the model, when GenDoc processes it.

Same as the other controls, Image Control has *HyperLink* tab. When control generates an ordinary image, this tab specifies an ordinary hyper-reference associated with it. However, when Image Control specifies the image of a diagram, the same definition in *HyperLink* tab can create hyper-references for all model elements depicted on the diagram.

To adjust this, all expressions in the hyper-reference definition should be relative to RWI-element returned by the call `getDGRwiElement("diagramMapElement")`.

When GenDoc generates the image of a diagram, it creates image map, which includes all model elements depicted on the diagram. While doing this, it iterates through diagram elements, substitutes `diagramMapElement` variable with every diagram element, calculates hyper-reference for it, and then, inserts it into the image map.

Creating compound Hyper-References

When you design the multi-frame documentation, you may find that sometimes, when clicking on a hypertext reference, it would be nice to reload two HTML documents into different frames simultaneously. This feature may be useful when, for example, diagram element represents a package (or a shortcut to a different diagram). Clicking on this element would load image of the new diagram represented by this element in the place of the previous one. At the same time, some other frame would be updated with the document associated with this new diagram (for example, alphabetically sorted index of elements contained in it).

To program such behavior an additional hyper-reference should be defined along with the primary hyper-reference. To do this, click *Add HyperLink2* button on the *HyperLink* tab. *HyperLink2* tab adds to the control Properties dialog, where you can specify the second hyper-reference.

JavaDoc Hyper-References

JavaDoc References (or JDRefs) are the expressions provided with certain JavaDoc tags (such as {@link}, @see, etc.) used to define hyper-text references inside documentation text ({@link}), and with some other documenting tags. DocGen enables converting JDRefs into real hyper-text links.

Each JDRef should conform to some rules described in JavaDoc documentation provided with any JDK. There are 3 types of JavaDoc references.

1. "element" reference is the type that refers to an element of the model (method, attribute, class, package).
   General form: `package.class#member label`
   where `package.class#member` identifies the referenced model element; and `label` is optional text to be displayed with the hyper-link (if omitted, the name of the referenced element is displayed).
   DocGen can convert each element reference into a real hyper-link basing on the notion of Element's Main Documentation. Thus, if such conversion is specified, each element JDRef is replaced with a hyper-link to the main documentation of the referenced element.
2. "url" reference represents a hyper-text link to a relative or absolute URL.
   General form: `<a href="URL#value">label</a>`.
3. "text" reference has the form "string" (i.e. a text string in double-quotes). Actually, such reference doesn't represent any hyper-link and may have only informative meaning.

How to convert JDRefs into hyper-links

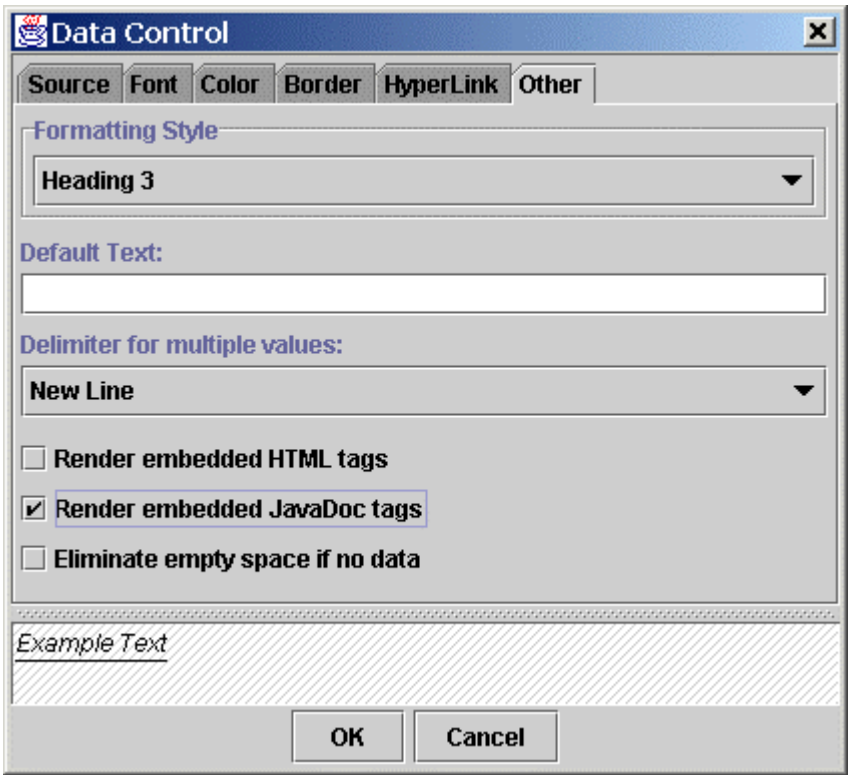
A JDRef appears in one of two forms:

1. inside `{@link}` tags imbedded in documentation text (it is the value of `$doc` property and the values of some other Javadoc element's properties)
2. as the value of some Javadoc element's properties (e.g. `see` property)

DocGen provides appropriate techniques for both cases.

Converting {@link} tags

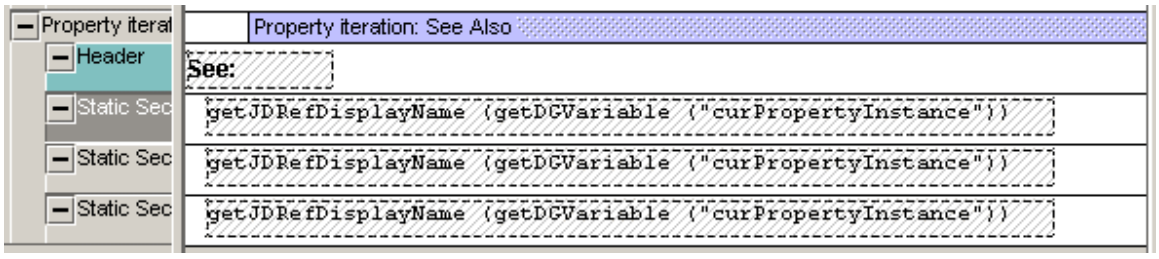
Since any documentation text can be printed only by one of the text controls (i.e. Label Control, Data Control and Formula Control), all you have to do to automatically convert `{@link}` tags is to select *Render embedded JavaDoc tags* check-box in the properties of the appropriate control.



Converting element's property value

If a JDRef is a value of an element's property, its conversion into a hyper-link is more complicated. It requires using the functions getJDRefType(), getJDRefDisplayName(), getJDRefElement(), getJDRefURL().

The following example shows how to display the value of `see` property with any and all hyper-links involved.



Since `see` property can have multiple values, it is advisable to use the Property Iterator, which iterates through the instances of the property `see`. The iterator's Output Style is *Delimited Text Flow,* with comma delimiter. Inside the Property Iterator there are 3 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 JDRef.

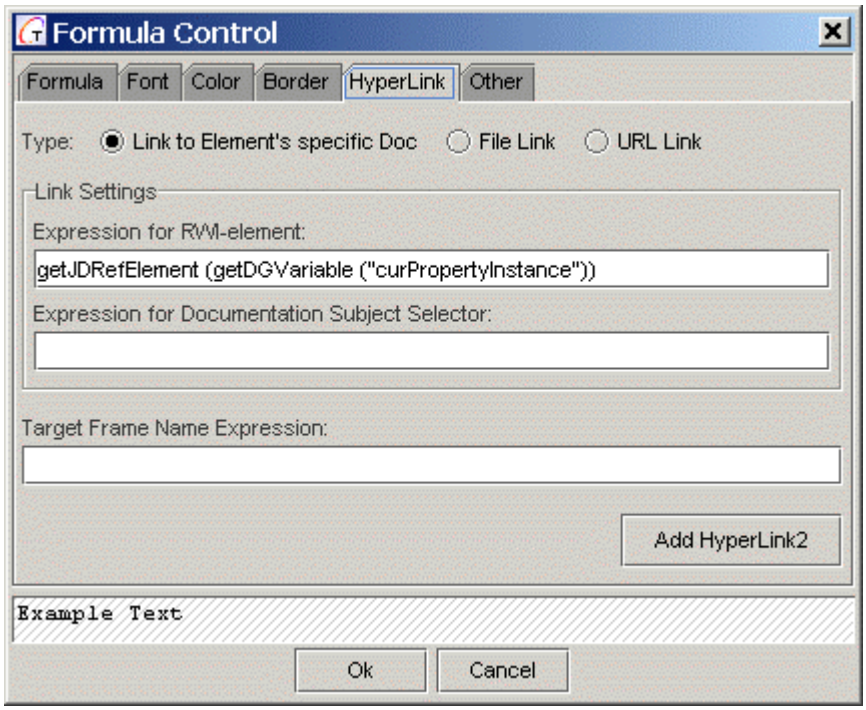
To activate the first Static Section for "element" type JDRefs only, the following Enable Condition is used:

```
getJDRefType(getDGVariable("curPropertyInstance")) ==
"element"
```

Section area contains the only Formula Control that produces a text, generated by the following expression:

`getJDRefDisplayName(getDGVariable("curPropertyInstance"))`.

This is the text to be displayed in the documentation. To create a hyper-link to the referenced element, choose *HyperLink* tab in the properties dialog of this control, and fill in the field *Expression for RWI element* as shown below:



Function getJDRefElement() returns an RWI-element referenced by JDRef. As you can see, the link's target will be the element's main documentation.

The second Static Section is activated when JDRef's type is "url". It is similar to the first one with the only difference that hyperlink in Formula Control 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, because since such a JDRef doesn't refer to any element, the function getJDRefElement() always returns null which produces no hyperlinks.

DocGen and DocDesigner Reference

When using the Documentation Designer to develop custom documentation templates for Documentation Generator building block (DocGen), you have to reference the internal variables and functions to specify formulae expressions, and provide section flow control.

DG Internal Variables

When the DG (Documentation Generator) executes a template and generates a report, it produces some specific internal information, which may be interesting to include in the report. This includes *project name*, *current date/time* etc.

Moreover, there are special internal temporary data that appear when DG executes some particular parts of the template.

DG variables enable access to this information and its insertion in the report. Each variable has specific name and represents some particular kind of internal DG information available at any particular moment.

Internal variables are not all accessible at any instant. Most of them appear only in special areas or inside special sections.

DG variables belong to one of the following types: **`String, RwiElement, RwiProperty.`** Access to these variables is provided by appropriate functions in formula expressions:
getDGVariable, *getDGRwiElement*, *getDGRwiProperty*.

Variable	Availability	Accessible via
curItemNo : **`String`** The current iteration item number (starting with 1)	inside any *Property Iterator* and *Element Iterator*	getDGVariable
curPropertyName : **`String`** Name of the current property	inside *Property Iterator*	getDGVariable
curPropertyFullName : **`String`** Full name of the current property (specified in DG MetaModel File)	inside *Property Iterator*	getDGVariable
curPropertyType : **`String`** Type of the element property. Since RWI-interface doesn't provide property types, they should be specified in DG MetaModel File *Possible values*: **`"string"`**, for `String` property; **`"boolean"`**, for `boolean` property	inside *Property Iterator*	getDGVariable
curPropertyValue : **`String`** value of the current property	inside *Property Iterator*	getDGVariable

Variable	Availability	Accessible via

curPropertyInstance : **RwiProperty**
 	
The *RwiProperty* object of the current property instance.
 	
This variable is useful when you have to get a subproperty of the current property instance.
 	
For example:
 	
Let the current model element be a **class** node, and you need to list information about all **interfaces** implemented by this class.
 	
You have to create a template section that iterates by instances of *IMPLEMENTS* property of the current **class** element.
 	
After that, within this iteration section you can use *curPropertyInstance* variable to access the subproperty *REFERENCED_ELEMENT* , which allows to obtain all information about the implementing class.
 	
Let one needs to get the **full names** of the implemented interfaces. Then the required expression should be:
 	
`findElement(getDGRwiProperty ("curPropertyInstance")->`
 	
`getSubproperty("$referencedElement"))->`
`getProperty("$fullName")`
 	
`See also DG Functions:`
 	
`getDGRwiProperty, getSubproperty,`
 	
`findElement, getProperty`
	inside *Property Iterator* while iterating by instances of the specified property	getDGRwiProperty

curPropertyInstance : **String** value of the current property instance	inside *Property Iterator* while iterating by instances of the specified property	getDGVariable
curElement : **RwiElement** the current model element	inside *Element Iterator*	getDGRwiProperty
prevElement : **RwiElement** previous element in the current iteration scope. *Possible values*: **null,** if it is the beginning of the scope	inside *Element Iterator*	getDGRwiProperty
diagramMapElement : **RwiElement** This variable should be used to create hyper-links from image elements of a diagram chart.	Inside *Image Control*	getDGRwiElement

Variable	Availability	Accessible via
projectName : **`String`** The Project name	in **report** / **page** **header** / **footer** areas	getDGVariable
nowDateTime : **`String`** The current date/time	in **report** / **page** **header** / **footer** areas	getDGVariable
outputFormat : **`String`** returns output type of the generated documentation. Use this variable to control behavior of your templates depending on the output format type selected for the generator. *Possible Values*: **`"RTF", "HTML", "TXT"`**	in any place	getDGVariable
reportScope : **`String`** shows the specified report scope. *Possible values*: **`"all_model"`** - the scope is the whole model **`"current_package"`** - the scope is the current package only **`"current_package_recursive"`** the scope is the current package with subpackages **`"current_diagram"`** - the scope is the current diagram only	any place	getDGVariable
stockParam : **`String`** parameter of the *stock section call*	inside stock sections	getDGVariable

DG functions in formulae expressions

This chapter gives brief description of the major functions used in the doc generation module.

getDGVariable

`String getDGVariable(String variableName)`

returns the specified DG variable of **`String`** type

Parameter: name of the variable

Returns: value of the variable or an **empty** string, if the variable is not defined in the given place

---

getDGRwiElement

`RwiElement getDGRwiElement(String variableName)`

returns the specified DG variable of `RwiElement` type

Parameter: name of the variable

Returns: value of the variable or **`null`**, if the variable is not provided in the given place

---

getDGRwiProperty

`RwiElement getDGRwiProperty(String variableName)`

returns `RwiProperty` type DG variable with the specified name

Parameter: name of the variable

Returns: value of the variable or **`null`**, if this variable is not in place

---

getDGOption

`String getDGOption(String optionName)`

returns the specified DG option.

Parameter: name of the option

Returns: value of the option or empty string, if such option is not defined

Features:

The option can be specified for an object of Report Generator (descendant of the class **`..gendoc.docgenerator.Generic.GnrReportGenerator`**, for example: class **`..gendoc.docgenerator.txt.TXTReportGenerator`**) using the method: *`addReportOption (String optionName, String optionValue)`*

Default values for some options can be defined in the template file. This definition persists even though TemplateDesigner subsequently modifies the template. However, method `addReportOption` overwrites the options values.

Example:

default values for the options inclSubpackages", "inclDoc", "DTLAdapter"

```
DEFAULT_OPTIONS={inclSubpackages='yes';inclDoc='yes';DT
LAdapter='com.togethersoft.modules.doorslink.DTLAdapter
'}
```

---

getParam

```
String getParam(String paramName)
```

Returns value of specified template parameter.

Parameter: parameter name

Warning:

The requested parameter should be declared in the *Template Parameters* tab of Template Properties. If parameter declaration is not defined, call to this function will cause an error message and stop the generator.

Since: Together 5

---

invokeForName

```
String invokeForName(String className, String
methodName)
String invokeForName(String className, String
methodName, String param1)
String invokeForName(String className, String
methodName, String param1, String param2)
```

invokes specified method of the user-provided class.

Parameters:

className

fully-qualified name of the user-provided class. This class should not be abstract.

DG creates an instance of `className` class and calls the method `methodName` with this instance. Note that this instance object is created for each entry of `invokeForName`call within each particular expression of template where this function is used.

However, the object is created only during the first call from such an entry, and will be used for the next calls, unless parameter `className` is changed.

methodName

name of the method in the class to be executed. The method should have the following signature:

```
String methodName (..gendoc.api.GenDocContext)
```

Parameter is an instance of **`..gendoc.api.GenDocContext`**. Class **`..gendoc.api.GenDocContext`** provides the following methods:

RwiReference getRwiReference

returns `RwiReference,` if the current DG iteration element is an RWI-reference within a diagram. Otherwise the method returns `null.`

RwiElement getRwiElement

always returns the `RwiElement`. If the current DG iteration element is an RWI-reference then returned element is `rwiReference.getElement()`. Otherwise, returned element is the current DG iteration RWI-element.

String getParameter1()
returns the value of the **first** optional parameter passed to `invokeForName` function, or **null**, if the parameter is omitted.
String getParameter2()
returns the value of the **second** optional parameter passed to invokeForName function, or **null,** if the parameter is omitted.
The method `methodName` should return `String` value it calculates.

Returns: value calculated by the user-provided method `methodName.`

---

getContainingDiagram

```
RwiDiagram getContainingDiagram()
```

returns the RWI-diagram containing the primary reference to the current element. Possible call in an expression is

```
rwiElement-> getContainingDiagram()
```

Returns: RWI-diagram containing the primary reference to the current element

---

isDiagram

```
boolean isDiagram()
```

Tests if the current RWI-element is a diagram. Call this function to test any RWI-element accessible in your expression.

For example:

```
rwiElement->isDiagram()
getDGRWIElement("diagramMapElement")->isDiagram()
```

This function may be useful when you design a Multi-Frame documentation and need to program some special behaviour when clicking hyper-links. For example, if a hyper-link references to a diagram you may want when clicking it to reload one frame with a document describing the diagram and another frame with the graphic chart of this diagram. Whereas, if the hyper-link references to any other model element only document frame should be reloaded. See also: Creating compound Hyper-References.

Returns: **`true`**, if the element is a diagram; **`false`** otherwise

---

isImported

```
boolean isImported()
```

Checks if the current element in the diagram is presented by a shortcut.

Returns: **`true`**, if the element is a shortcut; **`false`**, if the element is not a shortcut

---

getSubproperty

```
String getSubproperty(RwiProperty rwiProperty, String subpropertyName)
```

The function returns the value of subproperty `subpropertyName` contained in the RWI-property `rwiProperty`. See description of *curPropertyInstance* DG variable for an example of using this function. Possible call is:

```
rwiProperty->getSubproperty(subpropertyName)
```

Parameters:

rwiProperty the element property

subpropertyName the name of its subproperty

Returns: value of the specified subproperty

---

hasSubproperty

```
String hasSubproperty(RwiProperty rwiProperty, String subpropertyName)
```

Checks if the RWI-property `rwiProperty` contains the subproperty `subpropertyName`.

Possible call is: `rwiProperty->hasSubproperty(subpropertyName)`

Parameters:

rwiProperty the element property

subpropertyName the name of subproperty to be checked

Returns: **`true`**, if the property has the specified subproperty; **`false`** otherwise

---

getJDRefType

```
String getJDRefType(String jdref)
```

Returns type of the JavaDoc Reference specified as the parameter.

Returns: "element", if `jdref` references to a model element, i.e. if it has the form `package.class#member label`

"url", if `jdref` references to a URL, i.e. if it has the form: `<a href="URL#value">label</a>`

"text", if `jdref` has the form "string"

Since: Together 5

---

getJDRefDisplayName

```
String getJDRefDisplayName(String jdref)
```

Returns a text to be displayed in place of the specified JavaDoc Reference.

Returns: if `jdref` is an "element" reference (i.e. 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 (i.e. it has the form: `<a href="URL#value">label</a>`) the returned text is `label`.

if `jdref` is has the form "string", the returned text is `string`.

Since:Together 5

---

getJDRefElement

```
RwiElement getJDRefElement(String jdref)
```

If the specified JavaDoc Reference is an "element" reference (i.e. 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`.

Since: Together 5

---

getJDRefURL

```
String getJDRefURL(String jdref)
```

If the specified JavaDoc Reference is a "url" reference (i.e. it has the form: `<a href="URL#value">label</a>`) the function returns the text `URL#value`; otherwise, returns an empty string

Since: Together 5

---

findElement

```
RwiElement findElement(String uniqueName)
```

passes the call to the `RwiModel.findElement()` method which finds an element by its unique name.

Parameter: string with the unique name of an RWI-element that needs to be found

Returns: an element found by its unique name

---

getCodeElement

```
Object getCodeElement(RwiElement rwiElement)
```

Passes the call to `rwiElement.getCodeElement()` method declared in `com.togethersoft.openapi.rwi.RwiElement` interface.

This function is used in the template expressions together with one of the following functions: `findMember()`, `findNode()`, `findLink()`, `findPackage()`.

Since: Together 5

---

getCodeElements

```
Enumeration getCodeElements(RwiElement rwiElement)
```

Passes the call to `rwiElement.getCodeElements()` method declared in `com.togethersoft.openapi.rwi.RwiElement` interface.

This function is used in the template expressions together with one of the following functions: `findDocumentedMember()`, `findDocumentedNode()`, `findDocumentedLink()`, `findDocumentedPackage()`. These functions may be helpful when you need to provide hyper-links from some specific elements on a diagram chart.

For example, if you have set *Recognize Java Bean / C++ properties* option in Together's View Management, each JavaBean/C++ property is presented by a single element on

a class diagram, whereas actually, it consists of 3 elements: property's attribute, and setter/getter methods. When you generate the documentation for such a class you will get for every JavaBean/C++ property all those 3 elements documented (or at least, docs for accessor methods if you have specified to skip private members). Corresponding element on the diagram chart associates with a certain RWI-element, and you can obtain this RWI-element via `diagramMapElement` variable (**see also: create hyper-links from image elements)**. But actually, this RWI-element is a kind of a proxy. It will not be identical to any of those 3 elements your JavaBean/C++ property consists of, those elements which you can see in Java/C++ code and which will be documented by template's iterators.

Thus, in case of JavaBean/C++ property you cannot directly use the RWI-element representing it on the diagram to establish a hyper-link to anything contained in the generated documentation. Instead of this, you should use the following expression: `findDocumentedMember(getCodeElements(getDGRwiElement("diagramMapElement")))`.

Function `findDocumentedMember()` returns one of the RWI-elements associated with the JavaBean/C++ property and which is definitely presented in the generated documentation.

Since a diagram contains ordinary elements as well, your expression for diagram hype-links connecting RWI-element should be a bit more complicated:

```
if(getDGRWIElement("diagramMapElement")-
>hasPropertyValue("$shapeType","BeanProperty"),
findDocumentedMember(getCodeElements(getDGRWIElement("d
iagramMapElement"))),
getDGRWIElement("diagramMapElement"))
```

Since: Together 5

---

findMember

```
RwiElement findMember(Object codeElement)
```

Passes the call to `com.togethersoft.openapi.rwi.RwiModel.findMember()` method. This function should be used together with function `getCodeElement()`.

Since: Together 5

---

findNode

```
RwiElement findNode(Object codeElement)
```

Passes the call to `com.togethersoft.openapi.rwi.RwiModel.findNode()` method. This function should be used together with function `getCodeElement()`.

Since: Together 5

---

findLink

```
RwiElement findLink(Object codeElement)
```

Passes the call to `com.togethersoft.openapi.rwi.RwiModel.findLink()` method. This function should be used together with function `getCodeElement()`.

Since: Together 5

---

findPackage

```
RwiElement findPackage(Object codeElement)
```

Passes the call to `com.togethersoft.openapi.rwi.RwiModel.findPackage()` method. This function should be used together with the function `getCodeElement()`.

Since: Together 5

---

findDocumentedMember

```
RwiElement findDocumentedMember(Enumeration
codeElements)
RwiElement findDocumentedMember(Enumeration
codeElements, String subjectSelector)
```

This function should be used together with the function getCodeElements(). It utilizes method `com.togethersoft.openapi.rwi.RwiModel.findMember()` and seeks the model for an RWI-element that matches the following conditions:

- it is associated with the passed `codeElements`
- it is an RwiMember
- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: found `RwiElement`, or `null`, if the requested element doesn't exist in the model

Since: Together 5

---

findDocumentedNode

```
RwiElement findDocumentedNode(Enumeration codeElements)
RwiElement findDocumentedNode(Enumeration codeElements,
String subjectSelector)
```

This function should be used together with function getCodeElements(). It utilizes method `com.togethersoft.openapi.rwi.RwiModel.findNode()` and seeks the model for an RWI-element that matches the following conditions:

- it is associated with the passed `codeElements`
- it is an RwiMember
- it will definitely be presented among all generated documents by its Main

Documentation or, if `subjectSelector` specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: found `RwiElement`, or `null`, if the requested element doesn't exist in the model

Since: Together 5

---

findDocumentedLink

```
RwiElement findDocumentedLink(Enumeration codeElements)
RwiElement findDocumentedLink(Enumeration codeElements,
String subjectSelector)
```

This function should be used together with function getCodeElements(). It utilizes method `com.togethersoft.openapi.rwi.RwiModel.findLink()` and seeks the model for an RWI-element that matches the following conditions:

- it is associated with the passed `codeElements`
- it is an RwiMember
- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: found `RwiElement`, or `null`, if the requested element doesn't exist in the model

Since: Together 5

---

findDocumentedPackage

```
RwiElement findDocumentedPackage(Enumeration
codeElements)
RwiElement findDocumentedPackage(Enumeration
codeElements, String subjectSelector)
```

This function should be used together with function getCodeElements(). It utilizes method `com.togethersoft.openapi.rwi.RwiModel.findPackage()` and seeks the model for an RWI-element that matches the following conditions:

- it is associated with the passed `codeElements`
- it is an RwiMember
- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: found `RwiElement`, or `null`, if the requested element doesn't exist in the model

Since: Together 5

---

findDocBySubjectSelector

```
String findDocBySubjectSelector(String
subjectSelectorList)
```

Returns the first generated document that contains an area marked with one of the specified subject selectors from the list.

This is how it works: 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 the first. Otherwise, it iterates to the next subject selector from the list and repeats examination. When all subject selectors are passed and no document found, the function returns an empty string.

Parameter: List of subject selectors separated with semicolons.

Note: blank subject selector is allowed and will refer to the Main Documentation of an element.

Returns: Path of the found document relatively to the documentation's root directory. Subdirectories are delimited with slash "/". If no document found, returns an empty string.

Example:

```
findDocBySubjectSelector("package-summary;summary")
```

returns the fist generated document for one of the subject selectors: "package-summary", "summary"

Warning:

This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition.

Since: Together 5

---

findDocByTemplate

```
String findDocByTemplate(String templateList)
```

Returns the first generated document produced by one of the specified templates.

This is how it works: 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 the first. Otherwise, it iterates to the next template from the passed list and repeats examination. When all templates are passed and no document found the function returns an empty string.

Parameter: List of template names (without file name extensions) separated with semicolons.

Returns: Path of the found document relatively to the documentation's root directory. Subdirectories are delimited with slash "/". If no document found, returns an empty string.

Example:

```
findDocByTemplate("all-classes;all-diagrams")
```

returns the fist document produced by one of the templates: "all-classes.tpl" and "all-diagrams.tpl"

Warning:
This function can be used only inside Source File Name Expression of the node in FrameSet Structure definition.
Since: Together 5

---

checkStockSectionOutput

```
boolean checkStockSectionOutput(String
stockSectionName, RwiElement rwiElement)
```

Tests if a Stock Section with the name `stockSectionName` will produce a non-empty output, provided that it is invoked from a Stock Section Call and rwiElement is passed to it as the current model element. When calling this function no actual output is produced.

Parameters:

`stockSectionName` - name of the Stock Section to be tested. If no Stock Section with the specified name is found in the template, the function call issues an error message and stop the generator.

`rwiElement` - RWI-element passed to the Stock Section as the current model element.

Returns: `true`, if the tested Stock Section would have a non-empty output; `false`, otherwise

Example:

```
checkStockSectionOutput("Included Diagram List",
getDGRwiElement("curElement"))
```

Since: Together 5

---

getPropertyExt

```
String getPropertyExt(String propertyName)
```

This function gets any element property available in DG for the *metatype* to which this element belongs. It includes the properties provided by RWI and the properties calculated only by DG (names of such properties start with %. See file MetaModel.mm).

Possible call is: `rwiElement->getPropertyExt(propertyName).` In this case, the RWI-element whose property should be obtained, is specified before arrow.

Parameter: name of the required property

Returns: value of the property or **empty** string if the element has no such property

See also: getProperty()

Utility functions provided by DG

substring

```
String substring(String str, int beginIndex)
 String substring(String str, int beginIndex, int
endIndex)
```

Returns a new string that is a substring of the string `str`. Parameters are the same as in the standard Java `String.substring()` methods.

---

replace

```
String replace(String str, String oldStr, String
newStr)
```

Returns a new string produced by replacing all occurrences of `oldStr` in the string `str` with `newStr`. Operation is case sensitive.

Example:

```
replace("str-oldStr-newStr", "Str", "S")
returns: "str-oldS-newS"
```

This function is especially helpfull when you create a Call to Template section and location of the document, generated by the called template, should be derived from some properties of the current model element (for example, from the full nume of the package where the current element belongs). In such a case you can write in the field "Output Directory Expression" something like this:

```
replace(getContainingPackage()-
>getProperty("$fullName"), ".", "/")
```

See also: Linking document templates when designing Multi-Frame documentation.

Since: Together 5

---

duplicate

```
String duplicate(String str, int num)
```

Returns a new string resulting from duplication of the specified string `str num` times. If `num` is 0, returns an empty string.

Example:

```
duplicate("abc", 3)
```

returns: `"abcabcabc"`

Since: Together 5

---

length

```
int length(String str)
```

Returns the length of string `str`.

---

str

```
String str(Numeric N)
```

Converts numeric value to a string.

---

val

```
Numeric val(String str)
```

Converts numeric value represented as String into Numeric format. If conversion is impossible, returns 0.

The following functions, commonly provided in Together formulae queries, are also very useful in DG expressions.

getProperty

```
String getProperty(String rwiPropertyName)
```

Returns the value of the specified RWI property the current element has.

Possible call is `rwiElement->getProperty(rwiPropertyName).` In this case, the RWI-element, whose property should be obtained, is specified before arrow.

Parameter: name of the required property

Returns: value of the property or **empty** string if the element has no such property

See also: getPropertyExt()

---

hasProperty

```
boolean hasProperty(String rwiPropertyName)
```

Checks if the current element has the specified property.

Possible call is: `rwiElement->hasProperty(rwiPropertyName)`

In this case, the RWI-element, whose property should be checked, is specified before arrow.

Parameter: name of the property being checked

Returns: **`true`** , if the element has such property; **`false`**, otherwise

---

hasPropertyValue

```
boolean hasPropertyValue (String rwiPropertyName,
String value)
```

Checks if the current element has the property with the specified value.

Possible call is: `rwiElement->hasPropertyValue (rwiPropertyName, value)`

In this case, the RWI-element, whose property should be checked, is specified before arrow.

Parameter:

rwiPropertyName name of the property being checked

value required property value

Returns: **`true`** , if the element has specified property with the required value; **`false`**, otherwise

---

if

```
type if(boolean condition, type value1, type value2)
```

If the parameter `condition` is **`true`**, the function returns `value1.` If the `condition` is **`false`**, the function returns `value2`.

The ***`type`*** can be any data type allowed in queries.

---

getContainingNode

```
RwiNode getContainingNode()
```

returns the RwiNode element that contains the current element. Can be called for RWI *member* or *node* current element.

Possible call is: `rwiElement->getContainingNode()`

Example:

the following expression calculates visibility modifier for the class/interface member:

```
if (hasProperty("$private"), "private",
   if (hasProperty("$protected"), "protected",
     if (hasProperty("$public") &&
!getContainingNode()-> hasProperty("$interface"),
        "public", "")))
```

In this case, the `public` modifier is printed only when the containing node is not an interface, since all interface members are `public` implicitly.

Launching DocGen from the command line

You can launch the documentation generation system from the command line using the following syntax. Note that this does not presently run fully in console mode... the Generate Documentation dialog will appear to set required options.

Usage:

```
TgStarter -script:com.togethersoft.modules.gendoc.GenerateDocumentation
PrjName
```

Where:

Win 32	`TgStarter = %TgHome%\bin\Together.bat` **or** `%TgHome%\bin\TogetherCon.exe` **or** `%TgHome%\bin\Together.exe -con`
Other OS	`TgStarter = %TgHome%\bin\Together.sh`
PrjName = fully qualified project name, e.g. `%TgHome%\samples\java\CashSales\CashSales.tpr`	

Automated Doc 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 docgen facility through the command line interface. Together provides the possibility to generate documentation without actually opening Together window. You can use one of the launchers shipped with Together: `$TGH$\bin\Together.exe, $TGH$\bin\TogetherCon.exe, or $TGH$\bin\win32\umldoc.exe` (for Windows users only).

`Together.exe` and `TogetherCon.exe` allow you to choose between RTF and HTML documentation, while `umldoc.exe` produces HTML documentation only. On the other hand, `TogetherCon.exe` and `umldoc.exe` bypass Together window, while `Together.exe` starts the shell and brings you through the entire doc generation procedure.

The syntax rules are outlined in Command Line Parameters.

Examples

`Together.exe` for *genHTML*:

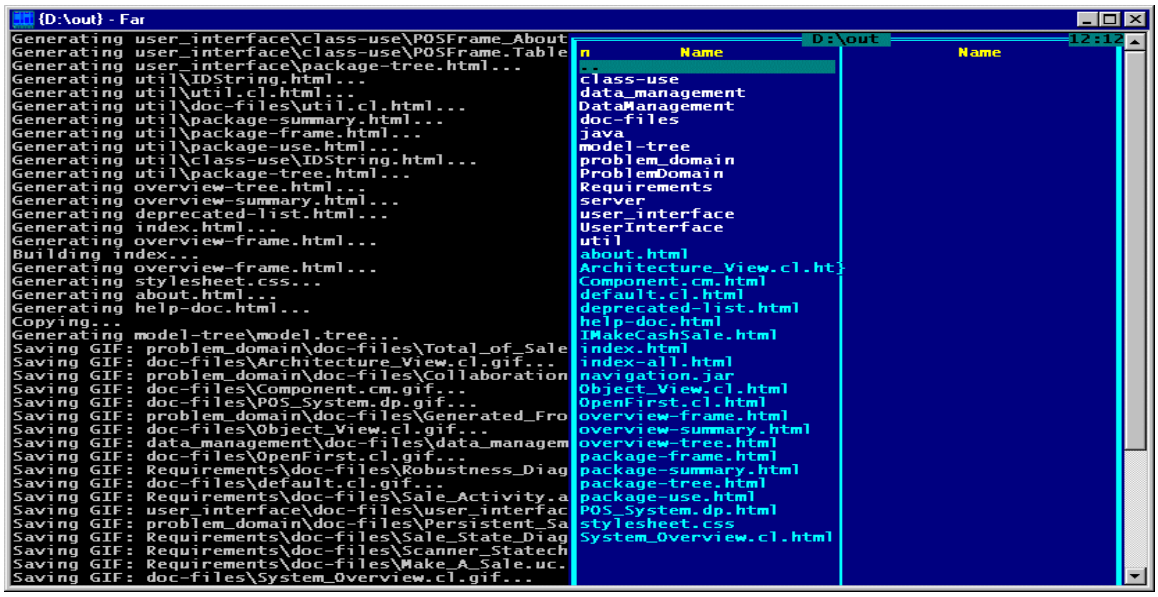
```
Together.exe -
script=com.togethersoft.modules.genhtml.GenerateHTM
L - sourcepath d:\Together\Samples\java\CashSales
d:\Together\Samlpes\java\CashSales.tpr
```

`Together.exe` for *gendoc*

```
Together.exe -
script=com.togethersoft.modules.gendoc.GenerateDocu
mentation
d:\Together\myprojects\CashSales\CashSales.tpr
```

`TogetherCon.exe` for *genHTML:*

```
TogetherCon.exe -
script=com.togethersoft.modules.genhtml.GenerateHTM
L -browser -d d:\out
d:\Together5\myprojects\CashSales\CashSales.tpr
```



`TogetherCon.exe` for *gendoc*

```
TogetherCon.exe -
script=com.togethersoft.modules.gendoc.GenerateDocu
mentation -d d:\out
d:\Together\myprojects\CashSales\CashSales.tpr
```

`umldoc.exe`:

```
umldoc.exe -d d:\out
d:\Together\myprojects\CashSales\CashSales.tpr
```

Quality Assurance

Metrics and Audits

Together provides Quality Assurance features to unobtrusively help you enforce company standards and conventions, capture real metrics, and improve what you do.

QA module is activated by default, and Quality Assurance command node is available on the Tools menu.

When you invoke QA commands from the Tools menu, the entire project is processed. On the contrary, QA invoked from the model tree or diagram speedmenu, deals with the selected classes, packages or diagrams.

You can create, save, and reuse custom sets of Metrics and Audits. 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 Metric/Audit Sets later in this topic.

Availability of Quality Assurance features depends on the project language. Java projects support full scale of metrics and audits, while for the other languages only limited sets are available.

You can find full description of each metric or audit rule in the appropriate sections of Metrics Reference and Audit Reference under the Table of Contents.

It is worth mentioning that both Audit and Metrics are only valid for the compilable source code. If your source code contains errors, or some libraries and paths are not included, QA can produce bizarre results.

How to perform metrics analysis

QA Metrics module evaluates object model complexity to support quality assurance. From the resulting table, it is possible to navigate directly to diagram or source-code.

This how it's done:

1. Open a project.
2. Make sure that Quality Assurance module is activated (Options | Activatable Modules).
3. Choose Quality Assurance | Metrics on the Tools menu (also present on diagram element and Explorer speedmenus).
4. Select the desired metrics from the list as described in the Metrics dialog and click *Start*.
5. In the QA output, select any element and choose *Open* from its speedmenu.

How to perform audit analysis

QA Audit module automatically checks for conformance to some standard or user-defined style, maintenance and robustness guidelines. From the resulting tables, one can navigate directly to diagram or source-code.

This how it's done:

1. Open a project.
2. Make sure that Quality Assurance module is activated.
3. Choose Quality Assurance | Audit on the Tools menu, diagram element or Explorer speedmenu.
4. Select the desired audits from the list as described in the Audit dialog and click *Start*.
5. In the QA output, select any element and choose *Open* from its speedmenu.

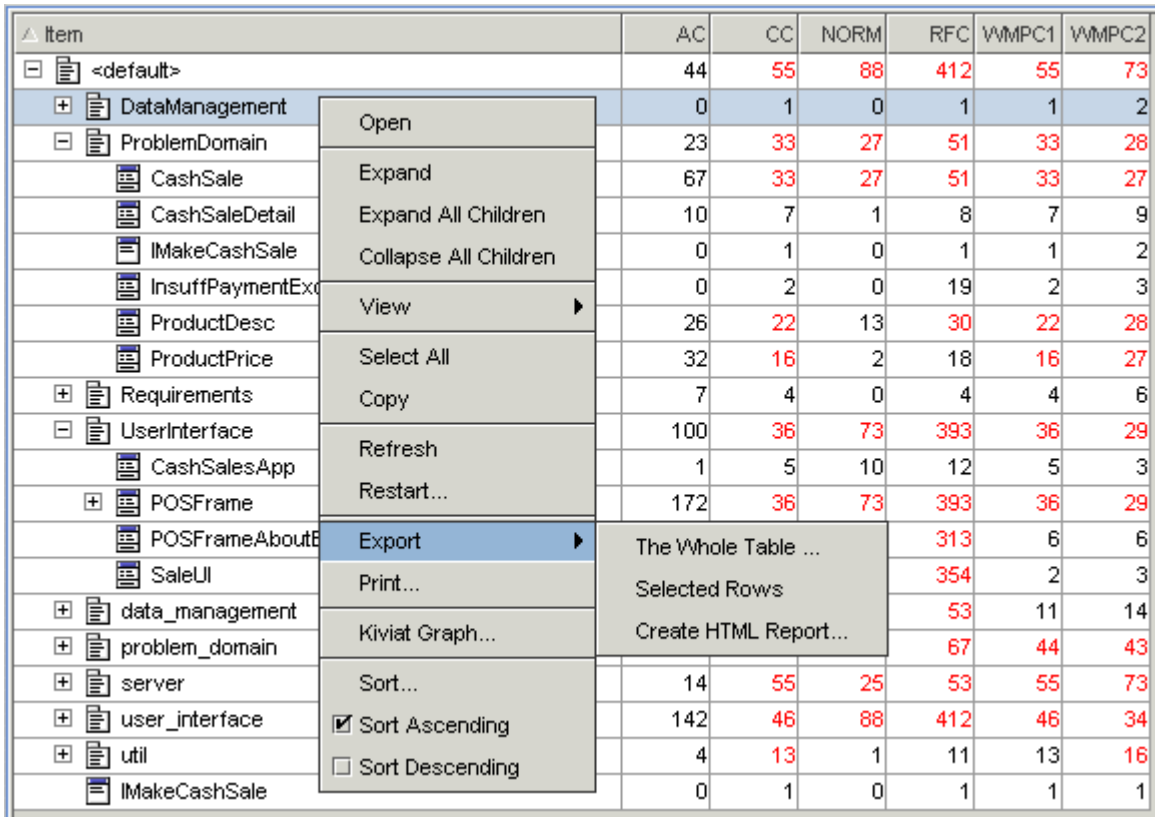
QA output

Audit and Metrics features display resulting reports of analysis in the appropriate tabs of the Message pane. Metrics resulting report displays a table of classes, packages or diagrams, included in the analysis, with the values of selected metrics. Audit resulting report represents a list of selected audits for the diagram elements, included in the analysis. Refer to the description of resulting reports' speedmenus for both Audit and Metrics modules.

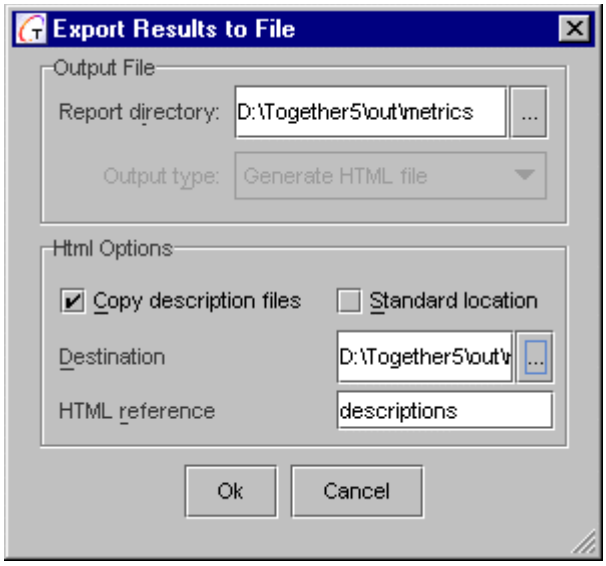
The results of both Audit and Metrics analysis are tightly connected with the source code. From any line of the resulting table, you can navigate to the appropriate location both in the Diagram pane, and in the Editor. To do this, select the desired Audit or Metric table row, and double click on it, or choose Open on the speedmenu. Same functionality is available on the graphic output.

Exporting analysis results

Speedmenus of both result reports contain Export command that allows to copy the description files of metrics and audits to the desired location.



In the *Export Results to File* dialog you can specify target file location and select export format from the Output Type dropdown list:



Possible output formats are:

- HTML format
- Text format separated with tabs
- Text format aligned with spaces

Copy Description Files

When HTML format is selected, *Copy Description Files* option is enabled that allows including descriptions of the audits and metrics to the exported result report.

In the *Generate HTML File* mode, the headers of the result table are interpreted as hyperlinks to the descriptions of the relevant audits and metrics. By default, these hyperlinks refer to the description files under `%TGH%\modules\qa`.

Selecting *Copy Description Files* option sets target location for the description files, and the hyperlinks in the tables headers change accordingly. This feature is very handy when you need to put quality assurance results to the network.

Linked HTML Report for Metrics

Metrics report in HTML format is generated as a set of linked HTML files.

The set of files representing the result 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 with each other. It makes possible to navigate through the project tree model.

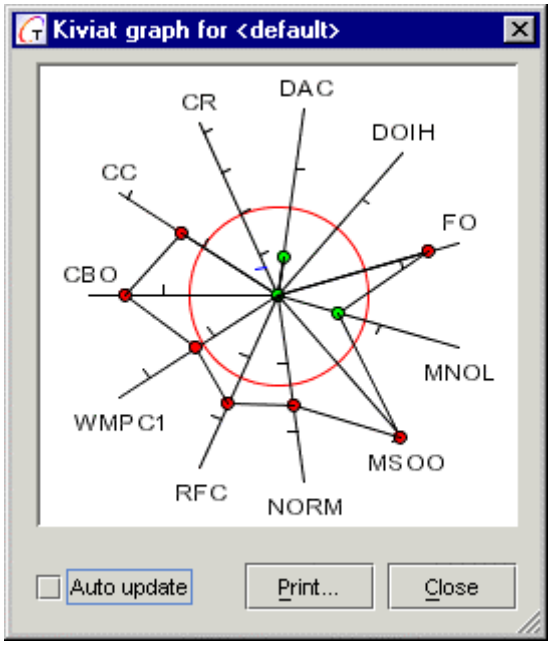
Graphic output of Metrics result report

Metrics module provides the possibility of graphic output. Speedmenu of the result report includes two commands, Kiviat graph and Bar graph.

Kiviat graph

This graph demonstrates the analysis results of a certain class or package for those metrics that have pre-defined limiting values. The metrics results are laid off along the axes that originate from the center of the graph. Each axis has special scale, selected in such a way that

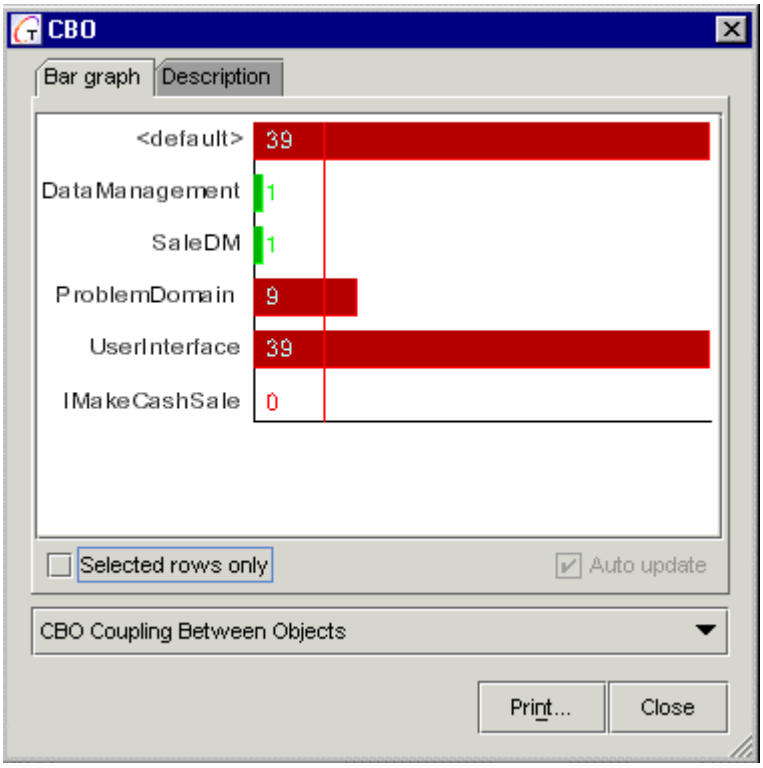
all limiting values are equidistant from the center. Thus, the limiting values represent a red circle, and the actual metrics show up as a star. The dot that falls out of the circle means that relevant metric is exceeded.



The following notation is used in the Kiviat graph:
- Green dots represent the normal values
- Blue dots represent the values below the lower limit
- Red dots represent the values over the upper limit.
- Scale marks are displayed as clockwise directional ticks perpendicular to the Kiviat ray. Low limit labels are displayed as anti-clockwise directional blue ticks perpendicular to the Kiviat ray.

Bar graph

This graph displays selected metric for all classes and/or packages. Color of the bars reflects conformance to the limiting values of the metric in reference. If metric for a certain class falls within the permissible range, the appropriate bar is green, exceeding metric is red. Blue bars represent the value that are lower than the minimal permissible value.



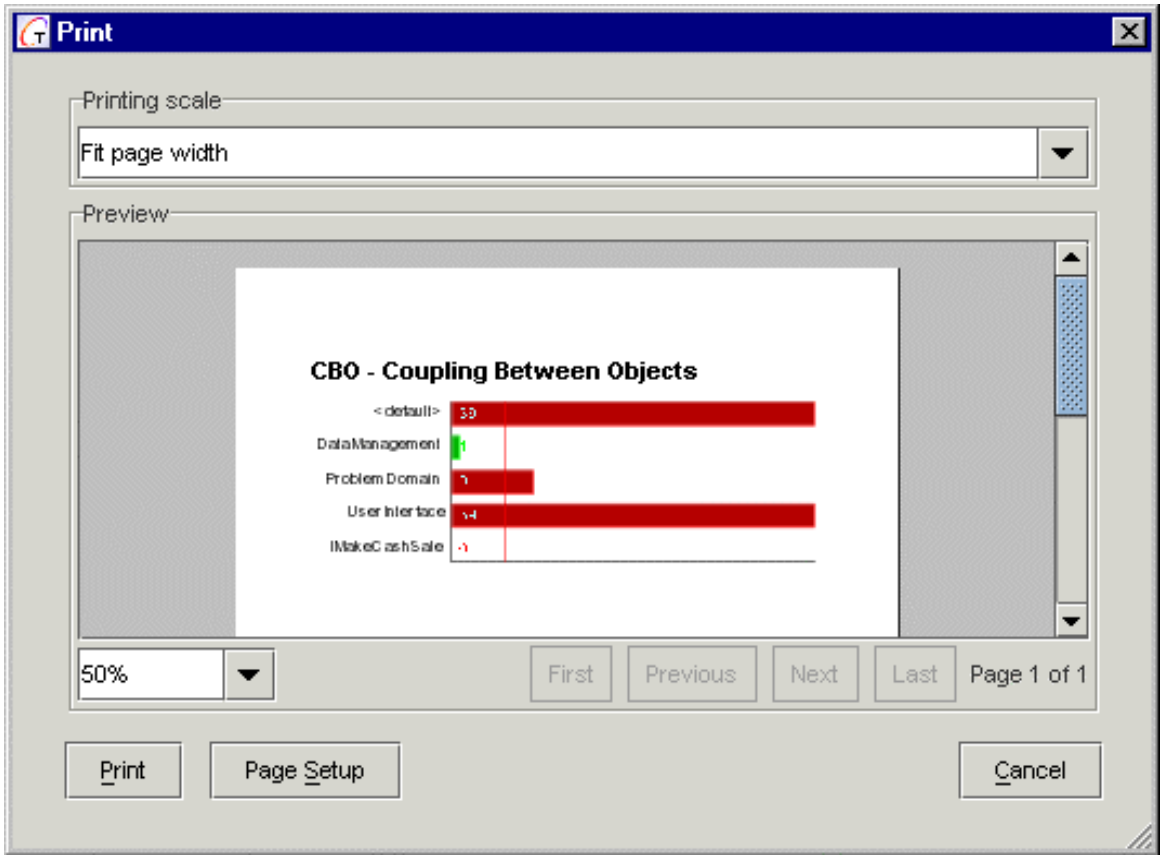
You can use dropdown list to change metric in the same dialog window. Checkbox *Selected rows only* allows to display metrics result for the highlighted rows of the result table.

Set *Auto update* checkbox to enable re-drawing of the graph while you browse through the result table.

Double click on a bar, or right click and Open command on the bar speedmenu navigates to the appropriate package or class of the source code, both in the Diagram pane, and in the Editor pane.

Print graphic results

Both Kiviat graph and Bar graph enable printout. Pressing the Print button invokes Print dialog, which is quite similar to the other print dialogs of Together. The dialog window displays preview and provides the choice of preview and printing scales.



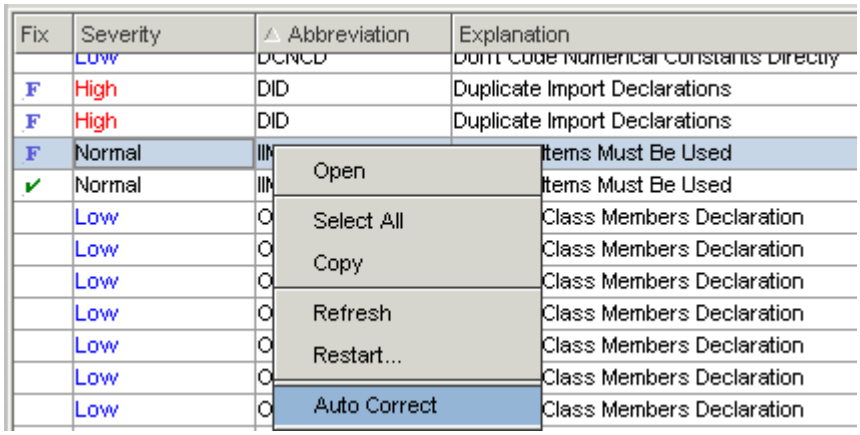
Automatic Correction

Some of the Audit rules allow automatic correction. Presently these rules are:

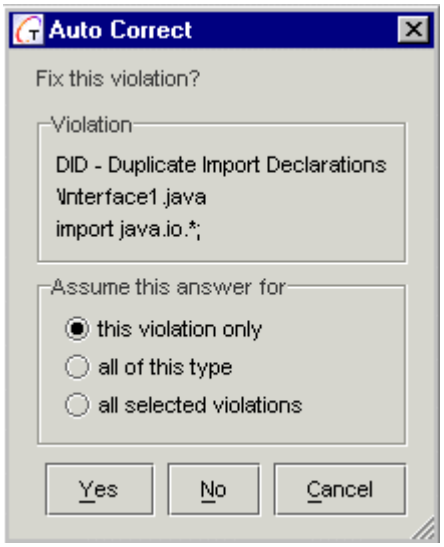
Audit Group	Abbreviation	Description
Coding Style	RFDI	Replacement For Demand Imports
Documentation	TC	Transparent Collections
Declaration Style	CPAMBF	Constant Private Attributes Must Be Final
Possible Errors	MFDCSF	Method finalize() Doesn't Call super.finalize()
Superfluous Content	DID	Duplicate Import Declarations
	DIPSFBT	Don't Import the Package the Source File Belongs To
	EIOJLC	Explicit Import Of the java.lang Classes
	IIMBU	Imported Items Must Be Used
	UOOIM	Use Of Obsolete Interface Modifier
	UOUIMM	Use Of Unnecessary Interface Member Modifiers

This is how it's done... Fixing is always enabled for these rules. In course of audit analysis, information required for correction is collected. Further, you can request correction from the result report.

The leftmost column Fix in Audit result report displays correction status for the audits that allow automatic



Select the desired rows and choose Auto Correct command on the speedmenu. Confirmation dialog shows up, where you can select the scope of automatic correction.



Corrected source code displays in the Editor pane, and fixing status appropriately changes in the Fix column.

Note

On the large-scale, and even medium-size projects, Audit is a rather time-consuming procedure. To make it work faster, the Autofix option is turned OFF by default.

Creating and Using Saved Metric/Audit Sets

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. If you open one of the dialogs and click *Start*, all of the active metrics/audits are processed. Active metrics/audits are indicated by a checkmark in the *Chosen* field of the respective dialog.

You will probably find that you don't want to run every metric/audit in the default active set every time, but rather some specific subset of available metrics/audits. Together enables 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.

You can 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 earlier in this topic.
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,* navigate to and select the desired saved set file (.adt for *Audits*, .mts for Metrics).
3. Go through the individual metrics/audits and select those you want to include in the set by checking the *Chosen* column next to each item. (You can 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. (The default location is recommended just to keep things simple.)

> **Tip:** *You might want to include the .adt and .mts files in your backup routine.*

To open a saved set of active metrics/audits:

1. Open the relevant dialog (Metrics or Audits) as described earlier in this topic.
2. Click *Load set* and navigate to the saved set file you want to use.

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

> **Note:** QA plugins reside in `$TGH$/modules/com/togethersoft/modules/qa`. You can find detailed API for the QA plugins in `$TGH$/modules/com/togethersoft/modules/qa/doc`.

See also

QA Audit/Metrics Command Mode

Refactoring

Refactoring is a technique used to improve the design of existing source code so that it is easier to understand and maintain, and easier to modify.

Refactoring a system should not alter its external behavior, but rather be an "invisible" improvement to the internal workings of the system. In general, it can even be thought of as a way to "clean up" the code - however, following a set of rules that keeps the code changes under control.

A software system may start with a good model and design. Over time, many people, who are often not the original authors, modify the design, which can (and usually does) result in degraded design and code quality. If you start to sense that code changes in one spot of your application begin to affect seemingly unrelated code in another spot, you have discovered a system that is no longer a good design. Congratulations! This is a good chance to try out refactoring.

Presently, Together provides a tool to support refactoring at the source code editor level and on the diagram level. In particular, it is possible to extract new operation from possibly repeated chunks of code in existing methods, and to rename a class, an attribute or an operation. So doing, all links and references are also updated.

Extract Operation

This is how it's done... In the Editor pane highlight the fragment of code, which you consider to be worth extracting to a separate method, and invoke the editor speedmenu. Choose *Refactoring/ Extract Operation* command. This brings in Extract Operation dialog, where you can specify new method name, comments etc.

Note that Extract Operation only applies to a semantically completed piece of code that complies with certain conditions. When doing selection, take into consideration the following issues.

Together analyses parameters and local variables involved in the selection. If referred, they become the parameters of the newly created method. If some variables are immediately initialized in the selection, then there is no need to pass them as parameters - on the contrary, they are declared as local variables in the new method.

If a certain local variable declared before the selection, is modified therein, then the newly created method uses it as a return value. When the selection is replaced with the created method, return value of the method is assigned to this variable, except for the situations when this variable is not referred to anymore after the selection.

If there is more than one such variable, it is impossible to extract a method, and an error message comes out.

Together also analyses exceptions thrown in the selection. The list of encountered exceptions is used in the new method signature.

If a chunk of code is repeated in several locations, it is the user's responsibility to replace these fragments in the other places with appropriate method calls.

Example:

```
int someOperation(int a, int b ) {
int i = 1;

i = a + b;
synchronized(this) {
if( attr < 0 ) {
attr = i;
}
}

return i;
 }
```

will be converted into

```
int someOperation(int a, int b ) {
int i = 1;
i = newMethod(a, b);
return i;
}

private int newMethod(int a, int b){
int i;
i = a + b;
synchronized(this) {
if (attr < 0) {
attr = i;
}
}
return i;
}
```

The variable `i` is immediately assigned in the selection. That's why it is declared as a local variable in the new method, rather than passed as a parameter.

Renaming

Select class icon, attribute or operation on the Diagram pane and right-click to display the speedmenu. *Refactoring* node contains appropriate command *Rename class / attribute / operation*, that invokes Rename dialog. The dialog window provides text area to enter the new name and displays treeview of usages to be renamed. When each specific node is selected, the relevant piece of code displays in the right frame.

Reference

The primary reference is Martin Fowler's book:

Fowler, Martin. Refactoring - Improving the Design of Existing Code. Reading, MA: Addison-Wesley, 1999.

See also

Extract Operation dialog
Renaming dialogs

Language-Specific Metrics and Audits

Metrics and Audits for the languages other than Java are not fully supported. This can be explained by the lack of deep parsing in C#, Visual Basic and VBNet. Deep parsing in C++ is also insufficient. That's why these languages support declaration-based metrics and do not support the ones that require deep parsing. For the same reason the audits are not supported, since deep parsing is involved in nearly every audit.

The sets of metrics for C++, C# and VBNet are almost similar. VB6 lacks inheritance, and thus all metrics related to inheritance and polymorphism are not available.

Metrics and Audits Support for C++

Some of the metrics included in the Metrics analysis are available for C++ classes. These are:

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling
DOIH	Depth Of Inheritance Hierarchy
LOC	Lines Of Code
MHF	Method Hiding Factor
MIF	Method Inheritance Factor
MNOP	Maximum Number Of Parameters
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
NOCON	Number of constructors
PF	Polymorphism Factor
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members
TCR	True Comment Ratio
WMPC2	Weighted Methods Per Class 2

The Audits supported for C++ are:

ACVFCD	Avoid Calling Virtual Functions from Constructors and Destructors
AHIA	Avoid Hiding Inherited Attributes
AHISM	Avoid Hiding Inherited Static Methods
APAPA	Avoid Public and Package Attributes
AUVC	Always Use "Virtual" keyword
CDPMD	Call Delete on Pointer Members in Destructors
HON	Hiding of Names
MVDWSN	Multiple Visible Declarations With Same Name
ONAMWAM	Overriding a Non-Abstract Method with an Abstract Method
OWS	Overloading with a Subclass
UVD	Use Virtual Destructor

Metrics Adapted for VB6

Metrics adapted for VB6 are:

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
NOCC	Number Of Child Classes
NOM	Number Of Members
NOO	Number Of Operations
PPrivM	Percentage of Private Members
PPubM	Percentage of Public Members
TCR	True Comment Ratio
WMPC2	Weighted Methods Per Class 2

Metrics supported for C#

The following metrics that do not require deep parsing of the code, are supported in C# projects:

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF**	Attribute Inheritance Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling
DOIH	Depth Of Inheritance Hierarchy
LOC	Lines Of Code
MHF	Method Hiding Factor
MIF**	Method Inheritance Factor
MNOP	Maximum Number Of Parameters
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOCON	Number Of Constructors
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
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
TCR	True Comment Ratio
WMPC2	Weighted Methods Per Class 2

Notes:

*This metric is specific for C# only.

** SCI can't 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 will produce wrong result.

Metrics Adapted for VBNet

Metrics adapted for VB6 are:

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling
DOIH	Depth Of Inheritance Hierarchy
LOC	Lines Of Code
MHF	Method Hiding Factor
MIF	Method Inheritance Factor
MNOP	Maximum Number Of Parameters
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOCON	Number Of Constructors
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
PF	Polymorphism Factor
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members
TCR	True Comment Ratio
WMPC2	Weighted Methods Per Class 2

Audits Reference

Section ACEV through AUVK

ACEV - Avoid Constants with Equal Values

This rule catches constants with equal values. Presence of different constants with equal values may result in bugs in case if these constants have equal meaning.

Wrong

```
final static int SUNDAY = 0;
final static int MONDAY = 1;
final static int TUESDAY = 2;
final static int WEDNESDAY = 3;
final static int THURSDAY = 4;
final static int FRIDAY = 5;
final static int SATURDAY = 0;

 // This method would never return "Saturday"
 void getDayName(int day) {
     if( day == SUNDAY )
         return "Sunday";
     ...
     else if( day == SATURDAY )
         return "Saturday";
     ...
 }
```

ACIUCFL - Avoid Complex Initialization or Update Clause in For Loops

When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables.

Wrong

```
for ( i = 0, j=0, k=10, l=-1 ; i < cnt;
    i++, j++, k--, l += 2 ) {
     // do something
 }
```

Tip: If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause).

Right

```
l = -1;
for ( i = 0, j=0, k=10; i < cnt;
 i++, j++, k-- ) {
     // do something
     l += 2;
 }
```

ACVFCD - Avoid Calling Virtual Functions from Constructors and Destructors

C++ only

In a constructor, the virtual call mechanism is disabled because overriding from derived classes hasn't yet happened. Objects are constructed from the base up, "base before derived".

Consider:

```
    #include< string>
    #include< iostream>
    using namespace std;
    class B {
    public:
    B(const string& ss) { cout << "B constructor\n"; f(ss);
    }
    virtual void f(const string&) { cout << "B::f\n";}
    };

    class D : public B {
    public:
    D(const string & ss) :B(ss) { cout << "D
    constructor\n";}
    void f(const string& ss) { cout << "D::f\n"; s = ss; }
    private:
    string s;
    };

int main()

{

D d("Hello");

}
```

the program compiles and produces the following result:

```
B constructor
B::f
D constructor
```

Note that it is *not* D::f. Consider what would happen if the rule were different so that D::f() was called from B::B(): Because the constructor D::D() hasn't run yet, D::f() would try to assign its argument to an uninitialized string s. The result would most likely be immediate crash.

Destruction is done "derived class before base class", so virtual functions behave as in constructors: Only the local definitions are used - and no calls are made to overriding functions to avoid touching the (now destroyed) derived class part of the object.

Source: Bjarne Stroustrup's C++ Style and Technique FAQ

ADVIL - Avoid Declaring Variables Inside Loops

This rule recommends declaring local variables outside the loops. The reason: as a rule, declaring variables inside the loop is less efficient.

Wrong

```
int good_var =0;
for (int i = 0; i < 100; i++) {
     int var1 = 0;
     // ...
 }
 while (true) {
     int var2 = 0;
     // ...
 }
 do {
     int var3 = 0;
     // ...
 } while (true);
```

Tip: Move variable declarations out of loop

Right

```
int good_var =0;
int var1;
 for (int i = 0; i < 100; i++) {
    var1 = 0;
    // ...
 }
 int var2;
 while (true) {
    var2 = 0;
    // ...
 }
 int var3;
 do {
    var3 = 0;
    // ...
 } while (true);
```

AECB - Avoid Empty Catch Blocks

Catch blocks should not be empty. Programmers frequently forget to process negative outcomes of a program and tend to focus more on the positive outcomes.

When 'Check parameter usage' option is on, this rule also checks, whether code does something with the exception parameter or not. If not, violation is raised.

You can also specify the list of exceptions, which should be ignored. For example, for `PropertyVetoException` catch block usually is empty - as a rule, the progam just does nothing if this exception occurs.

AHIA - Avoid Hiding Inherited Attributes

Detects when attributes declared in child classes hide inherited attributes.

Wrong

```
class Elephant extends Animal {
    int attr1;
     // something...;
 }
 class Animal {
     int attr1;
 }
```

Tip: Rename child class attribute.

Right

```
class Elephant extends Animal {
    int elphAttr1;
     // something...;
 }
 class Animal {
     int attr1;
 }
```

AHISM - Avoid Hiding Inherited Static Methods

Detects when inherited static operations are hidden by child classes.

Wrong

```
class Elephant extends Animal {
    void oper1() {}
     static void oper2() {}
 }
 class Animal {
     static void oper1() {}
     static void oper2() {}
 }
```

Tip: Rename either ancestor or descendant class operations

Right

```
class Elephant extends Animal {
    void anOper1 () {}
     static void anOper2 () {}
 }
 class Animal {
     static void oper1() {}
     static void oper2() {}
 }
```

AIPR - Avoid Implementation Packages Referencing

This rule helps you to avoid referencing packages, which normally shouldn't be referenced. For example, if you use Facade or AbstractFactory patterns, you might verify that nobody uses direct calls to underlying classes' constructors.

In this case you might divide your packages into interface and implementation ones and ensure that nobody ever references implementation packages - only interface ones.

You can set two lists - allowed (interface) and banned (implementation) packages. For each class reference in source code, this rule verifies, whether the package this class belongs to is in allowed and is not in banned list.

Package names in list may be

'*'	- any package is allowed (banned)
package name	- this package is allowed (banned)
package name postfixed by '*'	- any subpackage of the given package is allowed (banned)

In the case of conflict, narrower rule is stronger. For example, if you specify *allowed* list as

```
*
com.mycompany.openapi.*
```

and *banned* list as

```
com.mycompany.*
```

than all subpackages of `com.mycompany` package would be banned except for those belonging to `com.mycompany.openapi` subpackage.

ANFSA - Avoid Non Final Static Attribute

This rule helps you avoid non final static attributes.

Wrong

```
protected static ArrayList InterfaceListeners = new
ArrayList();
```

Right

```
protected final static ArrayList InterfaceListeners = new
ArrayList();
```

AOSMTO - Access Of Static Members Through Objects

Static members should be referenced through class names rather than through objects.

Wrong

```
class AOSMTO1 {
     void func () {
          AOSMTO1 obj1 = new AOSMTO1();
          AOSMTO2 obj2 = new AOSMTO2();
          obj1.attr = 10;
          obj2.attr = 20;
          obj1.oper();
          obj2.oper();
          this.attr++;
          this.oper();
     }
     static int attr;
     static void oper () {}
 }
 class AOSMTO2 {
     static int attr;
     static void oper () {}
 }
```

Tip: Always reference static members via class names

Right

```
class AOSMTO1 {
     void func () {
          AOSMTO1 obj1 = new AOSMTO1();
          AOSMTO2 obj2 = new AOSMTO2();
          AOSMTO1.attr = 10;
          AOSMTO2.attr = 20;
          AOSMTO1.oper();
          AOSMTO2.oper();
          AOSMTO1.attr++;
          AOSMTO1.oper();
     }
     static int attr;
     static void oper () {}
 }
 class AOSMTO2 {
     static int attr;
     static void oper () {}
 }
```

APAPA - Avoid Public And Package Attributes

Declare the attributes either private or protected and provide operations to access or change them.

There might be one exception when some class is used just as `struct` in C language: it just holds some values, and thus has no methods. Formally option regulates whether to raise violations for such classes. When formally option is off, violations are not raised.

Wrong

```
class APAPA {
     int attr1;
     public int attr2;
 }
```

Tip: Change visibility of attributes to either private or protected.
Provide access operations for these attributes.

Right

```
class APAPA {
    private int attr1;
     protected int attr2;
     public int getAttr1() {
         return attr1;
     }
     public int getAttr2() {
         return attr2;
     }
     public void setAttr2(int newVal) {
         attr2 = newVal;
     }
 }
```

ASL - Avoid String Literals

If your project can be internationalized you must avoid hard coded string and character literals and keep them in external resource. The rule helps to find such unsafe literals.

You can customize the filter by entering the list of appropriate literals separately for strings and for characters and by checking the "ignore white spaces" and "ignore digits" options.

ASWEB - Avoid Statements With Empty Body

As far as possible avoid statements with empty body.

Wrong

```
StringTokenizer st = new StringTokenizer(class1.getName(),
".", true);
String s;
for( s = ""; st.countTokens() 2;
     s = s + st.nextToken() );
```

Tip: Provide statement body or change the logic of the program (for example, use `while` statement instead of `for` statement)

Right

```
StringTokenizer st = new StringTokenizer(class1.getName(),
".", true);
String s = "";
while( st.countTokens() 2 ) {
     s += st.nextToken();
 }
```

ATFLV - Assignment To For-Loop Variables

For-loop variables should not be assigned to.

Wrong

```
for (int i = 0; i < charBuf.length; i++) {
     if ( Character.isWhitespace(charBuf[i]) )
         i++;
     ....
 }
```

Tip: Use `continue` operator or convert for-loop to while-loop.

Right

```
for (int i = 0; i < charBuf.length; i++) {
     if ( Character.isWhitespace(charBuf[i]) )
         continue;
     ....
 }
```

ATFP - Assignment To Formal Parameters

Formal parameters should not be assigned to.

Wrong

```
int oper (int param) {
     param += 10;
     return ++param;
 }
```

Tip: Declare internal variable and use it instead of formal parameter

Right

```
int oper (int param) {
     int result = param + 10;
     return ++result;
 }
```

ATLF - Avoid Too Long Files

According to Sun Code Conventions for Java, files longer than 2000 lines are cumbersome and should be avoided.

ATLL - Avoid Too Long Lines

According to Sun Code Conventions for Java, lines longer than 80 characters should be avoided, since they're not handled well by many terminals and tools.

ATSWL - Append To String Within a Loop

Performance enhancements can be obtained by replacing String operations with StringBuffer operations if a String object is appended to within a loop.

Wrong

```
public class ATSWL {
     public String func () {
          String var = "var";
          for (int i = 0; i < 10; i++) {
              var += (" " + i);
          }
          return var;
     }
 }
```

Tip: Use StringBuffer class instead of String

Right

```
public class ATSWL {
    public String func () {
          StringBuffer var = new StringBuffer("var");
          for (int i = 0; i < 10; i++) {
              var.append(" " + i);
          }
          return var.toString();
     }
 }
```

AUVK - Always Use 'Virtual' Keyword

C++ only

It is advisable to mark all virtual functions with **virtual** keyword. This makes code more readable, since one doesn't need to go through the entire class hierarchy to understand that a certain function is virtual.

Wrong

```
class Shape {
 public:
 virtual void draw();
 // ...
 };

 class Circle : public Shape {
 public:
 void draw(); // Circle::draw() is virtual but not marked as
virtual
 // ...
 };
```

Right

```
class Shape {
 public:
 virtual void draw();
 // ...
 };

 class Circle : public Shape {
 public:
 virtual void draw();
 // ...
 };
```

Section BLAD through DVIOSE

BLAD - Badly Located Array Declarators

Array declarators must be placed next to the type descriptor of their component type.

Wrong

```
class BLAD {
     int attr[];
     int oper (int param[]) [] {
         int var[][];
         // do something
     }
 }
```

Right

```
class BLAD {
     int[] attr;
     int[] oper (int[] param) {
         int[][] var;
         // do something
     }
 }
```

BTIJDC - Bad Tag In JavaDoc Comments

This rule verifies code against accidental use of improper JavaDoc tags.

Wrong

```
package audit;
 /** Class BTIJDC
   * @BAD_TAG_1
   * @version 1.0 08-Jan-2000
   * @author TogetherSoft
   */
 public class BTIJDC {
     /**
        * Attribute attr
        * @BAD_TAG_2
        * @supplierCardinality 0..*
        * @clientCardinality 1
        */
     private int attr;
     /** Operation oper
       * @BAD_TAG_3
       * @return int
        */
     public int oper () {}
 }
```

Tip: Replace misspelled tags. Or if your JavaDoc tool (doclet, etc.) uses some non-standard tags, add them to the list of valid tags.

CA - Complex Assignment

Checks for the occurrence of multiple assignments and assignments to variables within the same expression. Too complex assignments should be avoided since they decrease program readability.

If 'strict' option is *off*, assignments of equal value to several variables in one operation is permitted. For example the following statement would raise violation if 'strict' option is *on*, otherwise there would be no violation:

```
i = j = k = 0;
```

Wrong

```
// compound assignment
i *= j++;
k = j = 10;
l = j += 15;
// nested assignment
i = j++ + 20;
i = (j = 25) + 30;
```

Tip: Break statement into several ones.

Right

```
// instead of i *= j++;
j++;
i *= j
// instead of k = j = 10;
k = 10;
j = 10;
// instead of l = j += 15;
j += 15;
l = j;
// instead of i = j++ + 20;
j++;
i = j + 20;
// instead of i = (j = 25) + 30;
j = 25;
i = j + 30;
```

CDPMD - Call Delete on Pointer Members in Destructors

C++ only

The audit looks for classes, which have any pointer members, and checks for cleaning up the members in the destructor.

Right

```
class Image {
 ImageBuffer* buffer;
 // ...
 public:
 // ...
 ~Image() {
 delete buffer;
 }
 // ...
 };
```

CLE - Complex Loop Expressions

Avoid using complex expressions as repeat conditions within loops.

Wrong

```
void oper () {
      for (int i = 0; i < vector.size(); i++) {
          // do something
      }
      int size = vector.size();
      for (int i = 0; i < size; i++) {
          // do something
      }
 }
```

Tip: Assign the expression to a variable before the loop and use that variable instead.

Right

```
void oper () {
      int size = vector.size();
      for (int i = 0; i < size; i++) {
          // do something
      }
      int size = vector.size();
      for (int i = 0; i < size; i++) {
          // do something
      }
 }
```

CNMMIFN - Class Name Must Match Its File Name

Checks whether top level class and interface has the same name as the file in which it resides.

Wrong

```
// File Audit_CNMMIFN.java
class CNMMIFN {}
```

Tip: Rename either class or file

Right

```
// File Audit_CNMMIFN.java
class Audit_CNMMIFN {}
```

CPAMBF - Constant Private Attributes Must Be Final

Private attributes that never get their values changed must be declared final. By explicitly declaring them in such a way, a reader of the source code get some information of how the attribute is supposed to be used.

Wrong

```
class CPAMBF {
     private int attr1 = 10;
     private int attr2 = 20;
     void func () {
         attr1 = attr2;
         System.out.println(attr1);
     }
 }
```

Tip: Make all private attributes, which are never changed final.

Right

```
class CPAMBF {
     int attr1 = 10;
     private final int attr2 = 20;
     void func () {
         attr1 = attr2;
         System.out.println(attr1);
     }
 }
```

CQS - Command Query Separation

Prevents methods that return value from modifying state. The methods that you use to query the state of an object must be different from the methods you use to perform commands (change the state of the object).

Wrong

```
class CQS {
     int attr;
     int getAttr () {
         attr += 10;
         return attr;
     }
 }
```

CVMBF - Constant Variables Must Be Final

Local variables that never get their values changed must be declared final. By explicitly declaring them in such a way, a reader of the source code get some information of how the variable is supposed to be used.

Wrong

```
void func () {
     int var1 = 10;
     int var2 = 20;
     var1 = var2;
         System.out.println(attr1);
 }
```

Tip: Make all variables, which are never changed, final.

Right

```
void func () {
     int var1 = 10;
     final int var2 = 20;
     var1 = var2;
         System.out.println(attr1);
 }
```

DBJAOC - Distinguish Between JavaDoc And Ordinary Comments

Checks whether the JavaDoc comments in your program ends with '**/' and ordinary C-style ones with '*/'.

Wrong

```
package audit;
 /**
  * JavaDoc comment
  */
 public class DBJAOC {
      /*
      * C-style comment
      **/
      private int attr;
      /**
       * JavaDoc comment
       */
      public void oper () {}
 }
```

Right

```
package audit;
 /**
  * JavaDoc comment
  **/
 public class DBJAOC {
      /*
      * C-style comment
      */
      private int attr;
      /**
       * JavaDoc comment
       **/
      public void oper () {}
 }
```

DCFPT - Don't Compare Floating Point Types

Avoid testing for equality floating point numbers: floating-point numbers that should be equal are not always equal due to rounding problems.

Wrong

```
void oper (double d) {
     if ( d != 15.0 ) {
          for ( double f = 0.0; f < d; f += 1.0 ) {
               // do something
          }
     }
 }
```

Tip: Replace direct comparison with estimation of absolute value of difference

Right

```
void oper (double d) {
     if ( Math.abs(d - 15.0) < Double.MIN_VALUE * 2 ) {
          for ( double f = 0.0; d - f > DIFF; f += 1.0 ) {
               // do something
          }
     }
 }
```

DCNCD - Don't Code Numerical Constants Directly

According to Sun Code Conventions for Java, numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

Tip: Add static final attributes for numeric constants.

DID - Duplicate Import Declarations

There should be at most one import declaration that imports a particular class/package.

Wrong

```
package audit;
import java.io.*;
import java.io.*;
import java.sql.Time;
import java.sql.Time;

class DID {
}
```

Tip: Delete duplicate declarations

DIPSFBT - Don't Import the Package the Source File Belongs To

No classes or interfaces need to be imported from the package that the source code file belongs to. Everything in that package is available without explicit import statements.

Wrong

```
package audit;
import java.awt.*;
import audit.*;

public class DIPSFBT {
}
```

Tip: Delete unnecessary import statement.

DPMSSL - Don't Place Multiple Statements on the Same Line

According to Sun Code Conventions for Java, each line should contain at most one statement.

Wrong

```
if( someCondition ) someMethod();
i++; j++;
```

Tip: Place each statement on the separate line.

Right

```
if( someCondition )
     someMethod();
i++;
j++;
```

DUNOF - Don't Use the Negation Operator Frequently

The negation operator slows down the readability of the program, so it is recommended that it should not be used frequently.

Wrong

```
boolean isOk = verifySomewhat()
if ( !isOk )
     return 0;
else
     return 1;
```

Tip: Change your program logic to avoid negation.

Right

```
boolean isOk = verifySomewhat()
if ( isOk )
     return 1;
else
     return 0;
```

DVIOSE - Declare Variables In One Statement Each

Several variables (attributes and local variables) should not be declared in the same statement.

'Different types only' option can weaken this rule. When it is *on*, violation is raised only when variables are of different types, for example:

```
int foo, fooarray[]; // definitely wrong
```

Wrong

```
class DVIOSE {
     int attr1;
     int attr2, attr3;
     void oper () {
          int var1;
          int var2, var3;
     }
 }
```

Tip: declare variables in one statement each

Right

```
class DVIOSE {
     int attr1;
     int attr2;
     int attr3;
     void oper () {
          int var1;
          int var2;
          int var3;
     }
 }
```

Section EBWB through EOOBA

EBWB - Enclosing Body Within a Block

The statement of a loop must always be a block. The then and else parts of if-statements must always be blocks. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

Wrong

```
if( st == null )
     return;
 while( st.countTokens() 2 )
     s += st.nextToken();
```

Tip: Always enclose body within a block

Right

```
if( st == null ) {
     return;
}
while( st.countTokens() 2 ) {
     s += st.nextToken();
}
```

EIAV - Explicitly Initialize All Variables

Explicitly initialize all variables. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.

Wrong

```
void func () {
     int var0;
     int var1 = 1, var2;
     // do something.. }
```

Right

```
void func () {
     int var0 = 0;
     int var1 = 1, var2 = 0;
     // do something.. }
```

EIOJLC - Explicit Import Of the java.lang Classes

Explicit import of classes from the package 'java.lang' should not be performed.

Wrong

```
package audit;
import java.lang.*;
class EIOJLC {}
```

Tip: Delete unnecessary import statement.

EJB_CL - Don't create a class loader

According to Sun EJB specifications, the enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams.

These functions are reserved for the EJB Container. Allowing the enterprise bean to use these functions could compromise security and decrease the Container's ability to properly manage the runtime environment.

EJB_CONSOLE - Don't use console output

According to Sun EJB specifications, the enterprise bean must not attempt to output information to a display.

Most servers do not allow direct interaction between an application program and a keyboard/display attached to the server system.

EJB_FDESCR - Don't attempt to directly read or write a file descriptor

According to Sun EJB specifications, the enterprise bean should not attempt to directly read or write a file descriptor.

Allowing the enterprise bean to read and write file descriptors directly could compromise security.

EJB_FILES - Access files and directories is restricted

According to Sun EJB specifications, an enterprise bean must not use the `java.io` package to attempt to access files and directories in the file system.

The file system APIs are not well-suited for business components to access data. Business components should use a resource manager API, such as JDBC API, to store data.

EJB_IO - Don't use AWT, SWING and the other UI APIs

According to Sun EJB specifications, an enterprise Bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard.

Most servers do not allow direct interaction between an application program and a keyboard/display attached to the server system.

EJB_JDBC - Don't use JDBC API from Session Beans

It's a wrong solution to work with persistent data using Session Beans that should represent business processes. Using Entity Beans is better.

Entity Beans are persistent objects that can be stored in permanent storage.

EJB_NATIVE - Don't load a native library

According to Sun EJB specifications, The enterprise bean must not attempt to load a native library.

This function is reserved for the EJB Container. Allowing the enterprise bean to load native code would create a security hole.

EJB_REFL - Don't use Reflection.

According to Sun EJB specifications, the enterprise bean must not attempt to query a class to obtain information about the declared members those are not otherwise accessible to the enterprise bean because of the security rules of the Java language. The enterprise bean must not attempt to use the Reflection API to access information that the security rules of the Java programming language make unavailable.

This function is reserved for the EJB Container. Allowing the enterprise bean to load native code would create a security hole.

EJB_SEC - Don't obtain the security policy information

According to Sun EJB specifications, the enterprise bean must not attempt to obtain the security policy information for a particular code source.

Allowing the enterprise bean to access the security policy information would create a security hole.

EJB_SECOBJ - Don't use the security configuration objects

According to Sun EJB specifications, the enterprise bean must not attempt to access or modify the security configuration objects (Policy, Security, Provider, Signer, and Identity).

These functions are reserved for the EJB Container. Allowing the enterprise bean to use these functions could compromise security.

EJB_SFACT - Don't set the socket factory

According to Sun EJB specifications, the enterprise bean must not attempt to set the socket factory used by ServerSocket, Socket, or the stream handler factory used by URL.

These networking functions are reserved for the EJB Container. Allowing the enterprise bean to use these functions could compromise security and decrease the Container's ability to properly manage the runtime environment.

EJB_SOCKET - Don't listen on a socket

According to Sun EJB specifications, an enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast.

The EJB architecture allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean-- to serve the EJB clients

EJB_SUBST - Don't use the subclass and object substitution features

According to Sun EJB specifications, the enterprise bean must not attempt to use the subclass and object substitution features of the Java Serialization Protocol.

Allowing the enterprise bean to use these functions could compromise security.

EJB_THREADS - Don't manage threads

According to Sun EJB specifications, the enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread; or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups.

These functions are reserved for the EJB Container. Allowing the enterprise bean to manage threads would decrease the Container's ability to properly manage the runtime environment.

EOOBA - Equality Operations On Boolean Arguments

Avoid performing equality operations on boolean operands. You should not use 'true' and 'false' literals in conditional clauses.

Wrong

```
int oper (boolean bOk) {
     if (bOk) {
         return 1;
     }
     while ( bOk == true ) {
         // do something
     }
     return ( bOk == false ) ? 1 : 0;
 }
```

Right

```
int oper (boolean bOk) {
     if (bOk) {
         return 1;
     }
     while ( bOk ) {
         // do something
     }
     return ( ! bOk ) ? 1 : 0;
 }
```

Section GOWSNT through NOEC

GOWSNT - Group Operations With Same Name Together

Enforces standard to improve readability.

Wrong

```
package audit;
class GOWSNT {
     void operation () {}
     void function () {}
     void operation (int param) {}
}
```

Tip: Group operations that differ only by their parameter list together. Good to order from least number of parameters to most.

Right

```
package audit;
class GOWSNT {
     void operation () {}
     void operation (int param) {}
     void function () {}
}
```

HON - Hiding Of Names

Declarations of names should not hide other declarations of the same name. The option 'Formally' regulates whether hiding of names should be detected for parameter variable, if the only usage of it is to assign its value to the attribute with the same name.

Wrong

```
class HON {
      int index;
      void func () {
          int index;
          // do something
      }
      void setIndex (int index) {
          this.index = index;
      }
 }
```

(Note that the second violation would be raised only if 'Formally' option is switched on.)

Tip: Rename variable, which hides attribute or another variable.

Right

```
class HON {
      int index;
      void func () {
          int index1;
          // do something
      }
      void setIndex (int anIndex) {
          this.index = anIndex;
      }
 }
```

ICOMM - Inaccessible Constructor Or Method Matches

Overload resolution only considers constructors and methods are visible at the point of the call. If however, all the constructors and methods were considered, there may be more matches. This rule is violated in this case.

Imagine that ClassB is in a different package than ClassA. Then the allocation of ClassB violates this rule since the second constructor is not visible at the point of the allocation, but it still matches the allocation (based on signature). Also the call to oper in ClassB violates this rule since the second and the third declarations of oper is not visible at the point of the call, but it still matches the call (based on signature).

Wrong

```
public class ClassA {
     public ClassA (int param) {}
     ClassA (char param) {}
     ClassA (short param) {}
     public void oper (int param) {}
     void oper (char param) {}
     void oper (short param) {}
 }
```

Tip: Either give such methods or constructors equal visibility or change their signature.

Right

```
public class ClassA {
     ClassA (int param) {}
     public ClassA (char param) {}
     public ClassA (short param) {}
     public void oper (int param) {}
     void doOper (char param) {}
     void doOper (short param) {}
 }
```

ICSBF - Instantiated Classes Should Be Final

This rule recommends making all instantiated classes final. It checks classes, which present in the object model. Classes from search/classpath are ignored.

Wrong

```
class ICSBF {
     private Class1 attr1 = new Class1();
     // something...
 }
 class Class1 {
     // something...
 }
```

Tip: Make all instantiated classes final.

Right

```
class ICSBF {
     private Class1 attr1 = new Class1();
     // something...
 }
 final class Class1 {
     // something...
 }
```

IIMBU - Imported Items Must Be Used

It is not legal to import a class or an interface and never use it. This rule checks classes and interfaces that are explicitly imported with their names - that is not with import of a complete package, using an asterisk. If unused class and interface imports are omitted, the amount of meaningless source code is reduced - thus the amount of code to be understood by a reader is minimized.

Wrong

```
import java.awt.*;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.Stack;
import java.util.Vector;
class IIMBU {
     Dictionary dict;
     void func (Vector vec) {
         Hashtable ht;
         // do something
     }
 }
```

Tip: Delete unnecessary imports.

IIMBU - Imported Items Must Be Used

It is not legal to import a class or an interface and never use it. This rule checks classes and interfaces that are explicitly imported with their names - that is not with import of a complete package, using an asterisk. If unused class and interface imports are omitted, the amount of meaningless source code is reduced - thus the amount of code to be understood by a reader is minimized.

Wrong

```
import java.awt.*;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.Stack;
import java.util.Vector;
class IIMBU {
Dictionary dict;
void func (Vector vec) {
Hashtable ht;
// do something
}
}
```

Tip: Delete unnecessary imports.

LAPAPMF - List All Public And Package Members First

Enforces standard to improve readability. Methods/data in your class should be ordered properly.

Wrong

```
class LAPAPMF {
     private int attr;
     public void oper () {}
 }
```

Tip: Place public and package members first, before protected and private ones.
Note: this rule is rather arguable. On the other hand, grouping members by functionality rather than by scope can also make understanding the code easier.

Right

```
class LAPAPMF {
     public void oper () {}
     private int attr;
 }
```

MFDCSF - Method finalize() Doesn't Call super.finalize()

As it is mentioned in book 'The Java Programming Language' by Ken Arnold and James Gosling, calling of super.finalize() from finalize() is good practice of programming, even if base class doesn't define finalize() method. This makes class implementations less dependent from each other.

Wrong

```
void finalize () {
 }
```

Tip: Always call `super.finalize()`

Right

```
void finalize () {
     super.finalize();
 }
```

MLOWP - Mixing Logical Operators Without Parentheses

An expression containing multiple logical operators together should be parenthesized properly.

Wrong

```
void oper () {
     boolean a, b, c;
     // do something
     if ( a || b && c ) {
          // do something
          return;
     }
 }
```

Tip: Use parenthesis to clarify complex logical expression to reader.

Right

```
void oper () {
     boolean a, b, c;
     // do something
     if ( a || (b && c) ) {
          // do something
          return;
     }
 }
```

MVDWSN - Multiple Visible Declarations With Same Name

Multiple declarations with the same name must not be simultaneously visible excepting for overloaded methods.

Wrong

```
class MVDWSN {
     void index () {
         return;
     }
     void func () {
         int index;
     }
 }
```

Tip: Rename either of members (or variables) with clashing names.

Right

```
class MVDWSN {
     void index () {
         return;
     }
     void func () {
         int anIndex;
     }
 }
```

NAICE - No Assignments In Conditional Expressions

Use of assignment within conditions makes the source code hard to understand.

Wrong

```
if ( (dir = new File(targetDir)).exists() ) {
     // do something
 }
```

Right

```
dir = new File(targetDir);
 if ( dir.exists() ) {
     // do something
 }
```

NC - Naming Conventions

Takes a regular expression and item name and reports all occurrences where the pattern does not match the declaration.

Wrong

```
package _audit;
class _AuditNC {
     void operation1 (int Parameter) {
     void Operation2 (int parameter) {
         int _variable;
     }
     int my_attribute;
     final static int constant;
 }
```

Tip: Rename packages, classes, members and so on in correct manner.

Right

```
package audit;
class AuditNC {
     void operation1 (int parameter) {
     void operation2 (int parameter) {
         int variable;
     }
     int myAttribute;
     final static int CONSTANT;
}
```

NOEC - Names Of Exception Classes

Names of classes, which inherit from Exception, should end with Exception.

Wrong

```
class AuditException extends Exception {}
class NOEC extends Exception {}
```

Tip: Rename exception classes

Right

```
class AuditException extends Exception {}
class NOECException extends Exception {}
```

Section OCMD through TMSSC

OCMD - Order of Class Members Declaration

According to Sun Code Conventions for Java, the parts of a class or interface declaration should appear in the following order

1. Class (static) variables. First the public class variables, then the protected, then package level (no access modifier), and then the private.
2. Instance variables. First the public class variables, then the protected, then package level (no access modifier), and then the private.
3. Constructors
4. Methods

OMNBU - Operator '?:' Should Not Be Used

The operator '?:' makes the code harder to read, than the alternative form with an if-statement.

Wrong

```
void func (int a) {
     int b = (a == 10) ? 20 : 30;
 }
```

Tip: Replace '?:' operator with the appropriate if-else statement.

Right

```
void func (int a) {
     if (a == 10)
         b = 20;
     else
         b = 30;
 }
```

ONAMWAM - Overriding a Non-Abstract Method With an Abstract Method

Checks for the overriding of non-abstract methods by abstract methods in a subclass.

Wrong

```
class Animal {
     void func () {}
 }
 abstract class Elephant extends Animal {
     abstract void func ();
 }
```

Tip: Perhaps this is just an coincidence of names - then just rename your method. If not, either make given method abstract in ancestor or non-abstract in descendant.

Right

```
class Animal {
     void func () {}
 }
 abstract class Elephant extends Animal {
     abstract void extFunc ();
 }
```

OOAOM - Order Of Appearance Of Modifiers

Checks for correct ordering of modifiers.
For classes: visibility (public, protected or private), abstract, static, final.
For attributes: visibility (public, protected or private), static, final, transient, volatile.
For operations: visibility (public, protected or private), abstract, static, final, synchronized, native

Wrong

```
final public class OOAOM {
     public static final int attr1;
     static public int attr2;
 }
```

Tip: Change the order of modifiers

Right

```
public final class OOAOM {
     public static final int attr1;
     public static int attr2;
 }
```

OPM - Overriding a Private Method

A subclass should not contain a method with the same name and signature as in a superclass if these methods are declared to be private.

Wrong

```
class Animal {
     private void func () {}
}
 class Elephant extends Animal {
     private void func () {}
}
```

Tip: Rename descendant class' method

Right

```
class Animal {
     private void func () {}
}
class Elephant extends Animal {
     private void extFunc () {}
}
```

OWS - Overloading Within a Subclass

A superclass method may not be overloaded within a subclass unless all overloadings in the superclass are also overridden in the subclass. It is very unusual for a subclass to be overloading methods in its superclass without also overriding the methods it is overloading. More frequently this happens due to inconsistent changes between the superclass and subclass - i.e. the intention of the user is to override the method in the superclass, but due to the error, the subclass method ends up overloading the superclass method.

Wrong

```
public class Elephant extends Animal {
     public void oper (char c) {}
     public void oper (Object o) {}
 }
 class Animal {
     public void oper (int i) {}
     public void oper (Object o) {}
 }
```

Tip: Overload other methods too.

Right

```
public class Elephant extends Animal {
     public void oper (char c) {}
     public void oper (int i) {}
     public void oper (Object o) {}
 }
 class Animal {
     public void oper (int i) {}
 }
```

PCPTCE - Parenthesize Conditional Part of Ternary Conditional Expression

According to Sun Code Conventions for Java, if an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized.

Wrong

```
return x = 0 ? x : -x;
```

Right

```
return (x = 0) ? x : -x;
```

PDOBB - Put Declarations Only at the Beginning of Blocks

Sun Code Conventions for Java recommends to put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".) Don't wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

Wrong

```
void myMethod() {
     if (condition) {
         doSomeWork();
         int int2 = 0;
         useInt2(int2);
     }
     int int1 = 0;
     useInt1(int1);
 }
```

Tip: Move declarations to the beginning of the block.

Right

```
void myMethod() {
     int int1 = 0; // beginning of method block
     if (condition) {
         int int2 = 0; // beginning of "if" block
         doSomeWork();
         useInt2(int2);
     }
     useInt1(int1); }
```

PFC - Provide File Comments

According to Sun Code Conventions for Java, all source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

```
 /*
 * Classname
 * Version information
 * Date
 * Copyright notice
 */
```

This audit rule verifies whether the file begins with a c-style comment. It may optionally verify whether this comment contains the name of the top-level class given file contains.

PIIFS - Provide Incremental In For-Statement or use while-statement

Checks if third argument of the for-statement is missing.

Wrong

```
for ( Enumeration enum = getEnum(); enum.hasMoreElements();
) {
      Object o = enum.nextElement();
      doSomeProc(o);
 }
```

Tip: Either provide incremental part of the for-structure or cast the for-statement in to a while-statement.

Right

```
Enumeration enum = getEnum();
while (enum.hasMoreElements()) {
      Object o = enum.nextElement();
      doSomeProc(o);
}
```

PJDC - Provide JavaDoc Comments

Checks whether JavaDoc comments are provided for classes, interfaces, methods and attributes. Options allow to specify whether to check JavaDoc comments for public, package, protected or all classes and members.

"Sun Code Conventions for Java" also recommends that `@param` tags order should correspond operation parameters order and `@throws` (or `@exception`) tags should be sorted alphabetically. Turning each of 'ordered' checkbox on makes audit to check tags order.

PMFL - Put the Main Function Last

Tries to make the program comply with various coding standards regarding the form of class definitions.

Wrong

```
public class PMFL {
     void func1 () {}
     public static void main (String args[]) {}
     void func2 () {}
}
```

Right

```
public class PMFL {
     public static void main (String args[]) {}
     void func1 () {}
     void func2 () {}
}
```

PPCF - Place Public Class First

According to Sun Code Conventions for Java, the public class or interface should be the first class or interface in the file.

Wrong

```
class Helper {
     // some code
}

 public class PPCM {
     // some code
}
```

Tip: Place public class or interface first

Right

```
public class PPCM {
     // some code
}

 class Helper {
     // some code
}
```

RFDI - Replacement For Demand Imports

Demand import-declarations must be replaced by a list of single import-declarations that are actually imported into the compilation unit. In other words, import-statements may not end with an asterisk.

Wrong

```
import java.awt.*;
import javax.swing.*;
class RFDI {
     public static JFrame getFrame (Component com) {
          while (com != null) {
               if (com instanceof JFrame)
                    return (JFrame)com;
               com = com.getParent();
          }
          return null;
     }
}
```

Tip: Replace demand imports with a list of single import declarations

Right

```
import java.awt.Component;
import javax.swing.JFrame;
class RFDI {
     public static JFrame getFrame (Component com) {
          while (com != null) {
               if (com instanceof JFrame)
                    return (JFrame)com;
               com = com.getParent();
          }
          return null;
     }
}
```

SBCCS - Supply Break or Comment in Case Statement

According to Sun Code Conventions for Java, every time a case falls through (doesn't include a break statement), a comment should be added where the break statement would normally be. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

Wrong

```
switch( c ) {
     case 'n':
          result += '\n';
          break;
     case 'r':
          result += '\r';
          break;
     case '\'':
          someFlag = true;
     case '\"':
          result += c;
          break;
     // some more code...
 }
```

Tip: Add `/* falls through */` comment where the break statement would normally be.

Right

```
switch( c ) {
     case 'n':
          result += '\n';
          break;
     case 'r':
          result += '\r';
          break;
     case '\'':
          someFlag = true;
          /* falls through */;
     case '\"':
          result += c;
          break;
     // some more code...
 }
```

SSSIDC - Switch Statement Should Include a Default Case.

According to Sun Code Conventions for Java, every switch statement should include a `default` case.

TC - Transparent Collections

For Java language, Together allows users to visualize "transparent" collections in a simple manner. That is, upon reverse engineering legacy code, Together can identify and display associations based on collections without requiring the users to inspect their own code and to then have to manually insert the Together `@association` tag so that the proper diagram link is drawn. TC is auto fix audit and it can put for you association tag into the code.

Here is example of TC results on simple class.

Before TC:

```
import java.util.*;
class A {
    private Vector v = new Vector();
    public void addElement(String element) {
        v.addElement(element);
    }
}
```

After TC:

```
import java.util.*;
class A {
    /**
    * @associates String
    */
    private Vector v = new Vector();
    public void addElement(String element) {
        v.addElement(element);
    }
}
```

TC audit can be adjusted for any set of collections.

The collections are listed in the Classes tab of the table in the options panel. To define a collection, press *Add before* or *Add after* buttons, and enter fully qualified name of the collection class (for example, java.util.Vector). Each collection can be disabled.

It is also possible to delete collections from the set, using *Delete* button.

For the selected collection you can switch to the Operations tab of the table. This tabbed pane enumerates the signatures of operations that add objects to the collection, and the number of parameter that encapsulates the object being added.

By default this table contains fully qualified names of all classes and interfaces of java.util package.

TMSSC - Too Many Switch Statement Cases

Switch statements should not have more than 256 cases.

Some processors have specialized hardware instructions that can take advantage of the presence of less than 256 switch cases to optimize code. It is useful to use this rule therefore.

Section UAAD through UVD

UAAO - Use Abbreviated Assignment Operator

Use the abbreviated assignment operator in order to write programs more rapidly. Also some compilers run faster when you do so.

Wrong

```
void oper () {
     int i = 0;
     i = i + 20;
     i = 30 * i;
 }
```

Right

```
void oper () {
     int i = 0;
     i += 20;
     i *= 30;
 }
```

UC - Unnecessary Casts

Checks for the use of type casts that are not necessary.

Wrong

```
class Animal {}
class Elephant extends Animal {
     void func () {
         int i;
         float f = (float) i;

         Elephant e1;
         Elephant e2 = (Elephant) e1;

         Animal a;
         Elephant e;
         a = (Animal) e;
     }
}
```

Tip: Delete unnecessary cast to improve readability.

UCVN - Use Conventional Variable Names

One-character local variable or parameter names should be avoided, except for temporary and looping variables, or where a variable holds an undistinguished value of a type. Conventional one-character names are:

```
b for a byte
c for a char
d for a double
e for an Exception
f for a float
i, j, k for integers
l for a long
o for an Object
s for a String
```

To avoid potential conflicts please change the names of local variables or parametes that consist of only two or three uppercase letters and coicide with initial country codes and domain names, which could be used as first components of unique package names.

Wrong

```
void func (double d) {
     int i;
     Object o;
     Exception e;
     char s;
     Object f;
     String k;
     Object UK;
 }
```

Tip: Give all local variables conventional names.

Right

```
void func (double d) {
     int i;
     Object o;
     Exception e;
     char c;
     Object o;
     String s;
 }
```

UEIOE - Use 'equals' Instead Of '=='

The '==' operator used on strings checks if two string objects are two identical objects. However, in most situations, one likes to simply check of two strings that have the same value. In these cases, the 'equals' method should be used.

Wrong

```
void func (String str1, String str2) {
     if (str1 == str2) {
         // do something
     }
 }
```

Tip: Replace '==' operator with 'equals' method.

Right

```
void func (String str1, String str2) {
     if ( str1.equals(str2) ) {
         // do something
     }
 }
```

UIOE - Unnecessary 'instanceof' Evaluations

Checks that the runtime type of the left-hand side expression is the same as the one specified on the right-hand side.

Wrong

```
class UIOE {
     void operation () {
         Animal animal;
         Elephant elephant;
         if ( animal instanceof Animal ) {
             doSomething1(animal);
         }
         if ( elephant instanceof Animal ) {
             doSomething2(elephant);
         }
     }
 }
 class Animal {}
 class Elephant extends Animal {}
```

Tip: Remove unnecessary checks

Right

```
class UIOE {
     void operation () {
         Animal animal;
         Elephant elephant;
         doSomething1(animal);
         doSomething2(elephant);
     }
 }
 class Animal {}
 class Elephant extends Animal {}
```

ULIOL - Use 'L' Instead Of 'l' at the end of integer constant

It is difficult to distinct lower case letter 'l' and digit '1'. As far as letter 'l' can be used as long modifier at the end of integer constant, it can be mixed with digit. It is better to use uppercase 'L'.

Wrong

```
void func () {
     long var = 0x00011111l;
 }
```

Tip: Change trailing 'l' letter at the end of integer constants with 'L'.

Right

```
void func () {
     long var = 0x0001111L;
 }
```

ULVAFP - Unused Local Variables And Formal Parameters

Local variables and formal parameters declarations must be used.

Wrong

```
int oper (int unused_param, int used_param) {
     int unused_var;
     return 2 * used_param;
 }
```

Tip: Get rid of unused local variables and formal parameters.

UOOIM - Use Of Obsolete Interface Modifier

The modifier 'abstract' is considered obsolete and should not be used.

Wrong

```
abstract interface UOOIM {}
```

Tip: Remove unnecessary 'abstract' modifier.

Right

```
interface UOOIM {}
```

UOSM - Use Of the 'synchronized' Modifier

The 'synchronized' modifier on methods can sometimes cause confusion during maintenance as well as during debugging. This rule therefore recommends against using this modifier and instead recommends using 'synchronized' statements as replacements.

Wrong

```
class UOSM {
     public synchronized void method () {
         // do something
     }
 }
```

Tip: Use synchronized statements instead of synchronized methods.

Right

```
class UOSM {
     public void method () {
        synchronized(this) {
             // do something
        }
     }
 }
```

UOUIMM - Use Of Unnecessary Interface Member Modifiers

All interface operations are implicitly public and abstract. All interface attributes are implicitly public, final and static.

Wrong

```
interface UOUIMM {
     int attr1;
     public final static int ATTR2;
     void oper1 ();
     public abstract void oper2 ();
 }
```

Tip: Get rid of superfluous interface member modifiers

Right

```
interface UOUIMM {
     int attr1;
     final static int ATTR2;
     void oper1 ();
     void oper2 ();
 }
```

UPCM - Unused Private Class Member

An unused class member might indicate a logical flaw in the program. The class declaration has to be reconsidered in order to determine the need of the unused member(s).

Wrong

```
class UPCM {
     private int bad_attr;
     private int good_attr;
     private void bad_oper () {
         // domething...;
     }
     private void good_oper1 () {
         good_attr = 10;
     }
     public void good_oper2() {
         good_oper1();
     }
 }
```

Tip: Examine the program. If given member is really unnecessary, remove it (or at least comment it out).

URSP - Unnecessary Return statement Parentheses

According to Sun Code Conventions for Java, a return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

Wrong

```
return;
return (myDisk.size());
return (sizeOk ? size : defaultSize);
```

Right

```
return;
return myDisk.size();
return (sizeOk ? size : defaultSize);
```

USAI - Use of Static Attribute for Initialization

Non-final static attributes should not be used in initializations of attributes.

Wrong

```
class ClassA {
     static int state = 15;
     static int attr1 = state;
     static int attr2 = ClassA.state;
     static int attr3 = ClassB.state;
 }
 class ClassB {
     static int state = 25;
 }
```

Tip: Either make static attributes used for initialization final, or use another constants for initialization.

Right

```
class ClassA {
     static int state = 15;
     static final int INITIAL_STATE = 15;
     static int attr1 = INITIAL_STATE;
     static int attr2 = ClassA.state;
     static int attr3 = ClassB.state;
 }
 class ClassB {
     static final int state = 25;
 }
```

UTETACM - Use 'this' Explicitly To Access Class Members

Tries to make you use 'this' explicitly when trying to access class members. Often, using the same class member names with parameters names makes what you are referring to unclear.

Wrong

```
class UTETACM {
     int attr = 10;
     void func () {
         // do something
     }
     void oper () {
         func();
         attr = 20;
     }
 }
```

Right

```
class UTETACM {
     int attr = 10;
     void func () {
         // do something
     }
     void oper () {
         this.func();
         this.attr = 20;
     }
 }
```

UVD - Use Virtual Destructor

C++ only

When should we declare a destructor virtual? Whenever the class has at least one virtual function. Having virtual functions indicate that a class is meant to act as an interface to derived classes, and when it is, an object of a derived class may be destroyed through a pointer to the base. For example:

```
class Base {
// ...
virtual ~Base();
};

class Derived : public Base {
// ...
~Derived();
};

void f()
{
Base* p = new Derived;
delete p; // virtual destructor used to ensure that ~Derived
is called
}
```

Had Base's destructor not been virtual, Derived's destructor would not have been called - with likely bad effects, such as resources owned by Derived not

Source: Bjarne Stroustrup's C++ Style and Technique FAQ

Metrics Reference

AC - Attribute Complexity

Defined as the sum of each attribute's value in the class.

You can set up weights for types and its arrays separately. Use "*" to define types of a package with all its subpackages. For example, "java.lang.*" means that the row defines all classes of the java.lang package and its subpackages. To process all types not listed in the table specify the last row as "*". The row order is important, because checking of attributes goes from the top of the table downward up to the first coincidence.

AHF - Attribute Hiding Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator is the sum of the invisibilities of all attributes defined in all classes. The invisibility of an attribute is the percentage of the total classes from which this attribute is not visible. The denominator is the total number of attributes defined in the project.

AIF - Attribute Inheritance Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator is the sum of inherited attributes in all classes in the project. The denominator is the total number of available attributes (locally defined plus inherited) for all classes.

CBO - Coupling Between Objects

Represents the number of other classes to which a class is coupled. Counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations and local variables, and types from which attribute and method selections are made. Primitive types, types from java.lang package and supertypes are not counted.

Excessive coupling between objects is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

CC - Cyclomatic Complexity

This measure represents the cognitive complexity of the class. It counts the number of possible paths through an algorithm by counting the number of distinct regions on a flowgraph, i.e. the number of if, for and while statements in the operation's body. Case labels of switch statement are counted optionally.

CF - Coupling Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator represents the number of non-inheritance couplings. The denominator is the maximum possible number of couplings in a system.

CDBC - Change Dependency Between Classes

The Change Dependency Between Classes metric (CDBC) is a metric for the measure of coupling. It determines the potential amount of follow-up work to be done in a client class (**CC**) when the server class (**SC**) is being modified in the course of some maintenance activity.

Definition. CDBC between client class **CC** and server class **SC** is defined as:

1. *n* if **SC** is a super class of **CC**,
2. *n* if **CC** has an attribute of **SC**,
3. *j* if **SC** is used in *j* methods of **CC** as local variable or parameter of **CC** method or reference to **SC** method.

where *n* is a number of methods in **CC**.

Note 1. You can qualify single class, classes list, package or package list which contain sever classes.

For example the list:

java.lang.String
java.awt.*

contains **String** class and all package classes of **java.awt** including classes in sub packages.

Note 2. If a class satisfies items 2 and 3 (or 1 and 3) of the metric definition, then CDBC equals n.

Note 3. Unused local variables will have no impact on CDBC.

CR - Comment Ratio

Counts the ratio of JavaDoc and ordinary comments to total lines of code including JavaDoc and ordinary comments. Blank lines may be optionally interpreted as code ones.

DAC - Data Abstraction Coupling

Counts the number of reference types used in the attribute declarations. Primitive types, types from java.lang package and supertypes are not counted.

DOIH - Depth Of Inheritance Hierarchy

Counts how far down the inheritance hierarchy a class or interface is declared. High values imply that a class is quite specialized.

FO - FanOut

Counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations and local variables. Simple types and supertypes are not counted.

HDiff - Halstead Difficulty

This measure is one of the Halstead Software Science metrics. It is calculated as ('Number of Unique Operators' / 'Number of Unique Operands') * ('Number of Operands' / 'Number of Unique Operands').

HEff - Halstead Effort

This measure is one of the Halstead Software Science metrics. It is calculated as 'Halstead Difficulty' * 'Halstead Program Volume'.

HPLen - Halstead Program Length

This measure is one of the Halstead Software Science metrics. It is calculated as 'Number of Operators' + 'Number of Operands'.

HPVoc - Halstead Program Vocabulary

This measure is one of the Halstead Software Science metrics. It is calculated as 'Number of Unique Operators' + 'Number of Unique Operands'.

HPVol - Halstead Program Volume

This measure is one of the Halstead Software Science metrics. It is calculated as 'Halstead Program Length' * Log2('Halstead Program Vocabulary').

LOC - Lines Of Code

This is the traditional measure of size. It counts the number of code lines. JavaDoc and ordinary comments as well as blank lines may be optionally interpreted as code ones.

LOCOM1 - Lack of Cohesion of Methods 1

Takes each pair of methods in the class and determines the set of fields they each access. If they have disjoint sets of field accesses increase the count P by one. If they share at least one field access then increase Q by one. After considering each pair of methods:

```
RESULT = (P > Q) ? (P - Q) : 0
```

A low value indicates high coupling between methods, which indicates high testing effort because many methods can affect the same attributes and potentially low reusability. The definition of this metric was provided by Chidamber and Kemerer in 1993.

LOCOM2 - Lack Of Cohesion Of Methods 2

Counts the percentage of methods that do not access a specific attribute averaged over all attributes in the class. A high value of cohesion (a low lack of cohesion) implies that the class is well designed. A cohesive class will tend to provide a high degree of encapsulation, whereas a lack of cohesion decreases encapsulation and increases complexity.

LOCOM3 - Lack Of Cohesion Of Methods 3

Measures the dissimilarity of methods in a class by attributes.

Consider:
m - number of methods in a class
a - number of attributes in a class
mA - number of methods that access an attribute
EmA - sum of mA for each attribute

Then:
RESULT = 100*(EmA/a-m)/(1-m)

The definition of this metric was proposed by Henderson-Sellers in 1995. Low value indicates good class subdivision implying simplicity and high reusability. High lack of cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

MHF - Method Hiding Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator is the sum of the invisibilities of all methods defined in all classes. The invisibility of a method is the percentage of the total classes from which this method is not visible. The denominator is the total number of methods defined in the project.

MIC - Method Invocation Coupling

Definition (MIC) MIC is the (relative) number of other classes to which certain class sends messages.

$MIC_{norm} = n_{MIC} / (N -1)$

where N is the total number of classes defined in the project, and n_{MIC} is the number of classes to which messages are sent.

Viewpoints. These viewpoints summarize the impact that coupling has on some external attributes.

- **Maintainability.** The maintenance of a strongly coupled class (high MIC value) is more difficult to do because of its dependency on the classes it is coupled to.
- **Comprehensibility.** A strong coupled class is more difficult to be understood as its understanding implies a partial (or sometimes total) understanding of the classes it is coupled to.
- **Error-prone and Testability.** Error-prone for a class is direct proportional with the number of couples to the other classes. Consequently high coupling has a negative impact on testability.

Observations

- The proposed definition of MIC is obviously a normalized one. Although this has advantages, still for some viewpoints, like maintainability, it is more important to operate on the absolute values, i.e. the number of classes to which it is coupled.
- For some viewpoints it might be important to count only the couplings of the system to user-defined classes, i.e. exclude the library classes.

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

MIF - Method Inheritance Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator is the sum of inherited methods in all classes in the project. The denominator is the total number of available methods (locally defined plus inherited) for all classes.

MNOL - Maximum Number Of Levels

Counts the maximum depth of if, for and while branches in the bodies of methods. Logical units with a large number of nested levels may need implementation simplification and process improvement because groups that contain more than seven pieces of information are increasingly harder for people to understand in problem solving.

MNOP - Maximum Number Of Parameters

Displays the maximum number of parameters amongst all class's operations. Methods with many parameters tend to be more specialized and so are less likely to be reusable.

MSOO - Maximum Size Of Operation

Counts the Maximum size of the operations for a class. Method size is determined in terms of cyclomatic complexity, i.e. the number of if, for and while statements in the operation's body. Case labels of switch statement are counted optionally.

NOA - Number Of Attributes

Counts the number of attributes. Inherited attributes may be counted optionally. If a class has a high number of attributes, it may be wise to consider whether it would be appropriate to divide it into subclasses.

NOAM - Number Of Added Methods

Counts the number of operations added by a class. Inherited and overridden operations are not counted. Classes without parents are not processed. The large value of this measure indicates that the functionality of the given class becomes increasingly distinct from that of the parent classes. In this case, it should be considered whether this class should genuinely be inheriting from the parent or if it could be broken down into several smaller classes.

NOC - Number Of Classes

Counts the number of classes.

NOCC - Number Of Child Classes

Counts the number of classes, which inherit from a particular class, i.e. the number of classes in the inheritance tree down from a class. Non-zero value indicates that the particular class is being re-used. However, the abstraction of the class may be poor if there are too many child classes. It should also be stated that the high value of this measure points to the definite amount of testing required for each child class.

NOCON - Number Of Constructors

Counts the number of constructors. You can specify whether to count all constructors or only public, or protected, and so on.

NOIS - Number Of Import Statements

Counts the number of imported packages/classes. This measure can highlight excessive importing and can also be used as a measure of coupling.

NOM - Number Of Members

Counts the number of members, i.e. attributes and operations. Inherited members may be counted optionally. If a class has a high number of members, it may be wise to consider whether it would be appropriate to divide it into subclasses.

NOO - Number Of Operations

Counts the number of operations. Inherited operations may be counted optionally. If a class has a high number of operations, it may be wise to consider whether it would be appropriate to divide it into subclasses.

NOOM - Number Of Overridden Methods

Counts the number of inherited operations, which a class overrides. Classes without parents are not processed. High values tend to indicate design problems, i.e. subclasses should generally add to and extend the functionality of the parent classes rather than overriding them.

NOprnd - Number of Operands

This measure is used as an input to the Halstead Software Science metrics. It counts the number of operands used in a class.

NOprtr - Number of Operators

This measure is used as an input to the Halstead Software Science metrics. It counts the number of operators used in a class.

NORM - Number Of Remote Methods

Processes all methods and constructors and counts the number of various remote methods called. Remote method is defined as a method, which is not declared in the class itself or in its ancestors.

NUOprnd - Number of Unique Operands

This measure is used as an input to the Halstead Software Science metrics. It counts the number of unique operands used in a class.

NUOprtr - Number of Unique Operators

This measure is used as an input to the Halstead Software Science metrics. It counts the number of unique operators used in a class.

PF - Polymorphism Factor

This measure is from the MOOD (Metrics for Object-Oriented Development) suite. It is calculated as a fraction. The numerator is the sum of overriding methods in all classes. This is the actual number of possible different polymorphic situations. A given message sent to a class can be bound, statically or dynamically, to a named method implementation. The latter can have as many shapes (morphs) as the number of times this same method is overridden in that class's descendants. The denominator represents the maximum number of possible distinct polymorphic situations for that class as the sum for each class of the number of new methods multiplied by the number of descendants. This maximum would be the case where all new methods defined in each class would be overridden in all of their derived classes.

PIntM - Percentage of Internal Members

Counts the percentage of internal members in a class.

PPIntM - Percentage of Protected Internal Members

Counts the percentage of protected internal members in a class.

PPkgM - Percentage of Package Members

Counts the percentage of package members in a class.

PPrivM - Percentage of Private Members

Counts the percentage of private members in a class.

PProtM - Percentage of Protected Members

Counts the percentage of protected members in a class.

PPubM - Percentage of Public Members

Counts the proportion of vulnerable members in a class. A large proportion of such members means that the class has high potential to be affected by external classes and means that increased effort will be needed to test such a class thoroughly.

TCR - True Comment Ratio

Counts the ratio of JavaDoc and ordinary comments to total lines of code excluding JavaDoc and ordinary comments. Blank lines may be optionally interpreted as code ones.

TRAP - Total Re-use from Ancestors Percentage

RAP - Reuse from Ancestors Percentage

The RA Metric

Definition 1 (Reuse of Ancestor-class - RA) The RA metric between a class **C** and one of its ancestor classes **A** is formally expressed as:

$$RA(C; A) = \sum_{i=1}^{i=n_C} RDA(mth_i; A)/n_C$$

where mth_i ($i = 1; n_C$) represent the methods defined in class **C**.

Explanations The RA metric quantifies the reuse from a super class by totalizing this reuse from all of its methods. The degree to which a method reuses an ancestor class is variable. We consider that the way this reuse degree is calculated depends on the goals of the measurement. Consequently we decided to "parameterize" the metric with a family of metrics called Reuse Degree of Ancestor-class (RDA), that evaluates this reuse degree. A description of this family of metrics is presented below.

The RDA Metrics

Definition 2 (Reuse Degree of Ancestor-class) A function expressing the measure of reuse of an ancestor class **A** in method mth_i of class **C** is called Reuse Degree of Ancestor-class **A** in method mth_i.

$RDA : SMF_C \text{ X } SAC_C \longrightarrow [0; 1]$

where SMF_C is the set of all member functions (methods) in class **C** and SAC_C is the set of ancestors classes **A** for class **C**.

Percentage RDA - RDA_{perc} The Percentage Reuse Degree of Ancestor-class is defined as:

$$RDA_{perc}(mth_C; A) = \frac{\sum_{i=1}^{n_A^{imp}} uses(m_i^{imp}; mth_C) + (1-k_A) \sum_{j=1}^{n_A^{int}} Uses(m_j^{int}; mth_C)}{n_A^{imp} + (1-k_A)\, n_A^{int}}$$

where m_i^{imp} ($i = 1; n_A^{imp}$) represent the usable class members of **A** belonging to the implementation of the class and m_j^{int} ($j = 1; n_A^{int}$) represent the class members of **A** belonging to the interface of the class. Function uses is defined as:

$$uses(m_A; mth_C) = \begin{cases} 1 \text{ if class member } m_A \text{ is used in method } mth_C \\ 0 \text{ if not} \end{cases}$$

Stability Factor is defined as follows. A class A is stable if most of the changes to the implementation of A can be performed without affecting its interface. Thus, we define a stability factor kA , kA [0; 1], as the quantitative expression of the stability of class A.

Observations: Because the stability of the ancestor-class plays an important role from the perspective of the client class, we proposed this definition of RDA that also considers the stability of ancestors interface.

The Total RA Metric - TRA

As we have seen the RA metric has two parameters: a particular class and one of its ancestor classes. We think that it is necessary to have also a metric that expresses the total reuse (from all the ancestors) for a given class. We will base the definition of this new metric, on the definition of the already defined RA metric.

Definition 3 (Total Reuse from Ancestors - TRA) The Total Reuse from Ancestors metric for a class **C** is defined as the sum of all RA values between class **C** and its superclasses. This can be formally expressed as:

$$TRA(C) = \sum_{i=1}^{i=n_A} RA(C; A_i)$$

where n_A represents the number of ancestor-classes for class **C**, and A_i is the iterator of its ancestor classes.

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

TRAU - Total Re-use from Ancestors Unitary

RAU - Re-use from Ancestors Unitary

Reuse of Ancestors

The RA Metric

Definition 1 (Reuse of Ancestor-class - RA) The RA metric between a class **C** and one of its ancestor classes **A** is formally expressed as:

$$RA(C; A) = \sum_{i=1}^{i=n_C} RDA(mth_i; A)/n_C$$

where mth_i ($i = 1; n_C$) represent the methods defined in class **C**.

Explanations: The RA metric quantifies the reuse from a super class by totalizing this reuse from all of its methods. The degree to which a method reuses an ancestor class is variable. We consider that the way this reuse degree is calculated depends on the goals of the measurement. Consequently we decided to "parameterize" the metric with a family of metrics called Reuse Degree of Ancestor-class (RDA), that evaluates this reuse degree. A description of this family of metrics is presented below.

The RDA Metrics

Definition 2 (Reuse Degree of Ancestor-class) A function expressing the measure of reuse of an ancestor class **A** in method mthi of class **C** is called Reuse Degree of Ancestor-class **A** in method mth_i.

RDA : SMF_C X SAC_C -- [0; 1]

where SMF_C is the set of all member functions (methods) in class **C** and SAC_C is the set of ancestors classes **A** for class **C**.

Unitary RDA - RDA_{unit} The Unitary Reuse Degree of Ancestor-class is defined as:

$$RDA_{unit}(mth_C; A) = \begin{cases} 1 \text{ if method } mth_C \text{ uses at least one member of class } \mathbf{A} \\ 0 \text{ if method } mth_C \text{ uses no member of class} \end{cases}$$

The Total RA Metric - TRA

As we have seen the RA metric has two parameters: a particular class and one of its ancestor classes. We think that it is necessary to have also a metric that expresses the total reuse (from all the ancestors) for a given class. We will base the definition of this new metric, on the definition of the already defined RA metric.

Definition 3 (Total Reuse from Ancestors - TRA) The Total Reuse from Ancestors metric for a class **C** is defined as the sum of all RA values between class **C** and its superclasses. This can be formally expressed as:

$$TRA(C) = \sum_{i=1}^{i=n_A} RA(C; A_i)$$

where n_A represents the number of ancestor-classes for class **C**, and A_i is the iterator of its ancestor classes.

You can see not only the total sum of the metric for a class, but also summands for all ancestors classes of the class in reference. The table of summands is available from the metrics results table by clicking right mouse button. If there are any ancestors for this class the results table speedmenu will include RAu command

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

TRDP

Reuse in Descendants

The RD Metric

Definition 1 (Reuse in Descendant-class - RD) The RD metric between a class **C** and one of its descendant classes **D** is formally expressed as:

$$RD(C; D) = \sum_{i=1}^{i=n_C} RDD(m_i;D) / n_C$$

where mi (i = 1; n_C) represent the usable class members of **C**.

Explanations The RD metric quantities the totalized reuse of all the members of a class **C**, in one of its descendant classes. The degree to which a particular member is reused in a descendant class is variable. We consider that the way this reuse degree is calculated depends on the goals of the measurement. Analogous to RA we decided to "parameterize" the metric with a family of metrics called Reuse Degree in Descendant-class (RDD), that quantities this reuse degree. A description of this family of metrics is presented below.

The RDD Metrics

Definition 2 (Reuse Degree in Descendant Class)A function expressing the measure of reuse of a class member m_C class **C** in a descendent class **D** is called Reuse Degree of m_C in Descendant-class **D**.

RDD : SM_C X SDC_C--> [0; 1]

where $_C$ is the set of all members in class **C** and SDC_C is the set of descendant classes **D** for class **C**.

Percentage RDD - RDD_{perc} The Percentage Reuse Degree in Descendant Class is defined as:

$$RDD_{perc}(m_C; D) = \sum_{i=1}^{i=n_D} uses(m_c; mth_D^{(i)})/n_D$$

where n_D is the number of methods in class **D**. Function uses is defined as:

$$uses(m_A; mth_C) = \begin{cases} 1 \text{ if class member } m_A \text{ is used in method } mth_C \\ 0 \text{ if not} \end{cases}$$

The Total RD Metric - TRD

In the previous sections we defined the RD metric with two parameters: a particular class and a descendant of that class. In the same way we defined TRA we also considered that it is necessary to define a metric that expresses the total value for the reuse of a class by all its descendants.

We considered following two viewpoints for the interpretation of this metric.

1. **Maintainability.** A high TRD value for a class indicates that a change in that class has a high impact on the underlying class-hierarchy, i.e. its descendants.
2. **Degree of Member Reuse.** A high TRD for a class indicates that the very most of its members are reused in the sub-classes.

In proposing these two viewpoints, we observed that because their focus is strongly different it would be quite impossible to have a single definition for TRD. So we proposed a definition for each one of the two viewpoints:

Definition 3 (Descendants-based Definition of TRD) The Total Reuse in Descendants metric for a class **C** is defined as the sum of all RD values between class **C** and its descendants. This can be formally expressed as:

$$TRD(C) = \sum_{i=1}^{i=n_D} RD(C; D_i)$$

where n_D represents the number of descendant-classes for class **C**, and D_i is the iterator of its descendants.

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

TRDU

Reuse in Descendants

The RD Metric

Definition 1 (Reuse in Descendant-class - RD) The RD metric between a class **C** and one of its descendant classes **D** is formally expressed as:

$$RD(C; D) = \sum_{i=1}^{i=n_C} RDD(m_i;D) / n_C$$

where mi (i = 1; n_C) represent the usable class members of **C**.

Explanations The RD metric quantities the totalized reuse of all the members of a class **C**, in one of its descendant classes. The degree to which a particular member is reused in a descendant class is variable. We consider that the way this reuse degree is calculated depends on the goals of the measurement. Analogous to RA we decided to "parameterize" the metric with a family of metrics called Reuse Degree in Descendant-class (RDD), that quantities this reuse degree. A description of this family of metrics is presented below.

The RDD Metrics

Definition 2 (Reuse Degree in Descendant Class)A function expressing the measure of reuse of a class member m_C class **C** in a descendent class **D** is called Reuse Degree of m_C in Descendant-class **D**.

RDD : SM_C X SDC_C--> [0; 1]

where $_C$ is the set of all members in class **C** and SDC_C is the set of descendant classes **D** for class **C**.

Unitary RDD - RDD_{unit} The Unitary Reuse Degree in Descendant-class is defined as:

$$RDD_{unit}(m_C; D) = \begin{cases} 1 \text{ if } m_C \text{ is used in at least one method of class } \mathbf{D} \\ 0 \text{ if } m_C \text{ is not used at all in class } \mathbf{D} \end{cases}$$

The Total RD Metric - TRD

In the previous sections we defined the RD metric with two parameters: a particular class and a descendant of that class. In the same way we defined TRA we also considered that it is necessary to define a metric that expresses the total value for the reuse of a class by all its descendants.

We considered following two viewpoints for the interpretation of this metric.

1. **Maintainability.** A high TRD value for a class indicates that a change in that class has a high impact on the underlying class-hierarchy, i.e. its descendants.
2. **Degree of Member Reuse.** A high TRD for a class indicates that the very most of its members are reused in the sub-classes.

In proposing these two viewpoints, we observed that because their focus is strongly different it would be quite impossible to have a single definition for TRD. So we proposed a definition for each one of the two viewpoints:

Definition 3 (Descendants-based Definition of TRD) The Total Reuse in Descendants metric for a class **C** is defined as the sum of all RD values between class **C** and its descendants. This can be formally expressed as:

$$TRD(C) = \sum_{i=1}^{i=n_D} RD(C;D_i)$$

where n_D represents the number of descendant-classes for class **C**, and D_i is the iterator of its descendants.

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

VOD

Law of Demeter

Definition 1 (Client) Method M is a client of method f attached to class C if inside M message f is sent to an object of class C or to C. If f is specialized in one or more subclasses, then M is only a client of f attached to the highest class in the hierarchy. Method M is a client of some method attached to C.

Definition 2 (Supplier) If M is a client of class C then C is a supplier to M. In other words, a supplier class to a method is a class whose methods are called in the method.

Definition 3 (Acquaintance Class) A class C1 is an acquaintance class of method M attached to class C2, if C1 is a supplier to M and C1 is not one of the following:

1. the same as C2;
2. a class used in the declaration of an argument of M
3. a class used in the declaration of an instance variable of C2

Definition 4 (Preferred-acquaintance Class) A preferred-acquaintance class of method M is either:

1. a class of objects created directly in M, or
2. a class used in the declaration of a global variable used in M.

Definition 5 (Preferred-supplier class) Class B is called a preferred-supplier to method M (attached to class C) if B is a supplier to M and one of the following conditions holds:

- B is used in the declaration of an instance variable of C,
- B is used in the declaration of an argument of M, including C and its superclasses,
- B is a preferred acquaintance class of M.

The class form of Demeters Law has two versions: a strict version and a minimization version. The strict form of the law states that every supplier class of a method must be a preferred supplier.

The minimization form is more permissive than the first version and requires only to minimize the number of acquaintance classes of each method.

Observations

- The motivation behind the Law of Demeter is to ensure that the software is as modular as possible. The Law effectively reduces the occurrences of certain nested message sends and simplifies the methods.
- The definition of the Law makes a difference between the classes associated with the declaration of the method and the classes used in the body of the method, i.e. the classes associated with its implementation. The former includes the class where the method is attached, its superclasses, the classes used in the declarations of the instance variables and the classes used to declare the arguments of the method. In some sense, there are an 'automatic' consequence of the method declaration. They can be easily derived from the code and shown by a browser. All other supplier classes to the methods are introduced in the body of the function, that means these couples were created at the time of concretely implementing the method. They can only be determined by a careful reading of the implementation.

Violations of Demeters Law - VOD

The definition of this metric, is based on the minimization form of the Law of Demeter.

Based on the concepts defined there, and remembering that the minimization form of Demeters Law requires that the number of acquaintance classes should be kept low, we define the VOD metric.

Definition 6 (VOD Metric) Being given a class C and A the set of all its acquaintance classes,

VOD(C) = |A|

Informally, VOD is the number of acquaintance classes of a given class.

Keeping the VOD value for a class low offers a number of benefits, enumerated below:

1. Coupling control. A project with a low VOD values is the sign of a minimal "use" coupling between abstractions. That means that a reduced number of methods can be invoked. This makes the methods more reusable.
2. Structure hiding. Reducing VOD represents in fact the reducing of the direct retrieval of subparts of the "part-of" hierarchy. In other words, public members should be used in a restricted way.
3. Localization of information. A low VOD value also means that the class information is localized. This reduces the programming complexity.

Source: Ing. Radu Marinescu. *An Object Oriented Metrics Suite on Coupling.* Universitatea "Politehnica" Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software. September, 1998.

RFC - Response For Class

The size of the response set for the class includes methods in the class's inheritance hierarchy and methods that can be invoked on other objects. A class, which provides a larger response set, is considered to be more complex and required testing efforts than one with a smaller overall design complexity. This measure is calculated as 'Number of Local Methods' + 'Number of Remote Methods'.

WMPC1 - Weighted Methods Per Class 1

This metric is the sum of the complexity of all methods for a class, where each method is weighted by its cyclomatic complexity. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. Only methods specified in a class are included, that is, any methods inherited from a parent are excluded.

WMPC2 - Weighted Methods Per Class 2

This metric is intended to measure the complexity of a class, assuming that a class with more methods than another is more complex, and that a method with more parameters than another is also likely to be more complex. Only methods specified in a class are included, that is, any methods inherited from a parent are excluded.

QA Audit/Metrics Command Mode

It is possible to run QA audits or metrics from the command line. This is useful for including QA information as part of an automated daily build or other process. This topic documents the command-line syntax and options.

Usage:

The following syntax runs QA modules in console mode (i.e., no GUI)

`TgStarter -script:com.togethersoft.modules.qa.QA [`*`options`*`] PrjName `*`[switch]`*

Where:

TgStarter is:
`%TgHome%\bin\Together.bat`, `%TgHome%\bin\TogetherCon.exe`
or
`%TgHome%\bin\Together.exe -con`

PrjName is a fully qualified project name, e.g.
`%TgHome%\samples\java\CashSales\CashSales.tpr`

[switch] is an optional switch `-con` for `Together.exe` (Windows only) to suppress GUI and redirect output to console

Note: Specify paths above using either " \ " or " / " as required by your operating system.

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 "-", 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 "-", it specifies the reverse order. abbr (metric abbreviation) item (default)

Name	Description
	optset = file containing the previously saved set of options (default is current.mts)
-pkg [pkg1 [pkg2 [...]]]	Process specified package(s) only
-cls [cls1 [cls2 [...]]]	Process specified class(es) only
-dcpy:[*directory*]	Target folder where the description is copied. This option is only valid for HTML format.
-dref:[*html-ref*]	Reference to this folder in HTML file. This option is only valid for HTML format.
-fmt:[*tab*|*align*|*html*|*report*]	Output format: *tab* = separate columns by tabs (default) *align* = align columns with spaces *html* = generate HTML file *report* = create HTML report (for the Metrics only)

Sets of options have the default extensions: `*.adt` for the Audit, and `*.mts` for the Metrics. If no path is specified for the options file name, it is sought for in the current folder, and then in the default folder where the settings are stored (`%TgHome%\modules\com\togethersoft\modules\qa\config`). This path is specified absolutely, or relatively to the default location of the settings.

XP Test Support

Together integrates JUnit to provide a universal testing tool, which allows to write tests for the code, pinpoint bugs, catch them with tests, and fix them. You can iterate through this bug-hunting process as long as necessary to produce spotless code.

XP support is implemented as an activatable module XPTest, and thus requires to be checked in the list of Activatable modules on the Options menu.

Together's XPTest allows to easily create the test incremental case templates, edit them and fill in with the specific business logic. Every time the source code changes, you can refactor the relevant test cases. You can generate a test suite, extend it as necessary and keep it running.

Presently, testing only applies to Java projects. In future, it is planned to extend this feature to the other languages.

Using JUnit integration

To make use of XPTest, you have to select this command in the list of Activatable modules. Being activated, the module adds `$TGH$/lib/junit/junit.jar` and `$TGH$/lib/junitx/junitx.jar` paths to the Search/Classpath of the Java project, *XPTest* node to the *Options* dialog, and *XP* node to the *Tools* menu and the diagram speedmenu. This node contains the following commands:

- Test
- Create test case
- Create test proxy
- Create test package
- Configure

Note that testing only applies to a compiled code. Hence, choose *Rebuild Node* command on the *Tools* speedmenu, or use keyboard shortcut CTRL+SHIFT+F8.

Configuring the testing environment

In order to set up the testing environment according to the specific demands of your case choose XP node in the Options dialog. Refer to Configuring XP Test for details.

Creating test cases and test suites

Create test case command allows to generate test class for the selected class on the diagram. Together generates a test case and stores it to the package defined in the configuration dialog.

> **Note**: It is possible to open the test folder in a new tab: right click on the test package icon and choose Open in New tab to view the diagram of test classes.

If the Boolean property *Access Private* is checked, test proxy is generated together with the test case and special test methods are added to provide access to the private and protected methods of the tested class.

Switch to the test diagram, generated by *Create test case* command. On the speedmenu of the test case choose *Create test package* command to generate a so-called test suite that combines all classes to be tested in a single object.

There is an alternative way to create test cases:

1. With XPTest module activated, choose *Class by Pattern* on the diagram toolbar and click on diagram to invoke Choose Pattern dialog.

2. Expand XP node in the pattern treeview. The possible options are TestCase,
3. Select the required pattern, make

Running the test

It is essential that testing only applies to a compiled code. Having rebuilt the node, you can start actual testing. *Test* command launches JUnit environment. Refer to JUnit documentation for usage details.

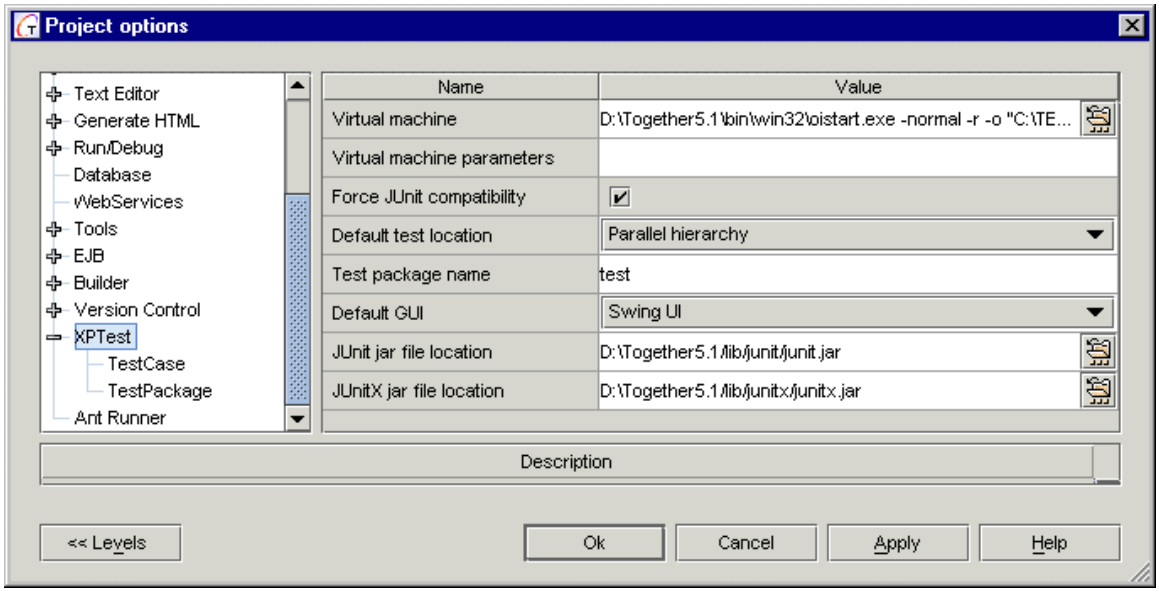
Configuring XPTest

Use the *Options* dialog to create the necessary XPTest configuration for the Default, Project or Diagram level. This is how it's done.

Make sure that XPTest module is activated. This adds *XPTest* branch to the end of the options tree. Open your project and invoke *Options* dialog from the main menu.

Set the required values in the XPTest branch and its sub-branches, or accept the defaults. The options are briefly described below.

Expand the *XPTest* node to see the options:



Virtual machine - location of *java.exe* or *javaw.exe* that will be used to launch JUnit. Together uses *oistart*. tools launcher to launch external applications. In Windows environment the default command for VM usually is:

```
$TGH$\bin\win32\oistart.exe -normal -r -o <temp file 1> -e <temp file 2>
$java.home$/bin/javaw
```

<temp file> parameters passed to *oistart* are the temporary files in the TEMP directory.

In Unix the following line is used:

```
java
```

The names of the temporary files are generated automatically.

Virtual machine parameters - parameters required for the launching of the Virtual Machine

Force JUnit compatibility - by default JUnit only generates test classes and methods for public classes/methods. To access private classes/members switch this option off, this gives you access to the menu options *Create test proxy* and *Create test package*

Default test location - here there are three choices:

> **Parallel hierarchy** - at the top level of the project creates a package, the name of which is set from the **Test package name** below. Beneath this package is a hierarchy that matches the hierarchy of Class the Test Case is being created for.
> **Same package** - create the Test Case in the same package
> **Subpackage** - uses the **Test package name** below to create a package at the same level as the Class, into which it places the Test Case.

Test package name - the name it should use when creating the test package

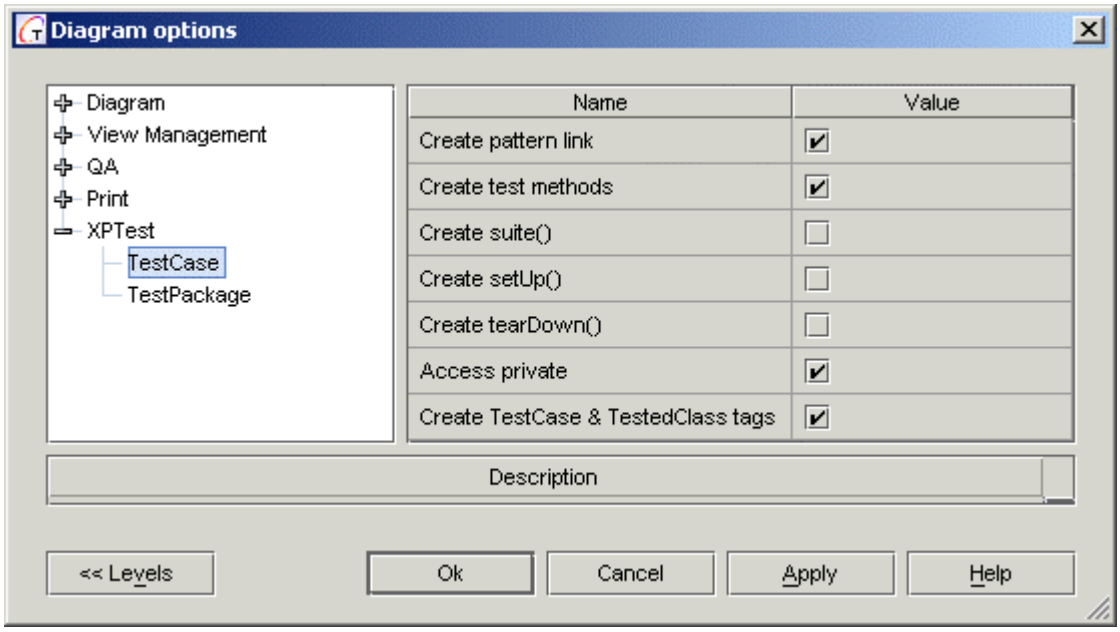
Default GUI - should it use the SwingUI or TextUI when launching JUnit

JUnit jar file location- the location of the *junit.jar* file

JUnitX jar file location- the location of the *junitx.jar* file

> **Tip:** If you don't want Junit/Junitx archives to be added to the Search/Classpath of your project, replace the default values of archive file locations with empty strings ("").

This branch also contains *TestCase* and *TestPackage* branches:



Create pattern link - creates a link between the Class being tested and the TestCase

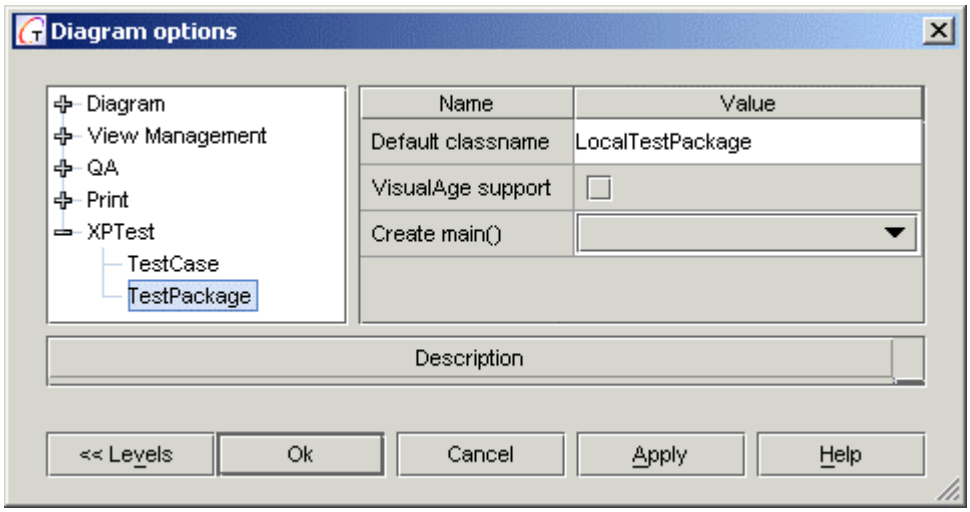
Create test methods - creates test methods in the TestCase

Create suite() - this is a method defined in JUnit; returns an instance of TestSuite class

Create setUp() and **Create tearDown()** - to be checked if test methods from a TestCase are used several times during testing an application

Access private - provides access to private methods to get them tested

Create TestCase & TestClass tags - with this option the Class gets the `@testcase` tag and in the Test Class the `@testedclass` tag is inserted



Default classname - the default name of the Class that is created when **Create test package** is selected

VisualAge support - adds a method to the generated Class that checks to see if VisualAge for Java is running

Create main() - should it create a Main() method in the Class that it generates when **Create test package** is selected?

Refer to **www.junit.org** for JUnit for more detailed descriptions of TestCase options. PrivateTestCase, TestPackage, access private methods are supported by Andreas Heilwagen (**www.extreme-java.de**).

J2EE Support. Rapid Development of Distributed and eCommerce Applications

J2EE Support

Overview of e-Commerce development Features

Development of e-commerce applications is now one of the most promising trends in the development of distributed client-server applications. Main requirements for such applications are:

- Information security and safety;
- Rapid and safe client-server connection;
- Effective implementation of business logic;
- High-quality GUI design.

In order to address these issues, developers usually use EJBs to implement business logic, and HTML pages, JSPs (HTML with Java code), servlets, or applets for the interfaces. Thus software product can include EJBs, servlets, JSPs, applets, HTML pages, GUI files, java-classes etc. User application archive is placed to an application server that supports access, connection and operation of the client.

The most time-consuming tasks involve creating Deployment Descriptor and performing deployment process. Together makes these tasks nice and easy: now it is possible to generate Deployment Descriptor *.xml file and to perform deployment automatically. For this purpose Together provides three types of diagrams: EJB Assembler diagram, Web Application diagram and Enterprise Application diagram.

> **Note:** e-Commerce features are implemented as an activatable module. To make use of it, check *ecommerce* option in the Options / Activatable modules menu.

J2EE Support

Together supports Java 2 Platform, Enterprise Edition (J2EE) specifications that enable easy implementation of highly available, secure, reliable and scalable e-commerce applications. You can familiarize yourself with J2EE specification at **http://java.sun.com/j2ee**.

Together efforts to create a standard application model for developing multi-tier, thin-client services (J2EE Application Model), standard platform for hosting J2EE applications (J2EE Platform), J2EE Compatibility Test Suite and J2EE Reference Implementation.

According to the J2EE specification, there are four application component types:

- *Application clients* (maybe GUI) running on a desktop computer,
- *Applets or simple HTML pages* typically running in a web browser, but sometimes in a variety of other applications or devices that support applet programming model,
- *Web components* (Servlets and pages created with the use of JavaServer Pages technology) that typically run on a web server and respond to HTTP requests from web clients,
- *Enterprise JavaBeans* (EJB) components, which typically contain business logic for a J2EE application and run in a managed environment that supports transactions.

These application components are divided by J2EE specification into three categories:

- Components that are deployed, managed, and executed on a J2EE server (JavaServer Pages, Servlets, and Enterprise JavaBeans),
- Components that are deployed and managed on a J2EE server, but are loaded to and executed on a client machine (HTML pages and applets embedded in the HTML pages),
- Components whose deployment and management is not completely defined by J2EE specification (Application clients).

Together supports the projects that include the following components:

- Web files (HTML, GIF etc.);
- Servlets and JSPs;
- EJBs;
- Classes required for the Servlets, JSPs and EJBs.

J2EE oriented diagrams

Together provides three types of J2EE oriented diagrams, with the business being realized via Enterprise JavaBeans and the user's interface being implemented via Web Applications:

Web Application Diagram enables visual creation of appropriate description and a Web Application archive (WAR) that stores all JSPs', Servlets' and WebFiles diagram elements (HTML, GIF, classes).

EJB Assembler diagram enables visual creation of a JAR archive that stores all EJBs (Entity, Session and Message-driven Beans) and classes required for the EJBs (exceptions, utility classes, etc.).

Enterprise Application diagram is a general (plural) form of WAR and JAR modules. Enterprise Application (EAR) diagram enables combining as many WARs and JARs, as the user wants to include into the EAR. Enterprise Application diagram can be also used for modeling of global security constraints (definition of the global Security Roles used among EJB/WAR modules).

Support of References

According to J2EE specification, Together supports the following references:

- EJB references (references to another EJBs),
- Security references (references to possible users' groups) with different access rights,
- Resource references,
- Environment references (references to constants in the environment).

Information about the references is saved in the Deployment Descriptor. EJB can request about the current status of transactions, security, links between EJBs etc via the descriptor context. To use the references mechanism, the developer has to define the corresponding reference as an additional attribute to the class, using Properties Inspector, or *New | EJB Reference* command of the diagram speedmenu.

Each reference has its own name and value (in quotes). The value of the attribute is something used in lookup, and the attribute name can be used instead of quoted string in lookup statement. Together may have all properties of references as attribute properties . Disadvantage of the approach is that the properties can vary in different bean implementations. It is possible to define intermediate visual *Reference* elements for various types of references.

EJB Reference support

According to J2EE specification, the concept of EJB Reference is very useful for providing connection between EJBs.

- One EJB can be regarded as a client with respect to another EJB, i.e. one EJB can refer to one or more EJBs. The source EJB and destination EJB can reside in the same jar file, or in different JAR files, but in the same J2EE block.
- Every EJB Reference has a set of its own properties. EJB Reference's properties can be defined manually, or linking by graphical linking to the necessary EJB. In the latter case all EJB's properties are passed to this EJB Reference.
- Two EJBs (one referring to another) can't be linked directly. For this purpose a special visual component should be used, namely EJB Reference component, that refers to the appropriate EJB and is used to access this EJB. In this case the first EJB gets properties of the second EJB via the intermediate EJB Reference component.
- Using separate visual EJB Reference element, it is possible to redirect links among EJBs.

    Let us consider two EJBs: EJB1 and EJB2. EJB Reference from EJB1 to EJB2 is defined in the Class diagram. Then the shortcut to EJB1 is added to EJB Assembler diagram. We can use an intermediate visual EJB Reference element to redirect the link from EJB1 to some EJB3. Thus in this example, if Deployment Expert is called from the Class diagram, there will be a reference from EJB1 to EJB2, but if Deployment Expert is called from the EJB Assembler diagram, there will be a reference from EJB1 to EJB3.

- It is possible to graphically draw EJB references, without having EJBs in project at all.

    The user can develop a WebApplication and, instead of importing actual EJB components to the diagram, create an EJB Reference element, specify properties (home, remote interfaces, etc. in the object inspector) and draw a link from Servlet to EJB Reference. This information will be stored in the Deployment Descriptor.

Defining an EJB Reference via visual design element with own properties enhances possibilities.

This element can be more flexibly used as a connection unit for establishing references between EJBs. In this case same EJB-element can be used in different applications without changes, but its properties can be modified through the corresponding EJB Reference element.

Security Reference Support

J2EE specification states that "... the J2EE authorization model is based on the concept of security roles. A security role is a logical grouping of users that is defined by an Application Component Provider or Assembler. It is then mapped by a Deployer to security identities (e.g., principals, groups, etc.) in the operational environment. A security role can be used either with declarative security or with programmatic security".

To provide control access to an EJB-method, the declarative authorization specified in the Deployment Descriptor is used. In this case the necessary EJB-method is associated with a Method-Permission element of Deployment Descriptor, which contains a list of methods accessed by the users with certain security role. If the principal (or group) has the security

role that allows access to this EJB-method, security system allows the principal to call and execute this EJB-method. Same technique is used for Web resources' protection. J2EE specification makes provisions for two variants of security realization:

Deployer maps security role to a user group in the operational environment, or

Deployer maps security role to a principal name in the security policy domain.

In the latter case the principal name of the calling principal is retrieved from its security attributes. Therefore Principal and Security Role are defined as separate design elements in Together. Principal is an Actor element separated from its security features. Principal can be linked to Security Role.

Security Role Reference defined as one of EJB attributes can be used for linking Security Role with Servlet/JSP. Security Role can be linked with EJB via Security Role Reference in EJB implementation class.

Resource Reference Support

Resource Reference can be also defined as an EJB's attribute or as a separate design element (Resource). This element has all properties of the referenced resource (for example, database object). To define Resource Reference's, the developer should know JNDI-name of this resource (res-ref-name in Together), resource type (res-type, i.e. which interface is used to work with this resource), creation mode of the resource manager (res-auth, Bean managed or Container managed). If the user creates a reference to a database resource, he/she will gain access to the corresponding tables.

Resource Reference can be linked with an EJB via Resource Reference in EJB implementation class. The user can change Resource's properties using special design element without making any changes to EJB's properties.

Environment Reference Support

Environment References are actually static constants that cannot be changed after EJB's deployment. Each Environment Reference has its type, value and name. In Together, Environment Reference is defined as an EJB's attribute or as a separate design element - Environment. This element has its own properties.

Environment Reference can be linked with EJB via Environment Reference in EJB implementation class. The user can change Environment properties using special design element without any changes in the EJB's properties.

Process overview

If you have business-logic implemented with EJBs, you can create EJB Assembler diagram and add EJB shortcuts. You can edit EJB Assembler diagram to allow for resources, environment variables, security roles, links between elements and etc. You can also use the J2EE Deployment Expert for generating Deployment Descriptor and deploying a JAR archive.

If you have the user interface realized with JSPs and Servlets, you can create Web Application diagram and add shortcuts to JSPs, Servlets and EJBs. You can create Enterprise Application diagram and add shortcuts to the necessary Web Application and EJB Assembler diagrams.

You can edit Enterprise Application diagram and use the Deployment Expert to generate Deployment Descriptor and deploy archive EAR.

You can find a comprehensive real-life example in EJB Deployment Step by Step under Server-Specific Examples section of this manual.

See also

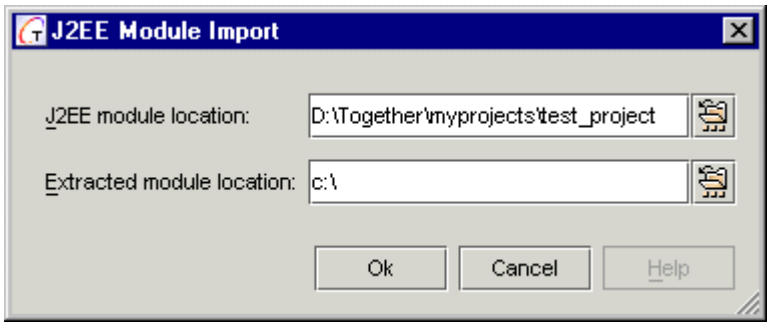
Creating diagrams in projects
Drawing diagram elements
Opening diagrams
Working with View Management
EJB Assembler diagram
Web Application diagram
Enterprise Application diagram

J2EE Module Import

Together extends reverse engineering to the archive files. It is possible to restore diagram structure and source code based on the existing `*.jar, *.ear` or `*.war` file.

This is how it's done...

On the Tools menu choose *J2EE Module Import* command. This brings in the following dialog:



Using the file chooser buttons, specify the archive file to be reverse engineered, enter the target folder for the extracted module and click *OK* to complete operation. Together restores appropriate diagram and generates source code.

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

> **Note**: As of this writing, J2EE module import is subjected to certain limitations. In particular, Security Constraints from *.war and Method Permissions from *.jar archives cannot be imported to diagram.

See also

EJB Assembler diagram
Web Application diagram
Enterprise Application diagram

Creating, Developing and Debugging Distributed Applications

Using Together, you can perform full cycle of developing and debugging the distributed applications.

Creating, Developing and Debugging Servlets

Servlets are used to implement client-server interaction. Normally, this is http-based interaction between an Internet browser and a Web server. Any application capable of creating http-requests can also play the role of the client, for example java applet or java application.

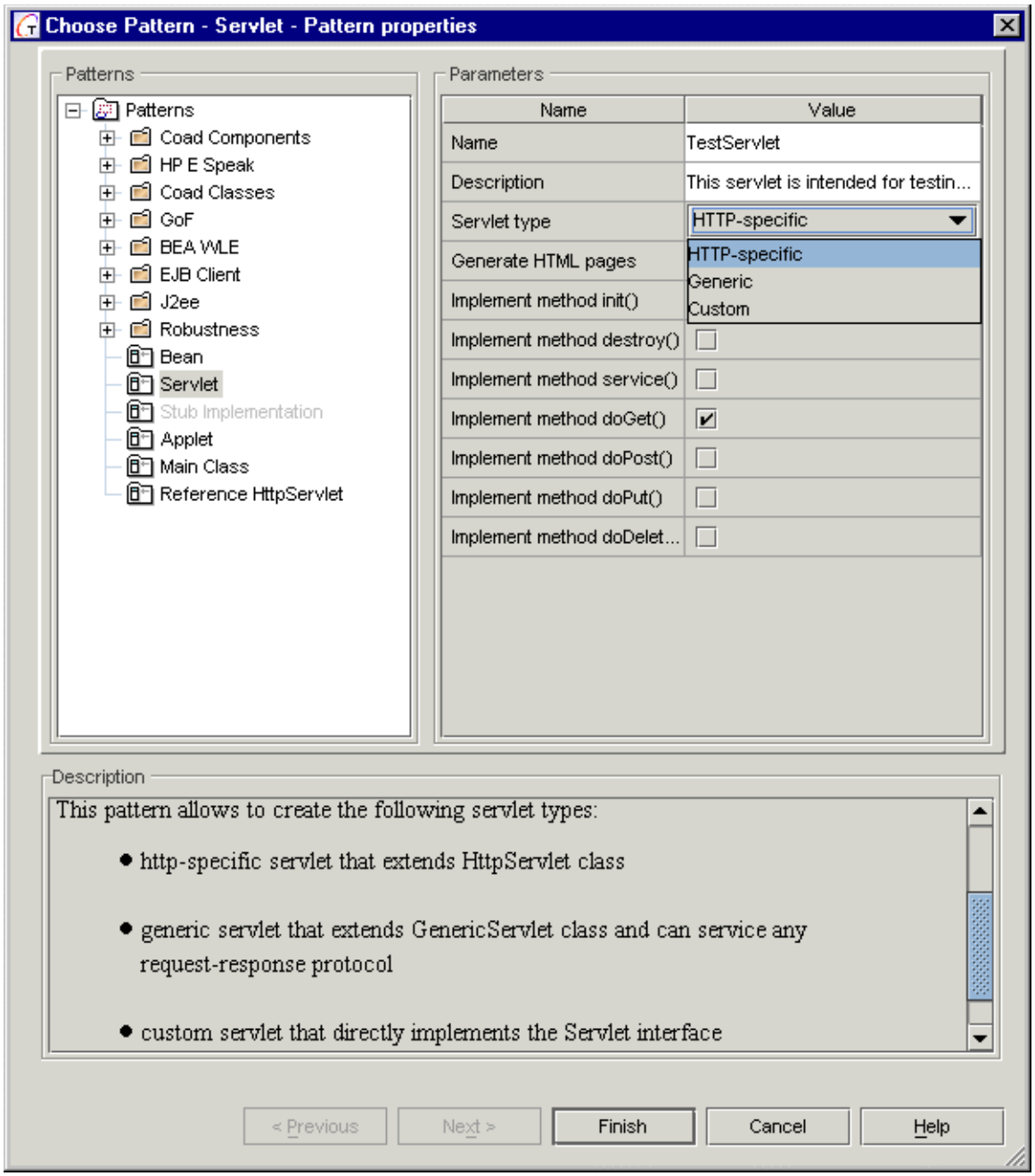
A servlet is a java class on the server side, that implements standard interface for handling http requests. It should be noted that there may be multiple classes on the server side to handle http requests. However, the servlet only gets direct requests from the client.

Amount of servlets allocated on the server is not determined. A single servlet can cater for one type of requests, or for multiple types. Hence, it is possible to create different configurations: allocate one servlet on the server to handle all incoming requests, or allocate numerous servlets for each request type. The solution is stipulated by specific task and developer's preferences.

Together provides a convenient way to create servlets using patterns. To create a servlet, use Class by Pattern command from the diagram toolbar or speedmenu. This invokes Choose Pattern dialog.

There are two servlet patterns available in *Together*. *ReferenceHttpServlet* pattern generates a skeleton of a basic servlet that extends HttpServlet. *Servlet* pattern allows to generate three types of servlets: Generic Servlet that extends GenericServlet class and can service any request-response protocol; Http Servlet that extends HttpServlet class; and Custom Servlet that directly implements the Servlet interface. This enables the user to create servlets in accordance with the specific task.

Parameters of the Choose Pattern dialog allow to select the methods to be generated in the body of the class. This spares the developers from tedious coding. It is also possible to choose generation of the methods' bodies (flag *Generate HTML*).



Checking the flag *Generate HTML pages* adds code to the request-processing methods (service, doGet, doPost, doPut, doDelete), that returns an empty html page. If a necessary method is not selected when creating a servlet by pattern, it can be added later.

Debugging a Servlet

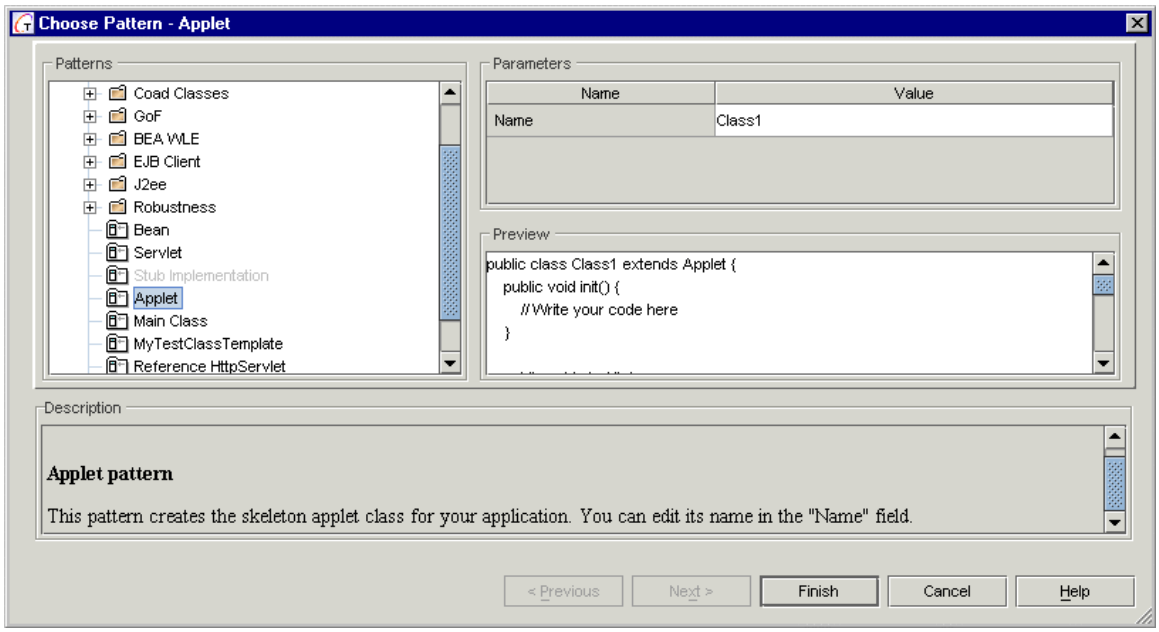
Jakarta Tomcat, the Reference Implementation for the Java Servlet 2.2 Technology, is now used to debug servelts. This is how it's done:

1. Make sure that `%TG_HOME%\bundled\tomcat\lib\servlet.jar` is added to the Search/Classpath of your project.
2. Select a servlet for debugging on the Class diagram and set breakpoints as required.
3. Choose Run Configuration command on the Tools menu, select Servlet/JSP tab and enter the name of the servlet, or choose one using Browse button.

4. Choose Tools | Run in Debugger and observe results in the Debugger pane, and in the browser window.

Creating, Developing and Debugging Applets

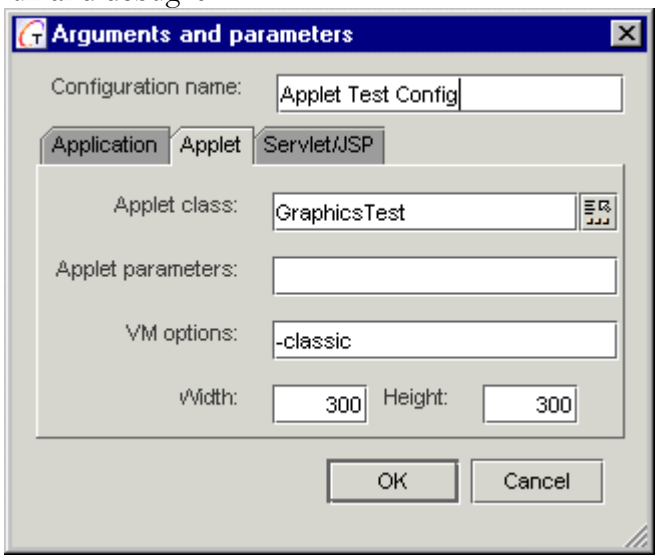
To create an applet, use Class by Pattern command from the diagram toolbar or speedmenu. This invokes Choose pattern dialog, with Applet pattern, suggesting default class name and standard set of methods.



Enter applet class name and press Finish button to complete. Further you can edit the source code in the editor to add the required functionality.

Having created an applet, you may try to run and debug it.

Debugging an Applet



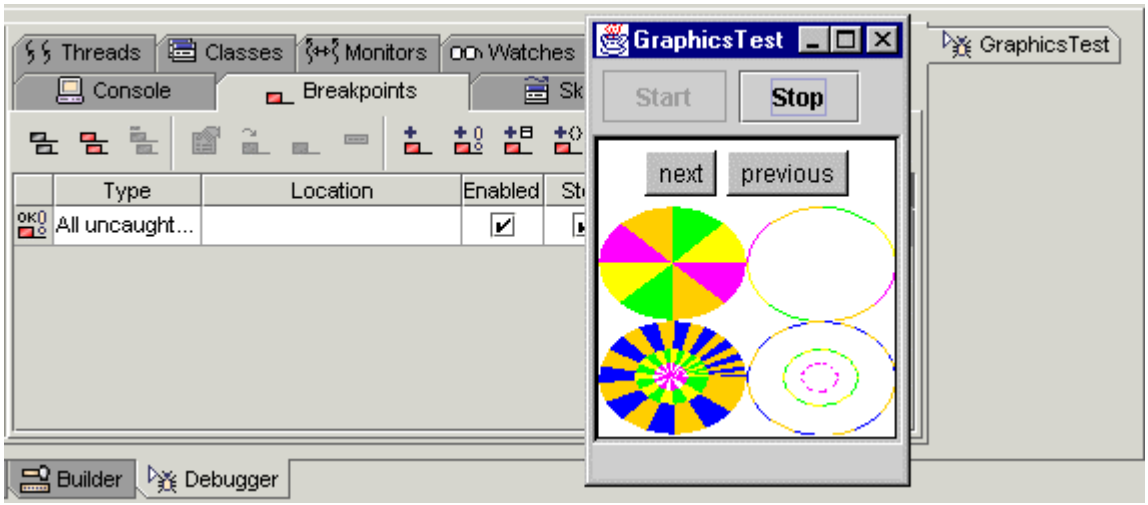
Together provides a simple way to run and debug applets.

Create a project for an applet: Select New Project on the File menu. In the Advanced mode, click *Remove* button in the *Project Paths* tab to delete the default project path.

Next, specify the required path for your project. Press 'Add Path' button and select the required path (for example, `c:\jdk1.2.2\demo\applets\GraphicsTest`). When ready, ht OK to complete and close the New Project dialog.

Now you can debug your applet. Select Run/Debug command on the Tools menu. The dialog contains "Applet" tab, where you must specify the name of the main class, parameters passed to the applet and VM options (if any), and the dimensions of the applet frame.

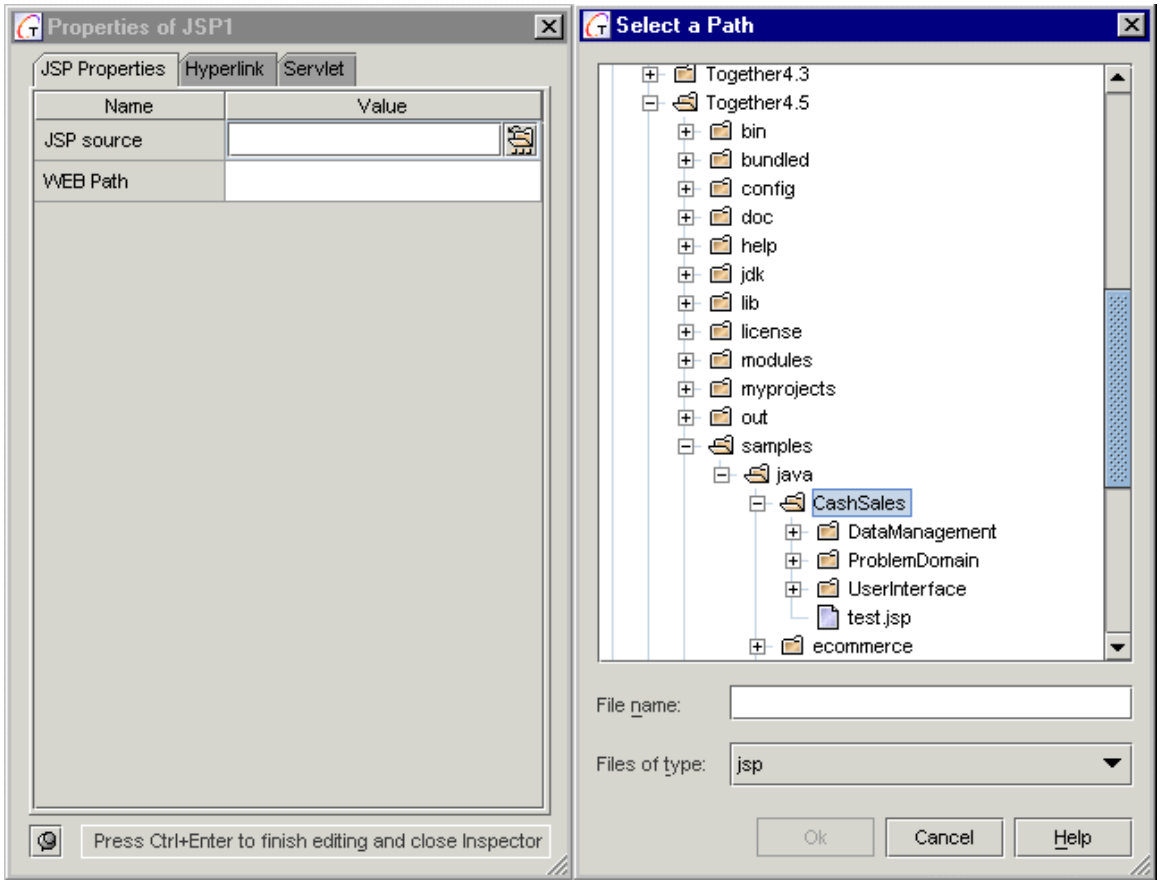
Parameters are entered in the format "name1"="value1" "name2"="value2" etc. Hit OK when ready. The Debugger (or Runner) pane shows up, and the applet frame appears:



Creating, Developing and Debugging JSPs

Java Server Pages are created in the Web Application diagram using a toolbar icon or New | JSP command on the diagram speedmenu. Having created a visual component, observe its properties in the inspector.

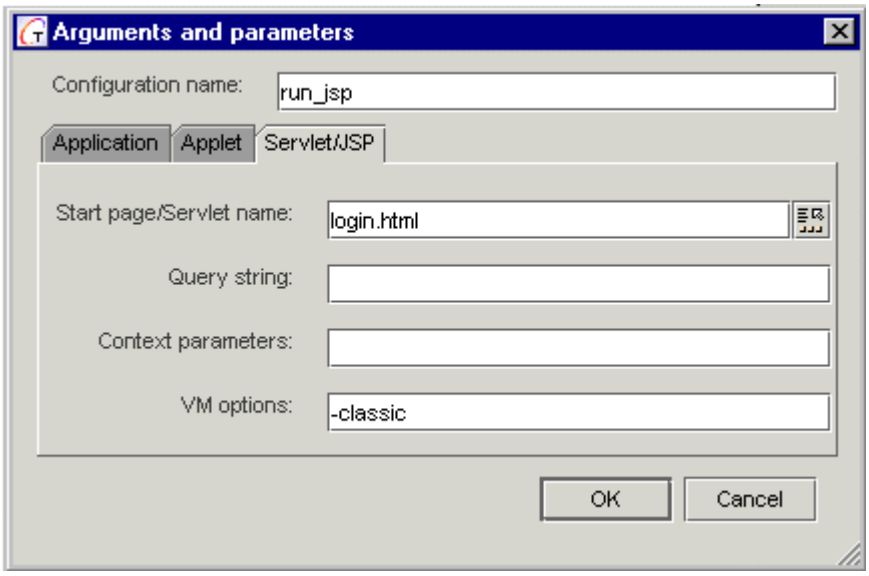
Press File Chooser button in the *JSP Source field*, and select the desired *.jsp file that will be loaded into the created visual component.



Having created a JSP, you can edit it as required, add necessary code, run and debug. To learn about editing JSP files, refer to JSP and HTML Editor section of the User's Guide.

Debugging JSPs

Together provides various ways to debug JSPs. The most basic way to run a JSP in debug mode requires appropriate settings to be done in the Run Configuration dialog.



In theory, after that it is possible to run/debug a JSP. However, in most cases, JSPs are intended for data presentation only, and as such, they need to be invoked by a servlet or a html page. Hence, this type of debugging is mostly suitable for the servlets, rather than the JSPs.

Together provides JSP debugging using Jakarta Tomcat 3.2, the official Reference Implementation for the Java Servlet 2.2 and JavaServer Pages 1.1 Technologies.

One type of JSP debugging takes place when an EJB is deployed to a certain application server. The client can be created as a JSP that allows to invoke EJB methods. For IBM WebSphere 3.5, Weblogic 5.1 and Weblogic 6.0 it is possible to create a JSP client in the debug mode. If this option is selected, the page "Simple JSP client generation" provides the fields for starting the debugger session under Tomcat.

The most comprehensive way of debugging JSPs suggests integration with Web Application Diagram deployment and use of the generated files and configurations as a background for launching the debugger session under Tomcat.

This is how it's done... On the current Web Application diagram, which contains JSP visual components, select the desired JSP, open it in the Editor pane, and set breakpoints in the desired lines. After that, invoke J2EE Deployment Expert, choose Generic 1.1 server and select the following tasks on the first page of the Expert:

- Compile classes from the selected diagram
- Generate WAR Deployment Descriptor
- Run Web Application under Tomcat in debug mode

After entering all necessary fields, on the page "Run Web Application under Tomcat in debug mode" specify the root folder of the web application.

> **Notes:**
>
> If a JSP can address to an EJB, the libraries of the appropriate application server, where the client will search for INITIAL_CONTEXT_FACTORY class, should be added the project classpath. Besides that, JSP file should contain the lines for proper handling of InitialContext object.

It is vitally important to properly specify the root catalogue. The jsp file being debugged should be the actual root element. Otherwise, the debugger session will start, but with a wrong component.

Having successfully passed through all these obstacles, you can observe your JSP running in the Debugger. To try JSP debugging hands-on, refer to the sample provided in the documentation.

How To Debug JSPs in the Web Application Diagram

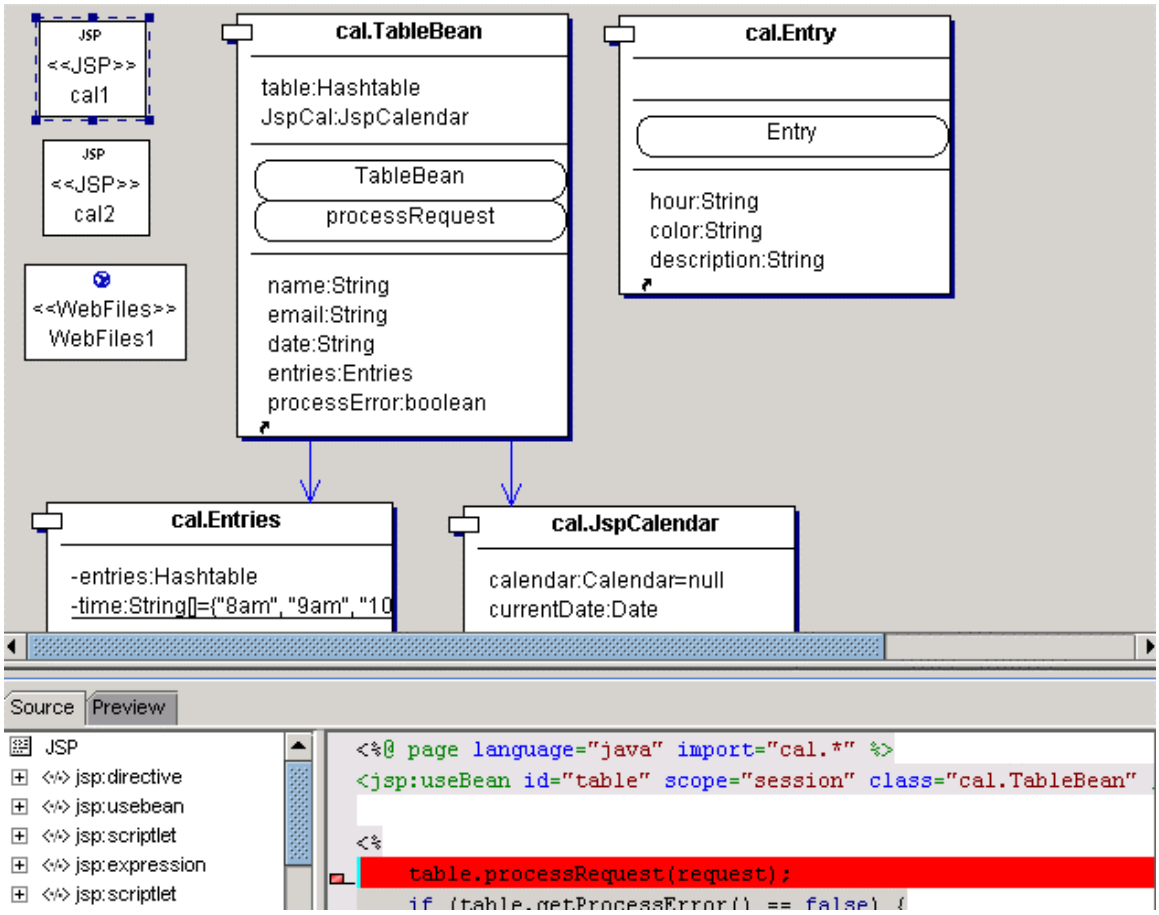
This example, supplied with the Tomcat server, demonstrates how to debug JSPs using the Web Application diagram.

Opening Project

Open the Project `%TG_HOME%\samples\java\ecommerce\jsp\cal\cal.tpr`, and choose the tab of Web Application diagram.

Select the visual component JSP1 and open in for editing. Make sure that full path to this component is specified in JSP source field of JSP properties.

Set breakpoint on the line 17 of the source code (table.processRequest).



How to set the Breakpoint

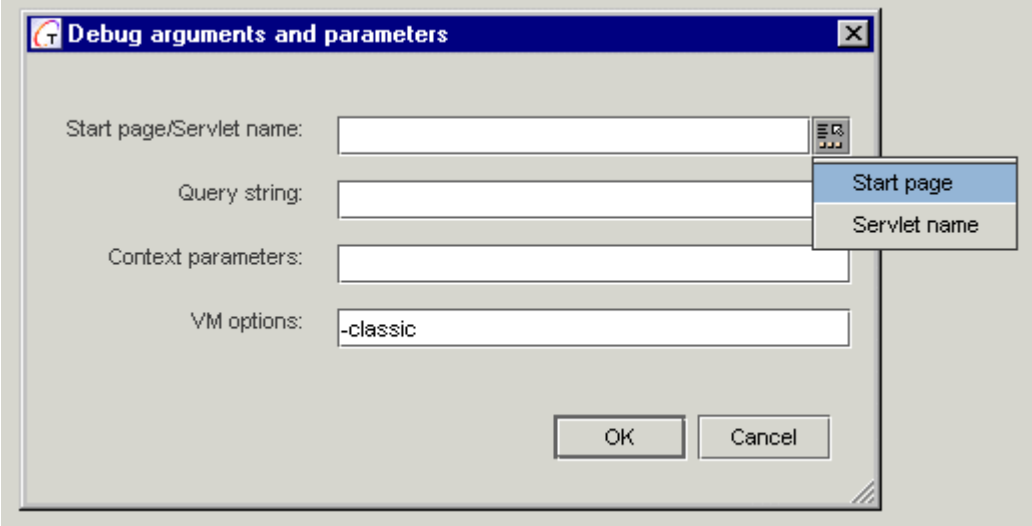
Debugging

Invoke J2EE Deployment Expert on the Tools menu and choose Generic 1.1 as the target server. Make sure that the **only** checked fields are *Compile classes from selected diagram, Generate Deployment Descriptor* and *Run Web Application under Tomcat in debug mode.*

On the *Common Properties* page, specify correct paths to JDK 1.2.2 and J2EESDK root directory (e.g. `c:\jdk1.2.2,c:\j2sdkee1.2.1`). Click *Next* to proceed.

On the next page, set *Root directory for Web Application which is deployed* to `%TG_HOME%\samples\java\ecommerce\jsp\cal\jsp`, and click *Finish.*

Debug arguments and parameters dialog shows up:



Press selection button, and choose *Start page* on the menu. This displays Select Start Page treeview, where you have to choose `%TG_HOME%\samples\java\ecommerce\jsp\cal\jsp\login.html`. Click *OK* to continue.

Executing process stops at your Breakpoint (line 17 of the code). Now you can proceed running your application Step-by-step with F8 key.

See also

Web Application diagram
JSP and HTML Editor

Developing EJBs

Together provides several productivity enhancing features especially for developers of distributed applications using Enterprise JavaBeans ™ (EJB). For information about which products provide EJB support, visit **www.togethersoft.com** or contact your Together Distributor. This chapter describes how to use Together to create, develop, and deploy EJBs.

It's beyond the scope of this documentation to teach EJB fundamentals. Documentation assumes you are familiar with the concepts, terminology, and Java Enterprise APIs, and that you have some experience developing EJBs. If you're just getting started with EJBs, a good resource is *Java Enterprise in a Nutshell* (2nd edition) by David Flanagan *et al,* 1999 O'Reilly & Associates. ISBN 1-56592-483-5.

Overview of EJB features

Together acts as a "control center" for EJB development: with it you can model, implement, compile, debug, document, and deploy your EJBs to an app server. Specifically, you can:

Create Together projects around existing EJBs. Together automatically reverse engineers the code and generates a visual model from which you can easily generate up-to-date documentation. Code and model remain synchronized at all times.

Create new Session or Entity EJBs with a single click. Together generates a default skeleton that includes both the visual model and the basic source code, which serve as the basis for further modeling and development. The skeleton includes home and remote interfaces as well as implementation class and primary key class (for entity beans).

Create multiple implementations. You can share Home and Remote interfaces among two or more EJB implementation classes enabling you to deploy different implementations of the same interfaces on different servers.

Compile your EJBs and generate XML deployment descriptors. Create SDK 1.2 DTDs, either generic, or platform-specific for supported leading application servers. Handy "expert" GUI simplifies the process.

Generation a JSP test client. During the deployment process you can optionally choose to generate a simple JSP (Java Server Pages) that you can use to test a running EJB directly in the server environment.

"Hot deploy" you EJBs from the Together environment directly to supported application servers.

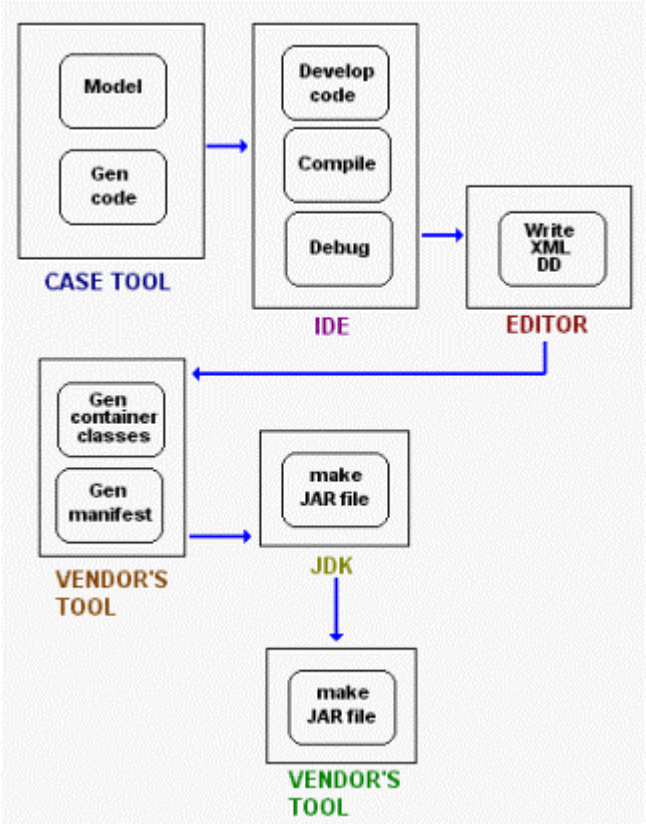
Create EJB Assembler, Web Application and Enterprise Application diagrams for your remote applications showing what components are required, EJB container handling, security roles and profiles, etc.

How Together simplifies EJB development

If you have developed EJBs before, then you know that the typical development process is something like this:

- Model EJB in some CASE tool.
- Generate source code framework with CASE tool.
- Develop the EJB code using an IDE or code editor.
- Compile the EJB classes using IDE or JDK.
- Debug any errors with some debugger.

- Write deployment descriptor in XML (typically with a text editor) that describes serialization and other properties of the EJB in the context of an application.
- Generate container-specific classes using tools from the application server vendor.
- Package everything into a JAR file using JDK utility or other tool.
- Deploy the EJB to the app server using tools from the application server vendor.



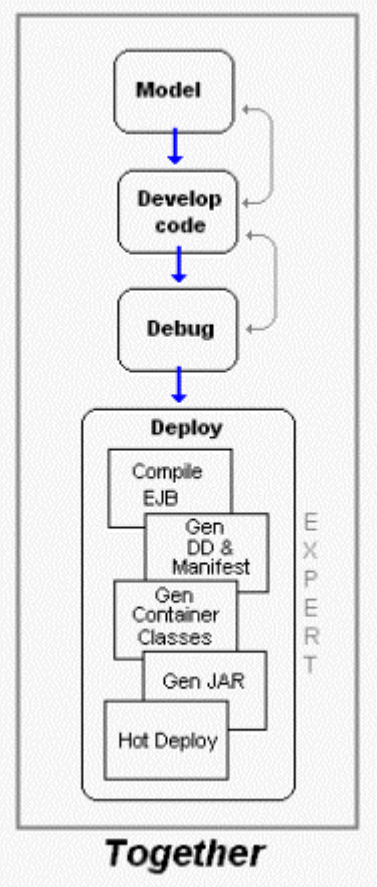
Typical EJB development process & tools

As you can see, the typical process involves a great deal of work that *has nothing to do with actually developing business solutions*. It adds the learning-curve overhead of a daunting array of different tools that only results in slowing down your distributed application development.

Together's approach to EJB development

Together's EJB development features are designed to speed up the process, get your distributed systems working sooner, and let you to focus on good design and development and robust functionality.

A major benefit is that *you don't have to write deployment descriptors*. Together stores bean properties right in the model. You can put bean-specific info into a special diagram type: the EJB assembler diagram, and at deployment time Together packages up everything and deploys to the server. With Together, the EJB development process looks like this:



- Model, code, and debug your EJB with Together's modeling and development tools. The source code framework stays in sync with visual model at all times.
- Run Together's J2EE Deployment Expert.

The J2EE Deployment Expert is a time saver that automates many of the tedious post-development chores into a simple set of steps. The expert will:

- Compile your EJB classes
- Generate container-specific classes for any supported app server
- Generate an XML deployment descriptor and manifest files for the selected server platform
- Generate a JSP client that you can use to live test a deployed EJB
- Package everything up into a JAR file written to any location you specify
- Connect to the app server and deploy.

Team Support is built right in

Together's approach fits seamlessly with team-based development. Because all bean information is stored with the right granularity, each EJB contains its own information, so that you can develop EJB's independently of one another. In the traditional approach there is one monolithic file - the "dreaded DD" - and all developers have to work with it. It has to be checked-in/out as a whole.

Configuring Together for EJB development

There are a number of configuration options that you may want to modify depending on where you are in the development process. You may find you want to set options differently depending on whether you are designing, implementing, or deploying an EJB.

Recognize JavaBeans

The Boolean config option *Recognize JavaBeans* controls whether or not Together recognizes JavaBean classes. This options is set to *True* by default, meaning that Together treats classes having methods that begin with *get* or *set* as JavaBeans. For EJB development, you should set this option to *True*.

The main menu command Options | Recognize JavaBeans toggles this option. There is a parallel icon on the Main Toolbar. The setting applies at the *Default* configuration level. (For information on configuration levels, see User's Guide: Configuring Together: Multi-level Configuration.

Diagram detail level

Depending on whether you are in the modeling/design or implementation stage, you may want to change the view management option *Diagram Detail Level*. The levels are *Analysis, Design,* and *Implementation*. Each shows a greater level of detail in the class icons in Class diagrams.

To set this options, choose Options | [*level*] - View Management - Diagram Detail Level (where *level* is the menu command for the desired configuration level).

(For information on configuration levels, see User's Guide: Configuring Together: Multi-level Configuration.)

EJB filtering options

You can set view management options that show or hide the following elements in the Class diagrams that contain EJB classes/interfaces:

- EJB Home interfaces
- EJB Remote interfaces
- EJB Implementation classes
- EJB Primary Key classes

By default, *only EJB classes* are shown in Class diagrams. Any elements that are hidden by these settings in diagrams still show up in the Model tab of the Explorer, so you can always tell that they exist in your model and your code.

To modify EJB filtering options:

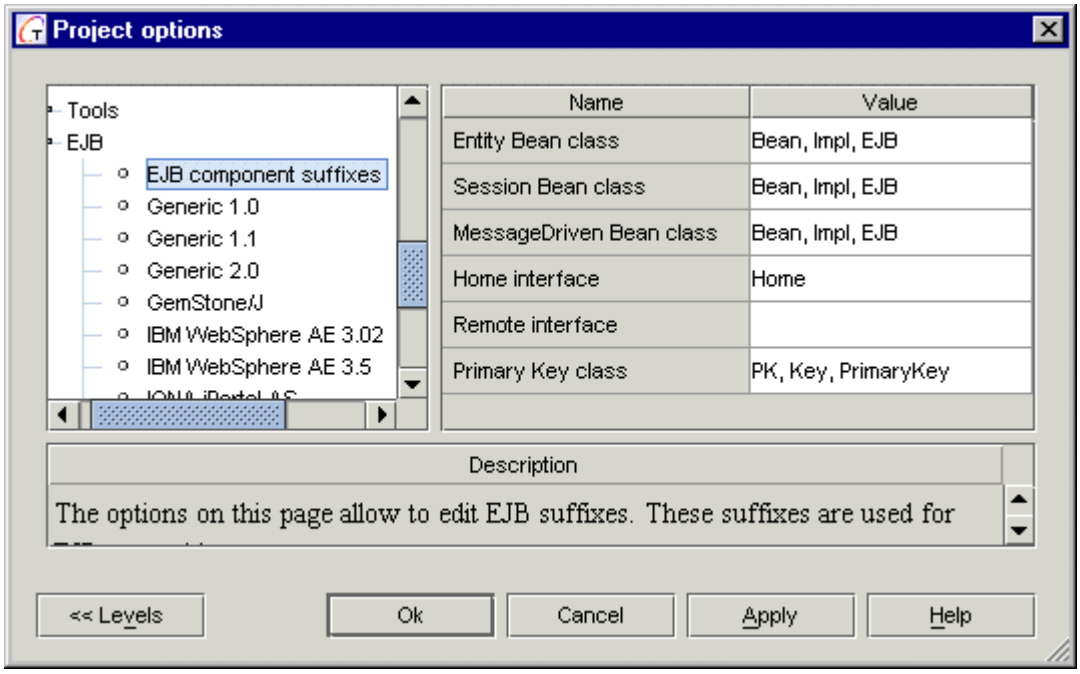
1. From the Main menu choose Options | [*level*] - View Management (where *level* is the menu command for the desired configuration level).
2. Expand the *View Management* node of the *Options* dialog, and choose *Show*.
3. The right pane of the *Options* dialog displays the list of EJB show options. Check or clear them as desired. Checking a box means the element should be shown in diagrams... clearing the box means the element should be hidden in diagrams.

Other filtering options

While you have the *Show* options open, review the other options that apply to classes in general to make sure that you can see what you want to see in your Class diagrams for EJBs.

Configuring suffixes

In the *EJB* node of the *Options* dialog you can specify the suffixes for various types of EJB classes. The suffixes help Together to properly recognize EJB's. If a project is created outside Together, class names may not follow the Together conventions. You can enter specific suffixes for your EJB classes:



Other configuration options

Before using Together for EJB development, you may want to review your entire configuration, especially View Management's Show Beans options for Java (Options | [*level*] - View Management).

If your license supports Together's advanced Editor features, and you will use this editor for EJB development, you may want to configure the Editor as well. (For more information, see User's Guide: Using the Editor.)

See also

Deploying Enterprise JavaBeans
EJB Assembler diagrams
Web Application diagram
Enterprise Application diagram

Creating EJBs in Together Projects

You create and develop EJBs in the context of a Together project. There are two ways you can create EJBs in a project:

1. Create a Together project for existing EJB source code.
2. Create a basic model and code skeleton for a new EJB using the "one-click" EJB feature in Class diagrams. You then specify EJB properties and add EJB fields, methods, etc. in the EJB Inspector.

This section explains how to use each of these techniques. See also Overview of EJB Features.

Creating a project using existing EJB code

If you have existing EJB source code, you can create one or more Together projects around it. Together creates visual diagrams when it reverse engineers the code, and then keeps the visual model synchronized with subsequent changes to the code, and vice versa. Once the code is part of a Together project, you can quickly and easily generate up-to-date documentation.

Creating a project from existing EJB code is no different from creating any other kind of Java project. See:

User's Guide: Creating a Project from Existing Source Code
Creating and opening a project
Project basics

Before you create a project for existing EJBs, read Developing and deploying EJBs: Configuring Together for EJB development.

Creating "one-click" EJBs

When your Together license enables EJB support, three EJB icons display in the Class diagram toolbar:

Entity bean

Creates elements in the visual model, and generates the underlying source code, for a default implementation of a persistent *entity EJB* with skeleton declarations for:

EJB implementation class, with:

- Entity context attribute
- One default field (integer type)
- Default set of method declarations:

  * set & unset context methods
  * Activate & Passivate methods
  * Remove method
  * Store & Load methods
  * Create & PostCreate methods
  * findByPrimaryKey method
  * getField & setField methods

EJB Home interface, with:

- Create method signature
- findByPrimaryKey method signature

EJB Remote interface

EJB Primary Key class, with:

- default field (integer type)
- primary key, hash code, and equals methods

Dependency links

Session bean

Creates elements in the visual model, and generates the underlying source code, for a default implementation of a nonpersistent *session EJB* with skeleton declarations for:

EJB implementation class, with:

- Session context attribute
- setSessionContext method
- Activate & Passivate methods
- Remove method
- ejbCreate method

EJB Home Interface
EJB Remote interface

MessageDriven bean

Creates elements in the visual model, and generates the underlying source code, for a default implementation of a *MessageDriven* bean with skeleton declarations for:

EJB implementation class, with:

- MessageDriven bean context attribute
- setMessageDrivenContext method
- ejbCreate method
- ejbRemove
- ejbActivate and ejbPassivate methods
- onMessage method

Note that MessageDriven beans are only used with 2.0 specification.

No properties or Business Methods are declared. There are several ways to add them:

- Add them visually in the diagram using the implementation class's *New* speedmenu.
- Add them visually using the EJB Inspector (speedmenu | Properties)
- Write the declarations in source code in the Editor pane when the implementation class icon is selected.

Home and Remote interfaces are hidden in the diagram by default, but you can see them in the Explorer (see Configuring for EJBs for more information). The interface names are automatically kept in sync with the name of the implementation class.

To create a "one-click" EJB:

1. Create or open a Together project and create or navigate to the desired package.
2. Create or open a Class diagram in the desired package.
3. On the Class diagram toolbar, click the *Entity EJB* icon  , *Session EJB* icon  or *MessageDriven* icon  to create appropriate type.
4. Click the diagram background to generate the EJB elements as described above.

The resulting implementation icon displays the name of the created bean and its type, compartments for the attributes, operations, home and remote interfaces, and primary key class. Linked interfaces are listed under the bean name.

Use View Management's Show options to populate the diagrams with the necessary elements only. You can opt to show or hide implementation classes, home/remote interfaces and the primary key class. However, it is still possible to view the source code of the hidden elements in the Editor pane: all elements, though hidden in the Diagram pane, are displayed in the Explorer and can be opened for editing.

Customizing the default code for EJBs

The default code generated by one-click EJBs should be an adequate starting point for many developers. However, the default code *is* customizable by modifying the appropriate Code Template. You can customize the templates for Entity Bean class, Session Bean class, MessageDriven bean class, PrimaryKey class, Home Interface, and Remote Interface. For information on modifying these templates, see Using Code Templates.

Configuring EJBs using EJB Inspectors

Once you have a skeleton created by the on-click feature, you can use the EJB Inspector to edit its properties.

To use the EJB Inspector:

1. Select the EJB in the diagram or the Explorer.
2. Press *Alt - Enter* or choose *Properties* from the speedmenu to display the EJB Inspector.

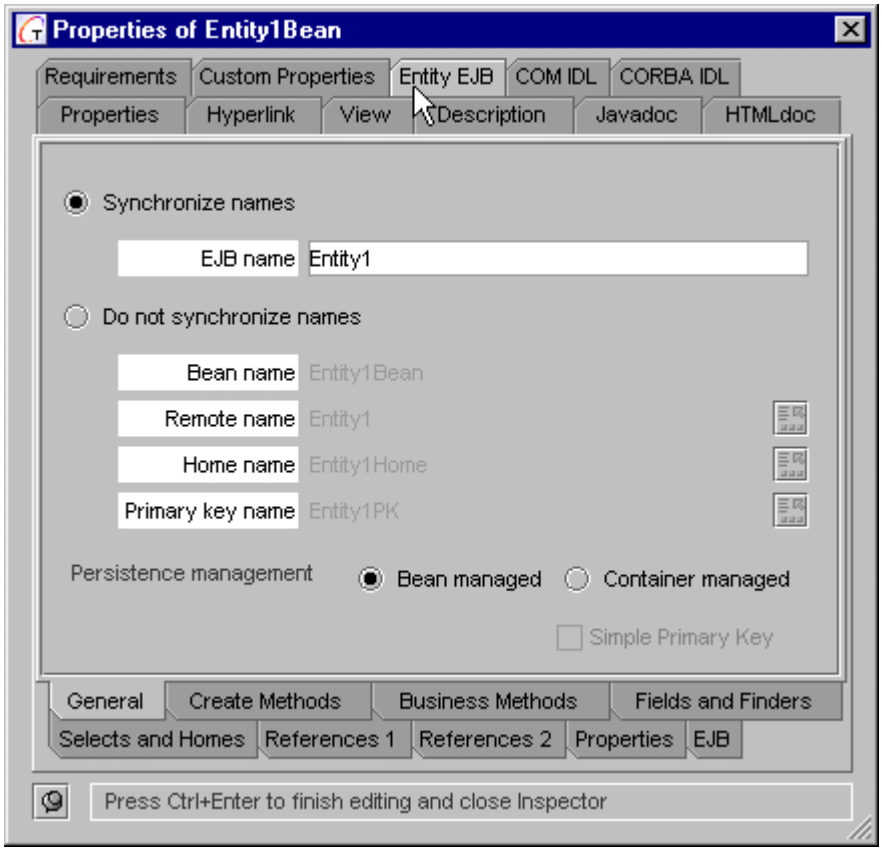
Using the EJB inspectors

You can develop EJBs visually using the EJB Inspector for bean classes (speedmenu | Properties). The Inspector has pages common to all classes, but adds some specifically for working with EJBs.

- For Entity EJBs, pay attention to the *Entity EJB* page.
- For Session EJBs, pay attention to the *Session EJB* page.
- For MessageDriven bean, pay attention to the *MessageDriven* page

The type-specific pages display a lower tabset that provide the means of specifying general properties, adding and removing business methods, defining references, and specifying bean type specific properties. Inspector for Entity EJB also has pages for defining Create and Finder methods.

Entity EJB Inspector



Primary keys

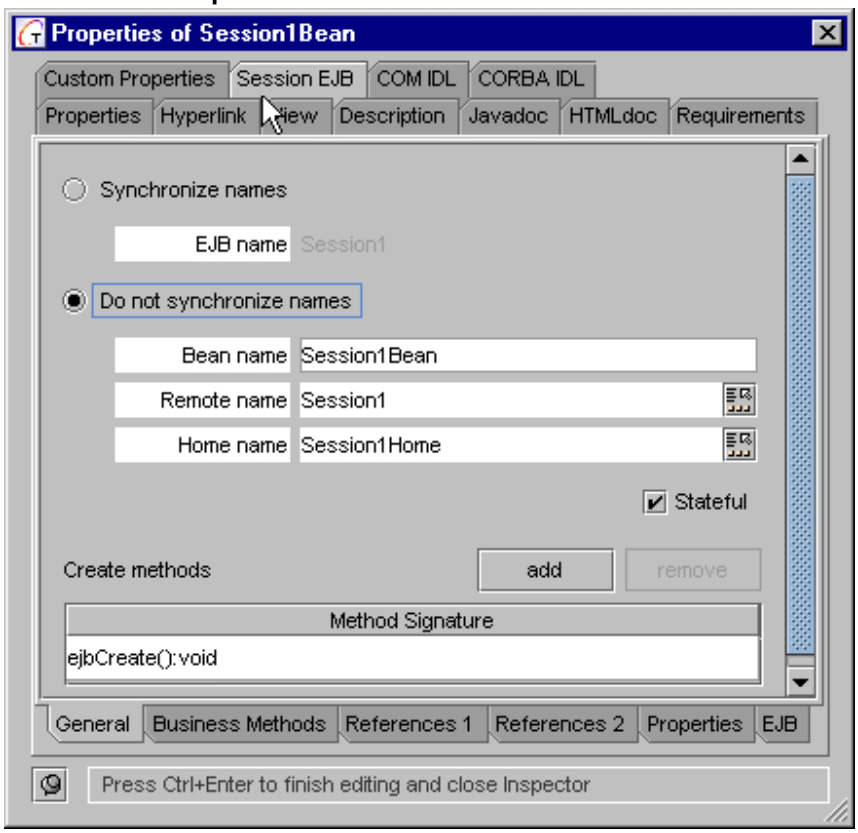
There are two ways to specify Primary key class for an EntityBean with container managed persistence:

1. Primary key maps to a single field of the entity bean class
2. Primary key maps to multiple fields of the entity bean class

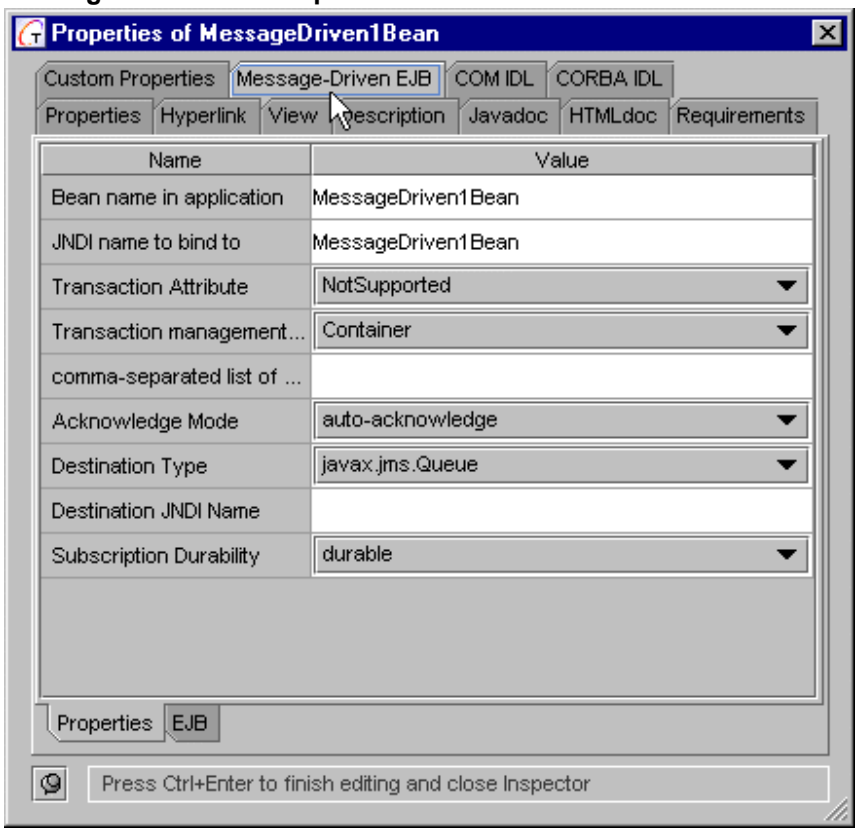
To create a primary key of the first type, set the flag *Simple Primary Key*, which is enabled for Container managed beans. This helps avoid wrapping of the simple type primary key (e.g. String) into a user-defined class.

The second method is helpful for implementation of compound keys.

Session EJB Inspector



Message-Driven EJB Inspector



Sharing Home/Remote interfaces

You can optionally share the same set of home and remote interfaces between two or more EJB implementation classes of the same type (Session or Entity). You may want such capability in cases where you need to deploy differing implementations of the same interfaces on different servers.

Creating a second implementation class

In a case of shared interfaces, you should develop one complete implementation with home and remote interfaces. You can then create a second implementation class, refactor it into an EJB implementation, and specify the home and remote interfaces from the first bean in the properties of the new one.

To create a second implementation class:

1. Create or open a Class diagram to show the new class.
2. Create a new class in the diagram and name it as desired.

To refactor the class to an EJB implementation:

1. Right-click and choose *Choose Pattern* from the speedmenu to display the Choose Pattern dialog.
2. Locate the *EJB Implementation* folder and expand it.
3. Select the *Session EJB implementation, Entity EJB Implementation* or *MessageDriven EJB Implementation* pattern and click *Finish*.

You can now begin developing the implementation, or you can proceed to specify the home and remote interfaces for the new bean class using the home/remote interfaces from the first implementation.

To reuse the first implementations home/remote interfaces:

1. In the properties Inspector for the new class, select the *xxxEJB* page (where xxx is either *Session* or *Entity* depending on the type of EJB).
2. Choose the *Do not synchronize names* option button.
3. In the *Remote Name* field, click the browse button to display the Select Element dialog.
4. Use the Model node of the treeview to locate and select the remote interface belonging to the first implementation... that is, the remote interface that you want the bean you are working on to share.
5. Click OK to accept the selection. Respond *Yes* to the *Change Class?* prompt.
6. Do the same thing in the *Home Name* field, selecting the home interface for the first implementation in the Select Element dialog.

Showing second implementation class in separate Class diagram

If the second implementation class in the same Class diagram as the first, Implementation links are automatically drawn from the interfaces to the second implementation class. If the two implementations are in different Class diagrams, you may want to show the second implementation class in the Class diagram for the first implementation.

To show a linked second implementation class:

1. Select the remote interface for the first implementation class.
2. Choose Add Linked from the interface speedmenu. Linked elements are shown in the Add Linked tab of the Message pane.

3. Select the second implementation class in the Message pane, right-click and choose Add.

A Class icon for the second implementation class is added to the diagram. You can now work on the second implementation class from either Class diagram.

Deleting implementation classes with shared interfaces

Normally, if you delete an EJB implementation class, home and remote interfaces are deleted along with the implementation class, both in source code and in the diagram. In cases where interfaces have been shared with another implementation class, deleting elements from the diagram does not automatically result in deletion of the relevant source code files... only the visual diagram elements are deleted.

You must explicitly delete implementation classes, and home/remote interfaces by selecting them in the Model tab of the Explorer and choosing Delete from the speedmenu.

Note: you can find an animated demoguide at **www.togethercommunity.com**.

See also

Drawing diagram elements
Working with patterns

Verification and Correction of EJB's

Generated EJB's must comply with certain requirements, which include general conformance with all EJB specifications and specific sine qua nons of EJB 1.0, 1.1 and 2.0. Describing the interfaces to be implemented or methods to be overridden are beyond the scope of this manual. Refer to the documentation at **www.sun.com**.

EJB's are verified according to the selected specification (EJB 1.0, 1.1 or 2.0). Subsequently, it is possible to correct the encountered errors.

To make sure that certain methods or properties exist in an EJB, or to obtain their values, it's necessary to scan the entire hierarchy of objects that comprise the EJB. The encountered members are added to a Hashtable.

In order to support the OOP approach, the users have a good chance to develop classes for customized verification and correction, using Open API.

Syntactic rules for verification are defined in the configuration file under `%TOGETHER_HOME%\modules\togethersoft\modules\ejb`, in the folder for the used EJB specification. It makes possible to easily adopt the ever changing requirements.

You can create customized syntactic rules by editing appropriate config files for the entity or session beans. However, this bestows on you the full responsibility for the results of such exercises.

Using verification and correction

To make use of this feature, select *Verify EJB* command on the bean's speedmenu. The encountered errors are corrected if the option *Correct after verification* is selected in the Options | EJB | Verification. The process adds Verification tab to the Message pane. Two kinds of messages display the results: the messages of the first type inform about the encountered errors, and the messages of the second type inform that the errors were fixed.

Deploying Enterprise JavaBeans

This section provides an introductory overview of Together's EJB deployment support features. Note that deployment support is not available in all Together products. Please visit **www.togethersoft.com** for product information.

Overview

In Together products that come with EJB deployment support, you can use Together both to develop EJBs, and also to *deploy* them to a supported application server. Together compiles your EJBs, generates XML deployment descriptors (including manifest), generates container classes, packages the EJB into a JAR file, and deploys the result to a location you specify. On server platforms that support it, you can direct Together to "hot" deploy directly to the app server. Also, for some servers, you can optionally generate a simple JSP (Java Server Pages) client which you can use to test the deployed EJB running on the application server.

Using Together, it is now possible to deploy complicated applications that contain multiple EJBs, servlets, JSPs and other web files. Together provides tools for visual assembling of the distributed applications for deployment.

Supported application servers and proxies

TogetherSoft directly develops deployment support for major application servers, and is working with vendors to rapidly develop deployment support for more and more server platforms all the time.

To see a list of the currently supported servers:

1. Open a project.
2. On the *Main menu*, choose *Tools | J2EE Deployment Expert*.
3. Activate the drop-down list of application servers to see the server platforms currently integrated.

Note: The **J2EE Deployment Expert** is available for one of the following diagrams: Class diagram, Enterprise Application diagram, EJB Assembler diagram, Web Application diagram.

Building Block Proxies

Some of the listed app servers may be *proxies*... that is, deployment support is available for the listed server, but the vendor's Building Blocks for Together deployment support do not ship with Together, and are not installed.

When you choose one of these proxy items in the server list, Together activates your Web browser and takes you to the URL on the server vendor's website to download and install the vendor's deployment support Building Block(s) for Together.

Generic server options

You can also choose the 'Generic" server options. Use these if you just want to generate generic EJB 1.0, EJB 1.1, EJB 2.0 compatible deployment descriptors.

Visual Assembling and Deployment Tools

You can create EJB Assembler diagram in Together to map the assembly of various EJB components into a distributed application. Assembler information is always in-sync with provider information - no redesigning of Assembler deployment descriptors when EJBs are changed.

Web Application diagram is used to collect all servlets, JSPs and other web files used in your application into a single archive file.

If your application contains multiple EJB Assembler diagrams and Web Application diagrams, you can create Enterprise application diagram to map your existing modules.

Requirements for deployment

In order to deploy your EJBs directly from Together, you need the following:

- An installation of Together with full EJB development and deployment support.
- An accessible installation of the target application server with appropriate access rights

Note: You can deploy your J2EE application remotely to the selected Application Server, using *Together*'s plugins.

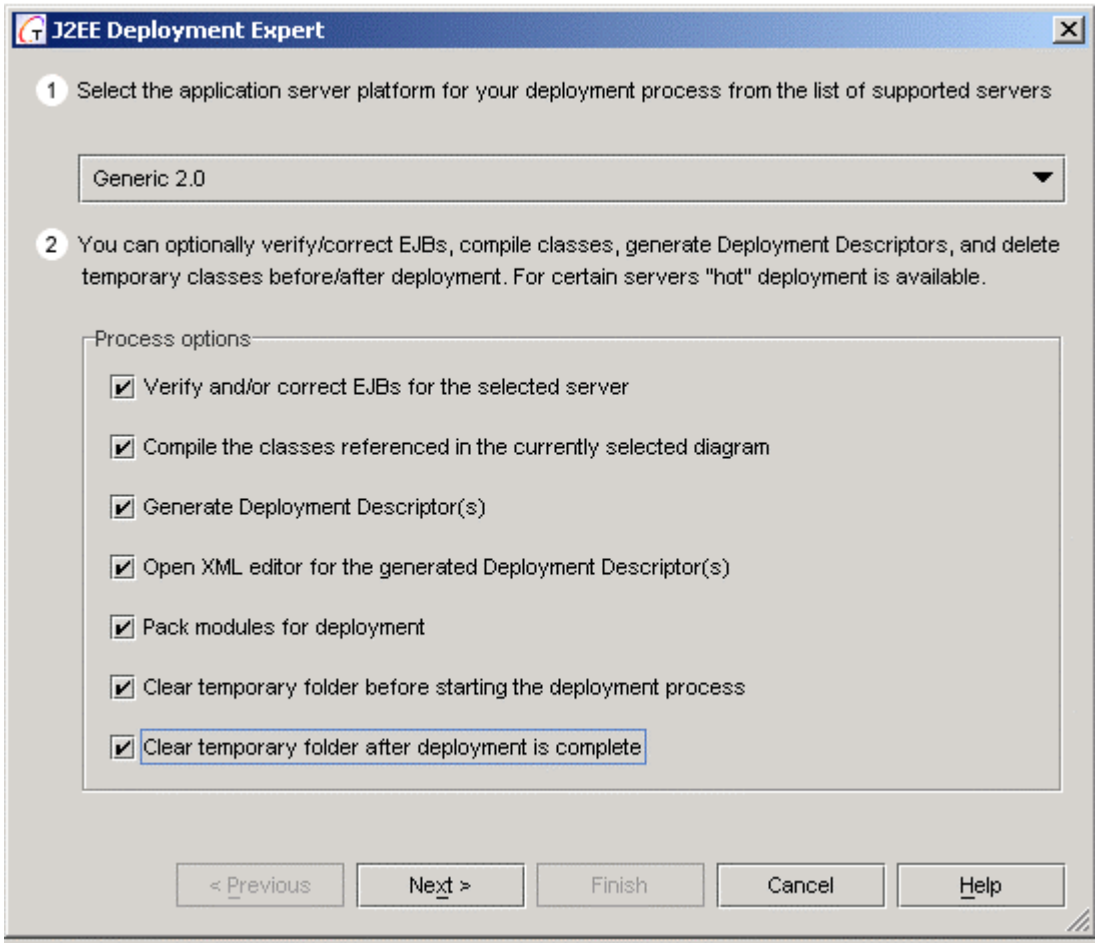
- An accessible installation of Java Enterprise Edition 1.2 or higher for compiling EJB classes and interfaces
- An accessible installation of Java 2 SDK (JDK 1.2 or higher) for generating JAR file
- An accessible temp directory and sufficient disk space for temporary files generated by the *J2EE Deployment Expert*
- An http-accessible location on the server for the generated JSPs (if generating a JSP test client).

Note: Before deployment Together checks whether old jar file exists, and tries to delete it. If deleting is not possible, the J2EE Deployment Expert doesn't start, and an error message displays. This note is not actual for Weblogic 6.0.

Using the J2EE Deployment Expert

Together provides the J2EE Deployment Expert... a convenient GUI that greatly simplifies J2EE deployment process. In this dialog, you can specify:

- the target application server platform,
- what deployment-related actions you want to take place (compile, etc.),
- 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 deployed EJB.



According to your option selections, Together can handle interaction with the compiler to compile EJB classes, generate the XML deployment descriptors (including manifest) for the target server, update the Bean's EJB specification conformity, generate container classes, then package everything into a JAR (WAR, or EAR) file, and deploy the JAR to a location you specify. On server platforms that support it, you can specify "hot" deployment directly to the application server.

There are two scenarios for using the J2EE Deployment Expert:

For "fast track" deployment with default values for security and permissions, or for deployment prototyping, run the J2EE Deployment Expert from an appropriate *Class diagram*.

If you need "full featured" assembly information with control over security and permissions, run the *Expert* from an EJB Assembler diagram, Web Application diagram, or Enterprise Application diagram.

To run the J2EE Deployment Expert:

1. Open the project and the diagram containing the elements to be deployed (Class, EJB Assembler, Web Application, or Enterprise Application diagrams).
2. On the *Main menu*, choose *Tools | J2EE Deployment Expert* to launch the expert dialog.
3. Choose the target server platform and set the other options as desired. For example, if the classes need to be compiled, check the *Compile Classes* option (if they are already compiled, clear this option).

Note: All server platforms represented in the list of plugins are supported for Classes and EJB Assembler diagrams.

Web Application diagram and Enterprise Application diagram support WAS 3.02/ WAS 3.5, WLS 5.1/6.0, Generic 1.1, Generic 2.0, WebLogic 6.0, iplanet 6.0, but do not support Generic 1.0.

4. If desired, check the option to generate a JSP client. (Note that this checkbox does not appear for all servers. For more information, see Server-specific information below.)
5. Click *Next* to advance through the page sequence of the Expert.

Note that the number of pages and content varies according to the selected server platform. During the process you will specify such things as:

- the paths to the server, various server tools, and a temp directory.
- server host name, port number, and password (if you are going to "hot deploy" to the server).
- output location and base URL for JSPs (if generating a JSP test client)

Starting WebLogic from within Together

If you are going to deploy your EJB's to WebLogic 5.1, or 6.0, you can enjoy a handy possibility to launch the target application server from within Together. Presently, Together provides two plugins: *Start WebLogic Application Server 5.1* and *Start WebLogic Application Server 6.0*. Use drop down menu of the field *Select application Server platform...* of J2EE Deployment Expert to launch the required server.

Server-specific information and examples

The above procedure is intended as a basic guideline for running the expert. In order for the expert to succeed, there are some things you need to set up in your project and/or your environment. These are dependent upon the target application server. You can find some examples under *Server-Specific Examples:*

How to deploy CMP bean from WebLogic 5.1, 6.0 to WebSphere 3.5
How to deploy session bean from WebLogic 5.1, 6.0 to WebSphere 3.5
How to create a servlet and deploy it on WebLogic 5.1, 6.0
How to create a servlet and deploy it on WebShpere 3.5
How to create a simple JSP client to test a deployed Bean

See most complete deployment example of a sophisticated real-life application in the section EJB Step by Step.

See also

EJB Assembler diagrams
Enterprise Application diagram
Web Application diagram
J2EE Deployment Expert for WebLogic
J2EE Deployment Expert for WebSphere

Step by Step How To Create a One-Click EJB and a Client Creating a Session Bean

This section gives a brief step-by-step description of creating and EJB and a client. The further steps of deploying the bean, compiling and running the client are described in the *Sample Project for WebSphere 3.5*.

Creating an EJB

Create a New Project (*Main Menu | File | New Project*). Enter "HelloWorld" as the project name. As you won't need standard libraries, remove them from the project: click *Advanced* button, select *Search Classpath* and make sure that checkboxes *Include standard libraries* and *Include Classpath* are cleared. Click *OK* button.

Next, create a New Package. Set its name to "hello" and choose *Open in New Tab* from the speedmenu.

In this package, click on *Session EJB* icon and create a session bean, with the name *HelloBean*.

Create business method (*New | Business Method* on the bean speedmenu). Using in-place editing, change the default method name to `hello:String`. In the Editor pane, add the following code to this method:

```
public String hello(){ System.out.println("hello()"); return
"Hello World"; }
```

Note: Make sure that return types of `hello()` method in the "HelloBean" and in "Hello" are identical (String).

The bean is now ready for deployment to WebSphere 3.5.

Then you can deploy EJB , or first create a client.

Creating a client

By now you have a project with *Hello* Session bean, which is deployed to WebSphere 3.5.

Create a *New Package* on the default diagram and set its name to *client*. Open this package in a new tab (choose Open in New Tab on the package speedmenu).

Create main class in this package using Class by Pattern icon. Select *Main Class* pattern, change name to *HelloClient* and press Finish button.

Add the following code to your main method:

```
public static void main(String[] argv) {
   try{
      Properties props = System.getProperties();
       props.put(Context.INITIAL_CONTEXT_FACTORY,
       "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
       props.put(Context.PROVIDER_URL,
       "iiop://localhost:900"); 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(); }
    }
```

Add the following import statements:

```
import javax.naming.*;
import java.util.Properties;
import hello.*;
```

The client is now ready for compiling and running.

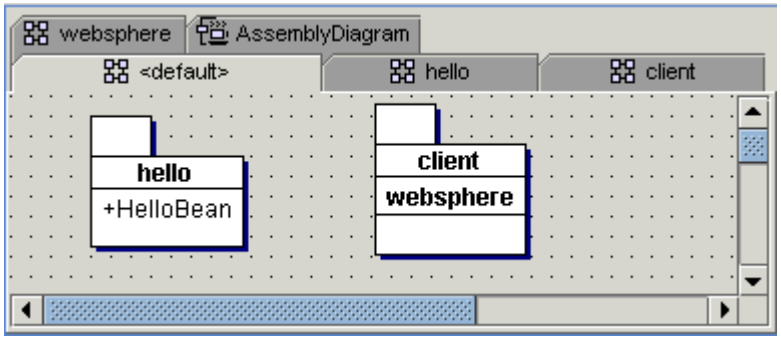
Then deploy EJB , if you did not deploy yet, compile and run the client.

How to Deploy the Bean to IBM WebSphere 3.5

This sample logically extends the previous topic in going step by step through deploying an EJB to IBM WebSphere 3.5, compiling and running the client.

Opening the Project

Open the sample project `HelloWorld.tpr` under `%TOGETHER_HOME%/Samples/java/ejb/WebSphere`.



Deploying an EJB

First, start WebSphere Server 3.5 services from *Control Panel* | *Administrative Tools* | *Services*. Next, in *Together*, invoke *J2EE Deployment Expert* on the *Tools* menu.

Select *IBM WebSphere AE 3.5* as the target server and make sure that all available checkboxes, except for *Generate start and compile file for the client, Generate simple JSP client, Generate command file for deployment* are set. Click *Next* to proceed.

Specify correct paths to WebSphere home directory and to IBM JDK 1.2.2 root directory (e.g. `c:/WebSphere/AppServer` and `c:/WebSphere/AppServer/jdk`), and click *Next*.

Set the following parameters on the next page:

Parameter name	Setting	Comment
Admin Node Name	Name of the Node to deploy	Name of the computer
Application Server Name	Your Application Server	"Default Server" by default
EJB Container Name	Your Container name	"Default Container" by default
Target WebSphere server directory	WebSphere home directory	e.g. `c:\WebSphere\AppServer`
Stop & Remove Beans(s), Servlet(s) before deployment	ON	
Start Beans(s), Servlet(s) after deployment	ON	
Launch server in debug mode	OFF	

Click *Next* and *Finish*, to complete you work with *J2EE deployment expert*. If you have checked *Hot Deploy* check box, the last message in the Message Pane should be:

```
//WAS35: Finished with 0 Errors, 0 Warnings.
```

It means that the deployment process successfully completed.

Compiling and Running the Client

Provide resources, required for compilation and run. Invoke Options dialog on the Project level. In the *Run/Debug* tab, specify path to jdk home as `%WAS_Home%\AppServer\jdk`.

In the *websphere* tab of the Diagram pane, select HelloClient class and choose *Tools | Rebuild* Node on its speedmenu. The Message pane displays a message that the tool is completed.

Select *Tools | Run/Debug | Run* from the main menu and see "Hello World" message in the Message pane and "hello()" in the server console.

Sample Project for BEA WebLogic Server

You can use the *J2EE Deployment Expert* to compile your EJBs and deploy them to BEA WebLogic Application Server 4.5.1, 5.1, and 6.0

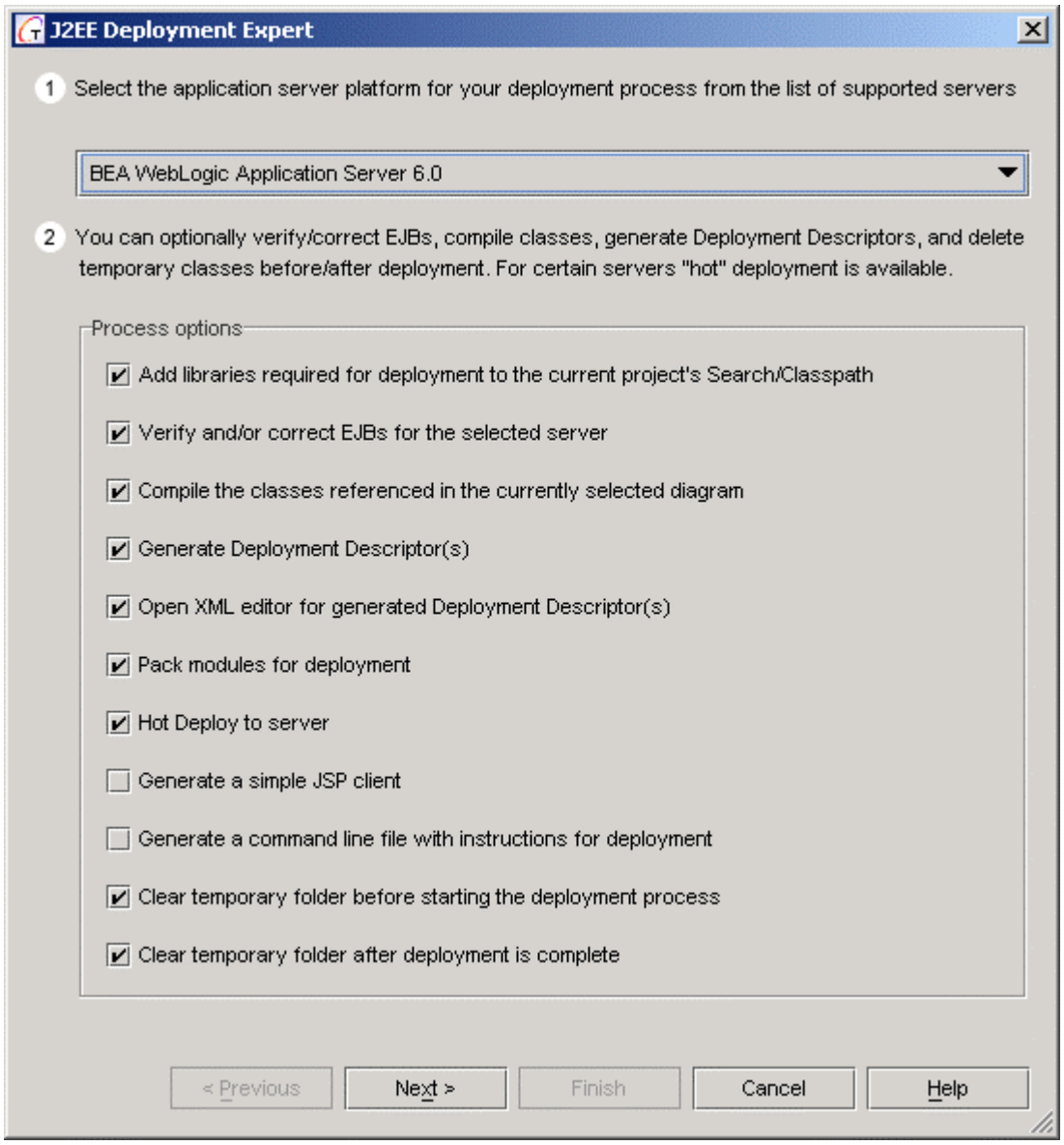
This section describes how to set up your Together project and your environment so that you can use Together to compile both EJB clients and deploy to EJBs to BEA WebLogic 5.1, 6.0.

Setting project properties and environment

To start BEA WebLogic Application Server: on the *Together's* main menu, choose *Tools | J2EE Deployment Expert*. Select *Start WebLogic Application Server 6.0* (or *Start BEA WebLogic Application Server 5.1*) as the target server from the dropdown list, if you want to use this server. On the second page of the J2EE Deployment Expert set the root directory of the WebLogic server using the *File/Path* Chooser button, and click *Finish* to complete.

Next, select the *EJB Assembler diagram* or *Class diagram* (with EJB on it) to make this diagram active, and choose *J2EE Deployment Expert* on the *Tools* menu. Select *WebLogic Application*

Server 6.0 (or *WebLogic Application Server 5.1*) as the target server. Make sure to set the following flags only (all the other flags should be unchecked):



If you do not want to deploy immediately, *Hot Deploy* should be unchecked.

Till this moment your actions were almost the same both for BEA WebLogic Application Server 6.0, and BEA WebLogic Application Server 5.1. Click *Next* to continue.

There is a difference between BEA WebLogic Application Server 6.0 and 5.1 now:

For BEA WebLogic Application Server 6.0

On the next page of the J2EE Deployment Expert, set paths to jdk1.3 (e.g. c:\jdk1.3), WebLogic (e.g. c:\bea\wlserver6.0), and paths to the resulting jar file and temporary files. Click *Next*. Note that it is impossible to use jdk1.2.

For BEA WebLogic Application Server 5.1

On the next page of the J2EE Deployment Expert, set paths to jdk1.2 or higher (e.g. c:\jdk1.2), WebLogic (e.g. c:\bea\wlserver5.1), to the directory for compiled servlets (e.g. c:\weblogic\myserver\servletclasses), and paths to the resulting jar file and temporary files.

Click *Finish,* if *Hot Deploy* checkbox was unchecked, i.e. WebLogic Server was not started.

You can choose *Search/Classpath* tab from *File | Project Properties* to see additional classpaths included automatically:

`%WL_HOME%\weblogic.jar` - for WebLogic 6.0

`%WL_HOME%\lib\weblogicaux.jar` - for WebLogic 5.1

`%WL_HOME%\classes`

`%JDK_HOME%\jre\lib\rt.jar`

Where: %WL_HOME% is the home directory of WebLogic Server (e.g. c:\weblogic), and %JDK_HOME% is the home directory of JDK (e.g. c:\jdk1.3 for BEA WebLogic Application Server 6.0 or c:\jdk1.2 for BEA WebLogic Application Server 5.1).

IMPORTANT: weblogicaux.jar should always be the first in the list, ahead of even standard libraries for **BEA WebLogic Application Server 5.1** .

To edit *Search/Classpath:*

1. On the Main menu choose *File | Project Properties* to display the *Project Properties* dialog.
2. Click the *Search/Classpath* tab.
3. Click the *Add Path or Archive* button to add a new path.
4. Click the *Remove* button to remove a path.

Deploying EJBs to BEA WebLogic Server

Click *Next,* if *Hot Deploy* checkbox was checked, i.e. WebLogic Server was started.

On the last page of the J2EE Deployment Expert, set server host (localhost), system password that you've set for WebLogic (e.g. together) and port (usually it is 7001), and click *Finish* to complete.

If the deployment process successfully completed, the *Message Pane* displays:

```
//WLS60: Finished with 0 Errors, 0 Warnings.//
```

Now we are ready to compile and run the client.

Deploying *Hello World* EJB sample to BEA WebLogic 6.0 Server

System requirements

This section shows you step-by-step how to compile an EJB, deploy it to the server, compile an EJB client in the same project, and run it with the deployed EJB.

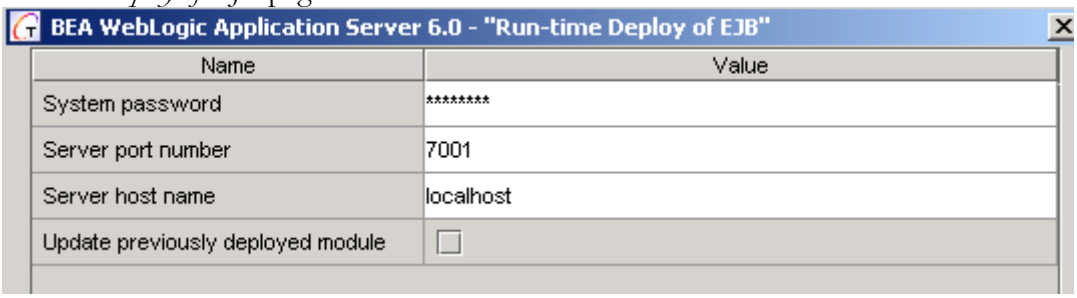
This example is based on *BEA WebLogic Server version 6.0* . It uses a simple example project *Hello World* which is located in `%TOGETHER_HOME%/samples/java/ejb/WeblogicServer`. To try it out, you need the following:

- Together product with Java language support and EJB support
- An installation of Java2 Enterprise Edition (SDK 1.2 or higher)
- An installation of Java2 SDK 1.3 or higher (installed with Together)
- An installation of *BEA WebLogic Server version 6.0* (local or accessible remote host)

Compiling and deploying the sample EJB

IMPORTANT: When deploying EJBs to BEA WebLogic 5.1, make sure your server does not already contain a deployed EJB with the same name as the one you are deploying (see *Verifying deployment* below). If it does, you should either restart WebLogic server after each deployment, or give your EJB a different name.

In J2EE Deployment Expert for WebLogic 6.0, if you already have deployed EJB with the same name, you should only check *Update already deployed module* checkbox in *Run-time deploy of EJB* page:



If it does, you must either restart WebLogic server after each deployment, or give your EJB a different name.

1. Open HelloWorld.tpr project from `%TOGETHER_HOME%/Samples/java/ejb/WeblogicServer/HelloWorld/` (*File | Open Project*).
2. Open *hello* diagram. You should see *HelloBean* class. Open the *Message pane*, so you can monitor process messages.
3. Check that the port number is correct, and password for WebLogic server is available.
4. Launch the *J2EE Deployment Expert* on the *Tools* menu. Choose *BEA WebLogic Application Server 6.0* as the default target server.
5. Make sure that all available check boxes are checked (including *Hot Deploy*).
6. Even though it's the default, open the drop-down list and select *BEA WebLogic Application Server 6.0* in the list to display server startup options.
7. In the server startup options, check *Start BEA WebLogic Server 6.0 (normal mode)* and uncheck *Start BEA WebLogic Server 6.0 (debug mode).*
8. Click *Next.*
9. Specify the paths to:
   - Java2 Enterprise Edition: ()
   - Java2 SDK1.3: (e.g. c:\jdk1.3),
   - WebLogic server(e.g. e:\weblogic)
   - Destination folders for the generated jar file and temporary files.
10. Click *Next.*
11. Specify the server host name (e.g. *localhost*), system password, and port number. Click *Finish.*

After that, Together does a number of things: starts WebLogic Server; invokes Java SDK compiler to compile the Hello bean; generates the necessary deployment files for Weblogic including deployment descriptor; invokes the SDK's JAR utility to package everything into a .JAR file; and communicates with the server to register the deployed EJB.

Verifying deployment

Assuming you have checked *Hot Deploy* in the J2EE Deployment Expert, the last message lines in the Message pane should be:

```
//WLS51: Connecting to localhost, port 7001...Successfully
connected.//
 //WLS51: Done deploying HelloWorld with
c:\temp\DEPLOYABLE_HelloWorld.jar//.
 //WLS51: Finished with 0 errors, 0 Warnings.//
```

The host name and port number values will match those you specified in the expert.

To be sure the generated JAR file was successfully deployed to the WebLogic server, you can run the WebLogic Console. In the console window, check for:

EJB Home interface*: 'hello.HelloHome'* deployed bound to the JNDI name*: 'hello.HelloHome'*.

This verifies that the bean is deployed.

Troubleshooting tips

If you have problems starting or running a WebLogic server, consult WebLogic documentation and/or technical support.

If you have problems connecting to a remote WebLogic server, consult the server's administrator or your network administrator.

Compiling and running the sample client

To test the deployed EJB, you need an EJB client to access it. The HelloWorld example contains a client class that you can compile and run to test your deployed sample EJB.

1. In the *Project Properties* dialog, check that the *Search/Classpath* specifications are as described in the project properties above. (*File | Project Properties*)
2. Open the `HelloWorld.client.weblogic` package in the Explorer and open the *weblogic* class diagram containing the *HelloClient* class.
3. Right-click on the client class and select *Tools | Make Node* from the speedmenu.
4. When compilation is complete the Message pane displays a message notifying that the tool is completed.
5. On the Main Menu choose *Tools | Run/Debug | Run*.

Your deployed EJB runs and outputs the string `"Hello World"`. Together picks up this output and displays it as a message line in the Message pane. The output also appears in the server console.

Debugging the sample bean and client

This section contains an exercise that you can follow to learn how to debug a bean and a client within the Together environment.

1. Close the running WebLogic Server.
2. For debugging your bean you must restart the server in debug mode. To do this:
   - On the Main Menu, choose *Tools | J2EE Deployment Expert* to launch the deployment expert again.
   - Choose *Start WebLogic Application Server 6.0*.
   - Uncheck *Start WebLogic Server 6.0 (normal mode)* and check *Start WebLogic Server 6.0 (debug mode)*. Click *Next*.

- Set a root directory of WebLogic Server 6.0 and click *Next*.

- Set Remote process port address and remember it. Click *Finish* to start server in debug mode.

3. Read instructions in J2EE Deployment Expert message window and follow them.
4. On the main menu choose *Tools* | *Run/Debug* | *Attach to Remote Process*.
5. In the subsequent dialog set the following values, then click OK:
   - Host: `localhost` (or the name of your server host)
   - Transport: `dt_socket`
   - Address: `<address>`
   - `<address>` - the value from the 2nd point above.
6. Deploy `HelloBean` to WebLogic Server as previously described.
7. Select `hello()` method of the `HelloBean` class in the diagram. In the Editor, set a breakpoint on the first string in it (F5 or speedmenu |*Toggle breakpoint*).
8. Compile the `HelloClient` class.
9. In the Editor, set a breakpoint on the first string after *try* statement in the main method.
10. On the Main menu choose *Tools* | *Run/Debug* | *Run in Debug*. The process will stop on the break point in the `HelloClient` class.
11. Use speed buttons to manage your process (*Restart/Resume* program, *Reset* program, step over, step in, etc. in the Debugger page of the process). Alternatively, you can use hot keys (Step over - F8, Step in - F7). When passing the line that invokes the `hello()` method, the debugger stops at the breakpoint in the `HelloBean` class.
12. When you have traced the whole `hello()` method, click the measure Program speed button on the remote process page to continue running the WebLogic. As a result you'll be stopped in the `HelloClient` class after the point at which the `hello()` method of the bean was invoked.
13. Continue step by step debugging, or click the *Resume Program* to finish the debugging process.

Deploying from an EJB Assembler diagram

In the previous sections we have deployed a bean described in a *Class* diagram. This technique is fine for prototyping, but in real scenarios you want to specify such things as container transaction attributes on classes and methods, security roles, and method permissions. The *EJB Assembler* diagram enables you to do this, so you can run the deployment expert against this type of diagram as well.

To deploy *Hello World* sample open Assembler Diagram as the current one, choose *Tools* | *J2EE Deployment Expert,* choose *BEA WebLogic Server 6.0* or *5.1*, and then read instructions in J2EE Deployment Expert message window and follow them.

For more information on this type of diagram and how to work with it see EJB Assembler diagram.

How to Create a Simple JSP Client

Together's deployment support will, at your option, generate a simple JSP-based client application to a location you specify in the J2EE Deployment Expert. You can use this client to access a deployed EJB running on the server. This capability is currently available for WebLogic Server 5.1 and 6.0, and WebSphere 3.5.

When you select *WebLogic Server 5.1* and check the *Generate Simple JSP Client* checkbox in the *J2EE Deployment Expert*, a Java Server Pages (JSP) client for Enterprise JavaBeans will be generated during deployment processing. This client is a set of interrelated JSP and HTML files which can be viewed in any Internet browser. The purpose of the JSP client is to provide access to a remote EJB object through its open interfaces (i.e., "Remote" and "Home"). Generally, the "Home" interface is intended for control over remote object life cycles and the "Remote" interface is for calling remote object business methods. The simple generated JSP client is able to perform these operations and save you the necessity to write your own client program for testing purposes.

Some notes that are worth mentioning:

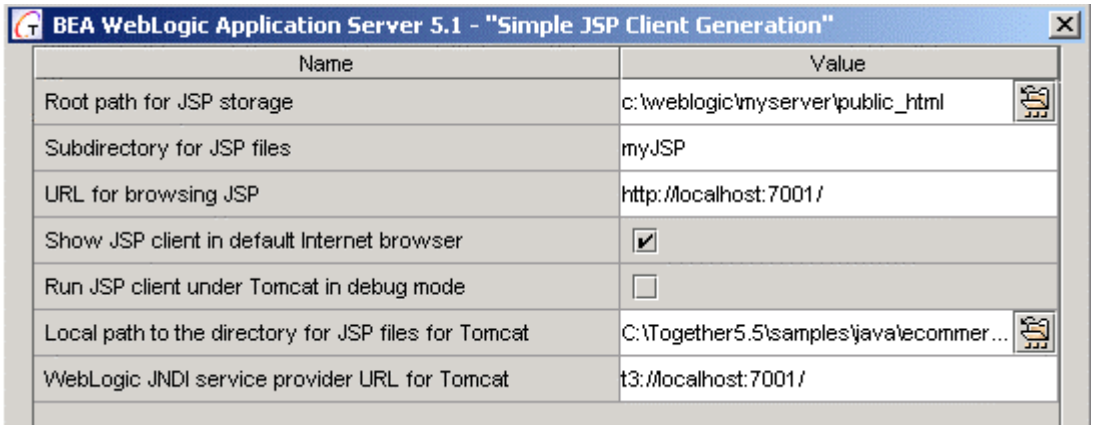
JSPs are "active" pages... you can view them only through the server (in this case - through the WebLogic 5.1 or WebSphere 3.5).

The semantic information necessary to generate a JSP client comes from the Together project's currently active diagram. Because this provides rather sparse information, it is not practical to create a sophisticated, commercial-grade client program. Therefore you need to consider the generated JSP client as universal testing facility.

Setting values on the JSP client page

When you check the JSP client generation option in the Deployment Expert, an additional page is included in the page sequence. You fill in a number of fields on this page with information to support JSP client generation.

Fields for the WebLogic 5.1 version



The following fields are displayed in the Deployment Expert when WebLogic 5.1 is the selected server:

Root path for JSP storage

The root of the WebLogic file hierarchy which can be accessed from outside. Thus if you want to allow access to some HTML for JSP files, then you need to allocates it somewhere here. By default the WLS public directory allocated at `[WebLogic home]/myserver/public_html` .

Subdirectory for JSP files

To avoid disorder in public directory it is desirable to allocate files by separated groups in some subdirectory. You can assign the path to a subdirectory where your JSP files are to be placed. This path is considered relative to root of the public directory. The subdirectory will be created automatically if it does not exist (after confirmation).

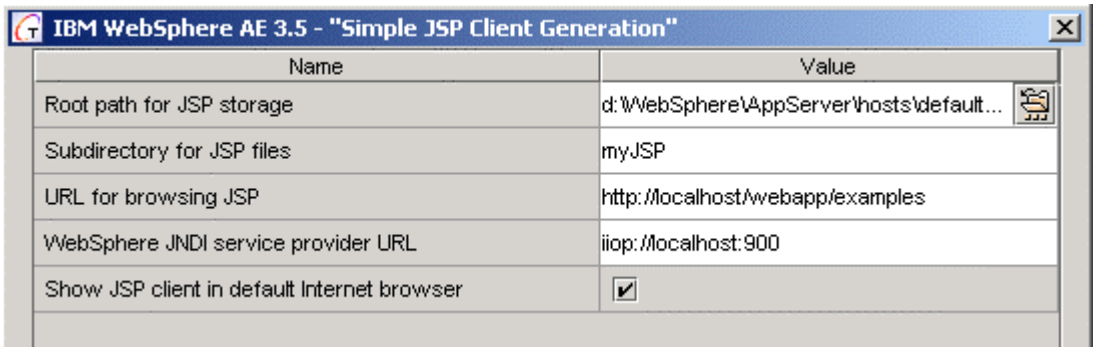
URL for browsing JSP

In order to browse any remote resource it is of course necessary to specify its URL. Standard Web URL includes protocol, host, port, web resource alias, and path. In this field you have to set all necessary parts except for the *path*. During request processing, the server will replace this base URL with the WebLogic public directory in order to access the requested web resource. If you make a typing error, or specify erroneous relativity to the WebLogic public directory, your will get an access error when you try to load the JSPs in a browser. By default, the base URL is `http://localhost:7001/`.

Show JSP client in default Internet browser

This field is a flag that indicates whether to show the start page after successful generation. If you check it, the standard Internet browser will launch and show the start page.

Fields for the WebSphere 3.5 version



The following fields are displayed in the Deployment Expert when WebSphere 3.5 is selected:

Root path for JSP storage

This directory is the root of file hierarchy, which can be accessed from outside. Thus, if you want to allow access to some HTML of JSP files, you need to allocate them accordingly.

Subdirectory for JSP files

To avoid disorder in public directory, it is desirable to allocate files in separate groups by the means of a subdirectory. You can assign the path to the subdirectory where your files will be placed. This path is relative to the root public directory. If the subdirectory doesn't exist, it will be created automatically (after confirmation).

URL for browsing JSP

In order to browse remote WEB resource, you have to set URL. Standard Web URL includes protocol, host, port, web resource alias, and path. In this field you have to set all necessary parts except for the path. During request processing, server will substitute this Web path to WebSphere document root directory in order to access the requested web resource. If this field, or the root directory field, is wrong, an access error will be reported when you try to load JSPs in a browser. By default root Web path is `http://localhost/`.

WebSphere JNDI service provider URL

This field is required to establish connection between JSP client and EJB server. It is well known that the client should first send request to the name service provider, in order to find EJB service provider. This value appends to the JSP files in course of generation and is used during execution. Normally, you do not need to change this field. Default value is `iiop://localhost:900`.

Show JSP client in default Internet browser

This field is a flag that indicates whether to show the start page after successful generation. If you check it, the standard Internet browser will launch and show the start page.

General considerations

WebSphere Application Server supports multiple document roots. Each root is associated with a so called "Web Application". The field values on the current page should comply with those of the Web application.

You can create new Web Application and automatically configure it in the "Process Servlet(s)" step. Otherwise you are personally responsible for the existence of Web Application.

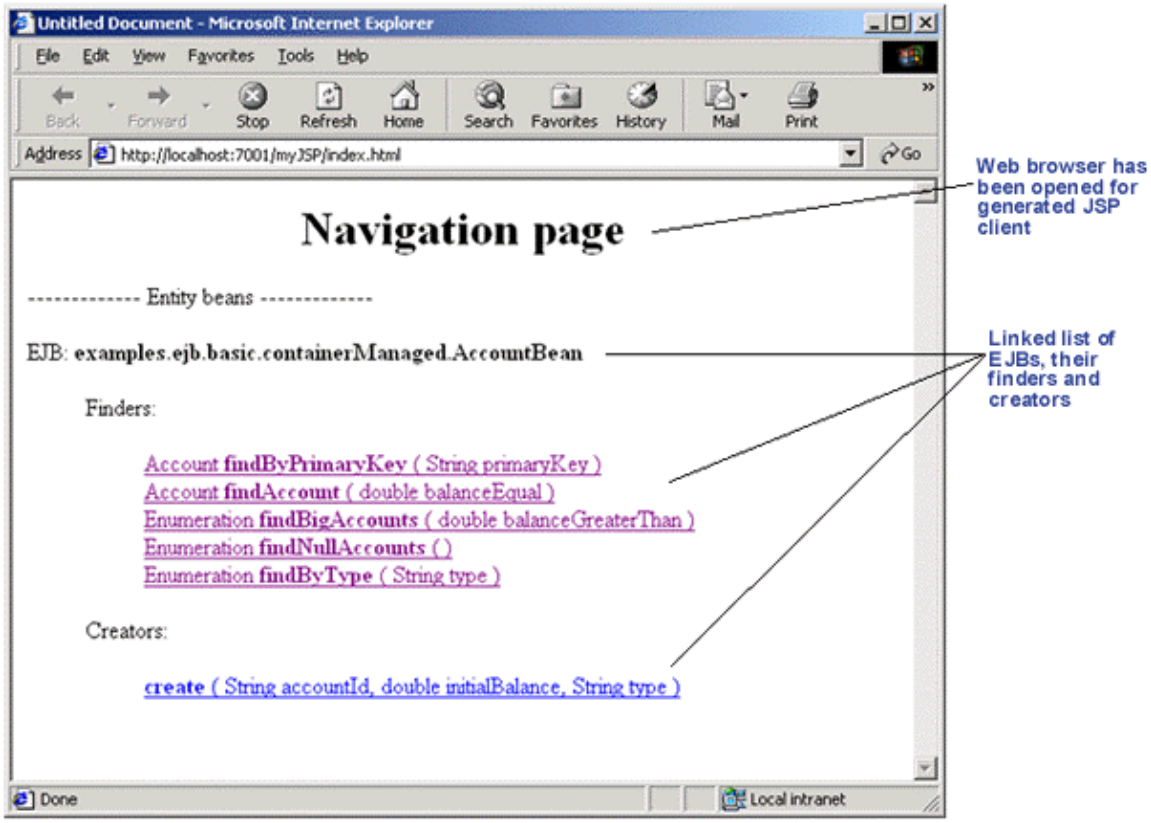
If you manually configure Web Application, make sure that:

1. "JSP 1.0 support servlet" `com.sun.jsp.runtime.JspServlet` is appended to your Web Application.
2. Servlet "Enables File Serving" (`com.ibm.servlet.engine.webapp.SimpleFileServlet`) is appended to your Web Application.

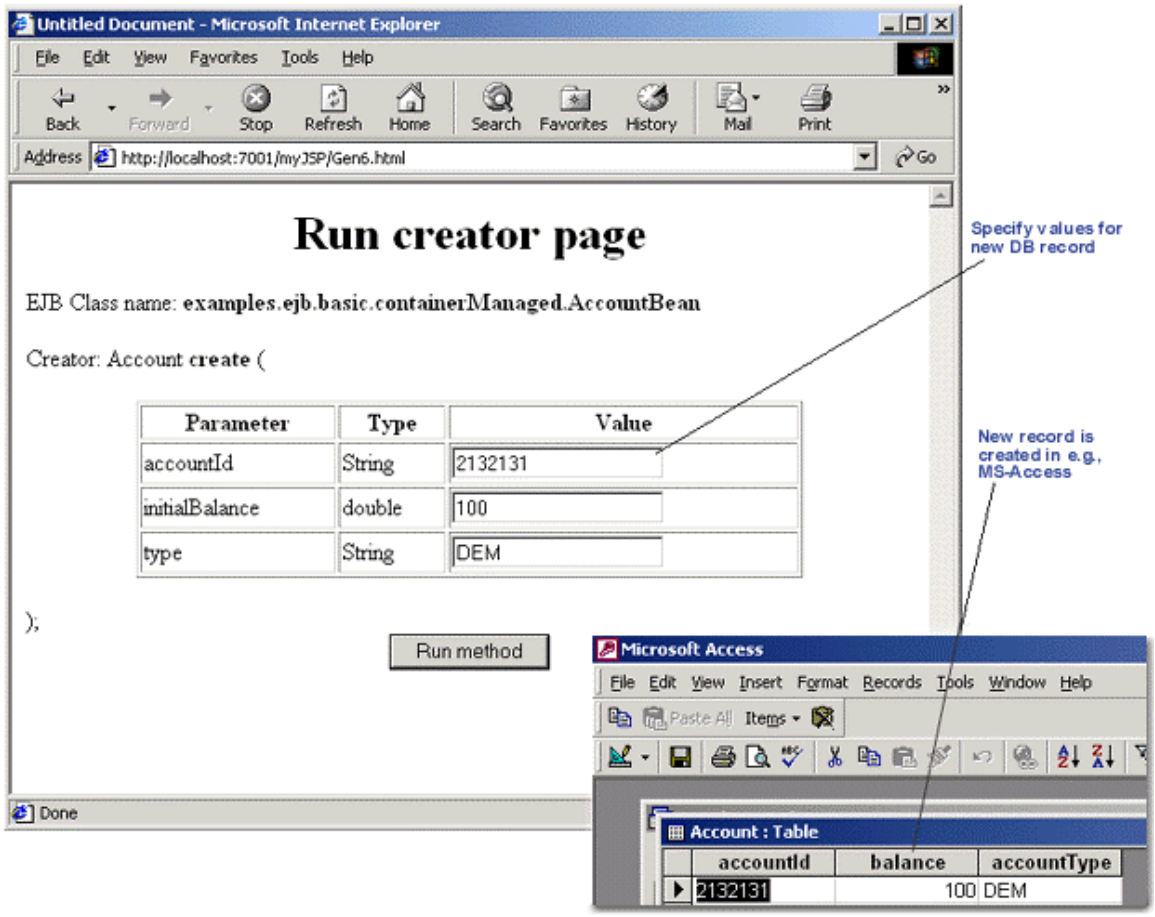
   Note: If these servlets are not available, the corresponding options on the "Servlet Properties" page of J2EE Deployment Expert should be checked.
3. Path to your deployed enterprise beans is added to the Web Application classpath.

Testing the deployed EJB using the JSP client

If you check the *Show result after generation* option, your browser opens and loads the html index page at the location you specified as the path for JSP browsing. Alternatively, you can use your browser to open the index.html file at that location.

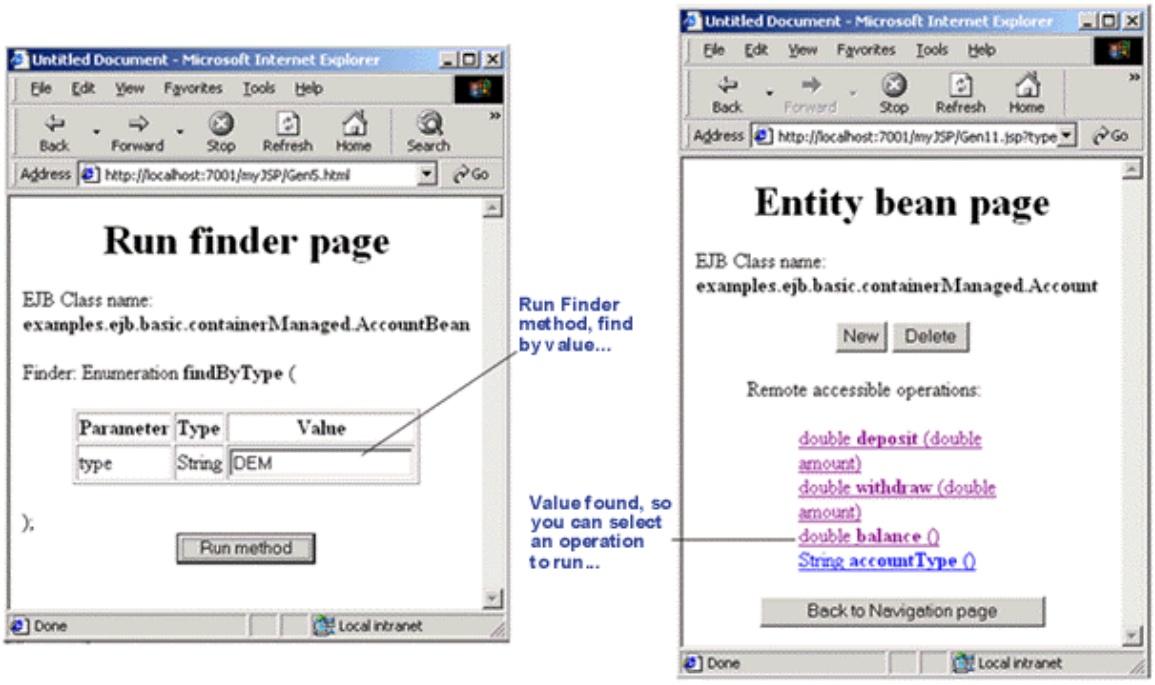


You can click to link to the *create()* method to launch another JSP that creates and instance of the EJB object and enables you to access it's fields, as shown in the following figure.

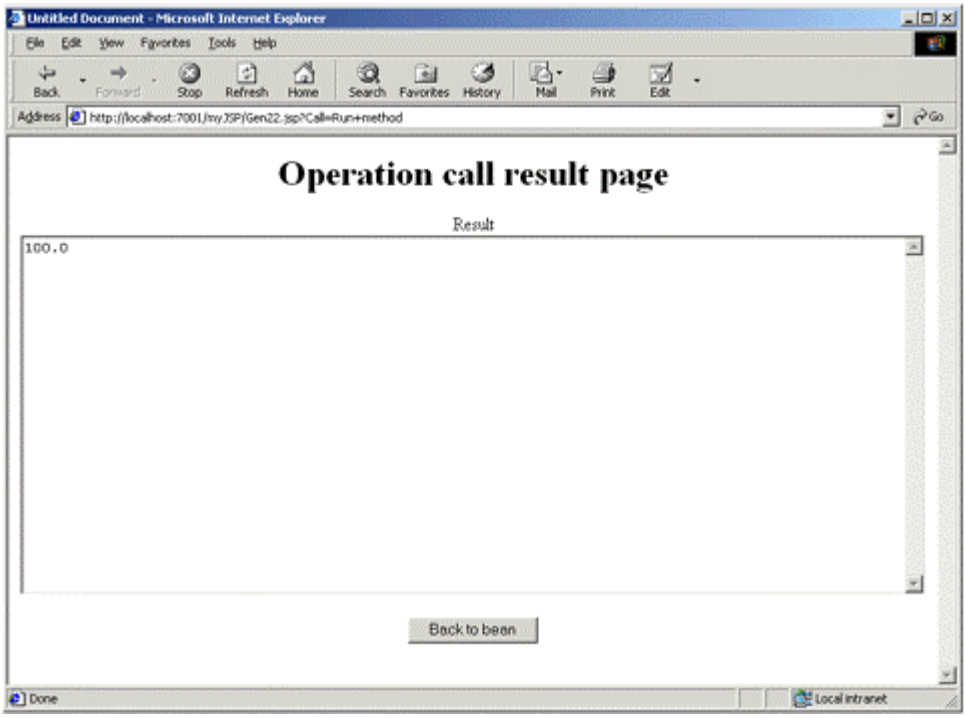


Of course, the field content and results of the interaction will vary with the design and coding of the EJB... this interaction of an EJB running against an Access database is just an example of how you can use the JSP client to test a running EJB.

To carry on with the example, you can use the client to test the Finder method on an EJB by returning to the initial navigation page and running that method against a known value, as shown below:



In this example, the result of the operation proves that the Bean and its business logic is functioning correctly:



Step By Step How To Create a Servlet and Deploy it to WebLogic 5.1

Creating a Servlet

Create a New Project (*Main Menu* | *File* | *New Project*). Enter "HelloWorld" as the project name. As you won't need standard libraries, remove them from the project: click *Advanced*, select *Search Classpath* and make sure that check boxes *Include standard libraries* and I*nclude Classpath* are cleared. Click OK.

Create a New Package, name it "hello" and choose *Open* in *New* Tab from the speedmenu. In this package, create a new servlet "HelloWorld" using Class by Pattern icon on the toolbar.

Edit `doGet`method of the servlet. Instead of the line `"//Write your HTML here",` type `out.println("Hello world");`

Now you are ready to deploy this Servlet to WebLogic 5.1

Compiling and running the Servlet

First, you have to run the WebLogic Application server. Uncomment the line `#weblogic.httpd.register.servlets=weblogic.servlet.ServletServlet` in the file `weblogic.properties`

Start WebLogic Server 5.1. To do this, use WebLogic Server starter plugin, provided in the J2EE Deployment Expert. Select *Start BEA WebLogic Application Server 5.1* on the first page of the Expert. Make sure that the flag *Start BEA WebLogic Server 5.1 (normal mode)* is checked, and the flag *Start BEA WebLogic Server 5.1 (debug mode)* is unchecked. Click *Next*.

On the next page of the Expert, set root directory of WebLogic Server 5.1 and click *Finish* to start the server.

> **Note:** Alternatively you can start WebLogic Server 5.1 from the Windows *Start* | *Programs* | *WebLogic 5.1* | *WebLogic Server*.

Next, you have to compile the servlet. Invoke J2EE Deployment Expert again and select *BEA WebLogic Application Server 5.1* as the target server. Make sure that checkboxes *Add libraries required for*...and *Process Servlet(s)* are checked, while the other check boxes are cleared. Click *Next*.

Set paths to jdk1.2 (e.g. `c:\jdk1.2.2`), to WebLogic (e.g. `c:\wls51`) and destination paths for the resulting Servlet default (`%WL_HOME%myserver\servletclasses`) and temporary files. Click Finish.

If the compile process successfully completed, the Message Pane should display the following message:

```
//WLS51: Finished with 0 Errors, 0 Warnings.//
```

Open your browser and set address `http://localhost:7001/servlets/hello/HelloWorld`. You should see "Hello World".

Step By Step How To Create a Servlet and Deploy it to WebSphere 3.5

Creating a Servlet

Create a New Project (*Main Menu | File | New Project*). Enter "HelloWorld" as the project name. As you won't need standard libraries, remove them from the project: click *Advanced*, select *Search Classpath* and make sure that checkboxes *Include standard libraries* and I*nclude Classpath* are cleared. Click *OK*.

Create a New Package, name it "hello" and choose *Open* in *New* Tab from the speedmenu. In this package, create a new servlet "HelloWorld" using Class by Pattern icon on the toolbar.

Edit `doGet`method of this servlet:

Find	Replace
`//Write your HTML here`	`out.println("Hello world");`

Now you are ready to deploy this Servlet to WebSphere 3.5

Compiling and running the Servlet

First, start WebSphere Server 3.5 services from *Control Panel | Services*. Next, in Together, invoke J2EE Deployment Expert.

Select *IBM WebSphere AE 3.5* as the target server and make sure that all available checkboxes, except of *Process Servlet(s)*, *Hot deploy to IBM WebSphere*, and *Add libraries required for deployment to the current project's Search/Classpath*, are off.

Specify correct paths to WebSphere home directory and to IBM JDK 1.2.2 root directory (e.g. `c:/WebSphere/AppServer` and `c:/WebSphere/AppServer/jdk`). Click *Next*.

Set the following parameters on the *EJB Deployment Properties* page:

Parameter name	Setting	Comment
Admin Node Name	Name of the Node to deploy	Name of the computer
Dependent ClassPath	Fully qualified path to WebSphere	Check if it refers to existing jars
Application Server Name	Your Application Server	"Default Server" by default
EJB Container Name	Your Container name	"Default Container" by default
Target WebSphere server directory	WebSphere home directory	e.g. `c:\WebSphere\AppServer`
Stop server and remove existing Beans/ Servlet before deployment	ON	
Start Beans/Servlets after deployment	ON	
Launch server in debug mode	OFF	

Set the following parameters on the *Servlet Deployment Properties* page:

Parameter name	Setting	Comment
Virtual Host name	Your virtual host	"default_host" by default
Servlet Engine name	Your servlet engine	"Default Servlet Engine" by default
Web Application name	Your web application name	"Default_app" by default
Web Application document root	The WebSphere address accessible through an Internet browser	Empty by default
Relative Path to Servlet	Your directory name	"/" by default
Relative Web Path to Servlet directory	Web resource alias for your Web application	"servlet" by default

All check boxes, except *Load Servlet at StartUp* and *Servlet Enabled*, are off. Click *Finish* when all parameters are properly specified.

If you have also checked *Hot Deploy* check box, the last message in the Message Pane is:

```
//was35: Finished with 0 Errors, 0 Warnings.//
```

Start your Internet browser, enter address `http://localhost/servlet/HelloWorldServlet`, and observe "Hello World" message.

Step By Step How To Deploy CMP Entity Bean from IBM WebSphere 3.5 Samples to BEA WebLogic 5.1

This topic shows you step-by-step how to deploy a CMP entity bean from IBM WebSphere 3.5 to BEA WebLogic 5.1 and run a client for it.

Creating CMP Entity Bean

Creating a project

First, provide sources for the project. Copy EJB-related files from `WebSphere\AppServer\EJBSamples\src\Increment` into a directory structure according to the package `%TG_HOME%\Samples\java\ejb\WebSphere`. Thus, the example will reside in the same place with the beans.

```
Increment.java
IncrementBean.java
IncrementKey.java
IncrementHome.java
VisitIncrementSite.java
```

Start Together and create project in `%TG_HOME%\Samples\java\ejb\WebSphere\WebSphereSamples\Increment`.

Together automatically recognizes the structure of parent directories, so that the upper folders correspond to the package statements. This helps avoid errors.

Editing the bean properties

Click on the Bean implementation class, invoke Inspector, go to *EntityEJB* and look at the *General* tab. Together recognizes fully qualified names of the classes, interfaces etc.

Now, set JNDI name for the *IncrementBean*. In the *Properties* tab of *EntityEJB* set *JNDI name to bind to* to *Increment* (it is not recognized by Together since it is stored in XML file rather than in the sources.).

In the *Fields and Finders* tab, set column property for *count* field to *counts*. This should be done because the word "count" is reserved.

Select *DB binding* tab and make sure that table name is set to "Increment", pool name is set to "demoPool", and Scheme name is set to "APP". EJB is now ready.

Creating Database

Creating a table in the database

Let us create a table in the *cloudscape* database. To do this, invoke Generate DDL Expert on the *Tools | Database Import/Export*, and select: *Export from:Enterprise Java Bean*. In the Expert, choose *Diagram name: Increment*, and click *Next*. On the next page, choose *Generate and Run DDL*. Enter the settings listed below and click *Next*.

Field	Value
Server type	`Cloudscape`
Set class path	`%WL_HOME%/eval/cloudscape/lib/cloudscape.jar`
Set Database	`%WL_HOME%/eval/cloudscape/data/demo`

Make sure that Schema name is set to APP. Click *Edit DDL* button and rename *count* field to *counts*. Change primaryKey type from VARCHAR(1) to NATIONAL CHAR VARYING(10). Replace name *IncrementBean* with *Increment*.

Restart Together (The evaluation version of Cloudscape that ships with BEA WebLogic Server supports only one database connection at a time to a `cloudscape.system.home` directory **http://www.weblogic.com/docs51/techsupport/cloudscape.html#settingup**)

Creating a Cloudscape pool

Ok, we're ready to take off! Now, start BEA WebLogic 5.1 application server (either manually from the Start menu, or using our starter plug-in). Uncomment the following lines in `%WL_HOME%\weblogic.properties` before you run the server:

```
"weblogic.httpd.register.servlets=weblogic.servlet.ServletServlet"
```

This allows to run unregistered servlets from the servlet classpath.

```
"weblogic.allow.reserve.weblogic.jdbc.connectionPool.demoPool=everyone"
"weblogic.jdbc.connectionPool.demoPool=\
 url=jdbc:cloudscape:demo,\
 driver=COM.cloudscape.core.JDBCDriver,\
 initialCapacity=1,\
 maxCapacity=2,\
 capacityIncrement=1,\
 props=user=none;password=none;server=none"
```

This creates a pool to work with Cloudscape.

Deploying the Bean to BEA WebLogic Server 5.1

In the J2EE Deployment Expert, make sure that all checkboxes are on, except for *Generate a simple JSP client*, and *Generate a command line file with instructions for deployment*. Specify correct paths / host / password / whatever required by the Expert, and click *Finish*.

In the XML Editor, make sure that description for the *IncrementBean* has been retrieved directly from the sources, without any user's efforts! Check that EJBName is short *IncrementBean*. Click *Ok* to continue.

In the XML Editor for `weblogic-ejb-jar.xml` go to `weblogic-ejb-jar/weblogic-enterprise-bean/ejb-name` and make sure it is *IncrementBean*. The associated JNDI name in `weblogic-ejb-jar/weblogic-enterprise-bean/jndi-name` is *Increment*. It is taken from the properties, specified in the previous

step. Click *Ok* to continue.

After a while Together will display a message

```
BEA WebLogic Server 5.1: Finished with 0 Errors, 0
Warnings
```

BEA WebLogic Server 5.1 console displays a message like:

```
Fri Aug 25 12:06:44 GMT+02:00 2000: EJB home
interface:
'WebSphereSamples.Increment.IncrementHome' deployed
bound to the JNDI name: 'Increment'
```

Ok, deployment complete. Now let us make some changes to the client.

Creating the Client

Editing the Client

Now you have to edit the client class.

Location	Find	Replace
VisitIncrementSite	incrementHome = (IncrementHome)javax.rmi. PortableRemoteObject. narrow ((org.omg.CORBA.Object) homeObject, IncrementHome.class)	incrementHome = (IncrementHome)javax.rmi. PortableRemoteObject.narrow (homeObject, IncrementHome.class);
init()	Hashtable env = new Hashtable(); env.put (Context.PROVIDER_URL, provider); env.put (Context.INITIAL_CONTE XT_FACTORY, factory);	comment out
init()	InitialContext ctx = new InitialContext(env);	InitialContext ctx = new InitialContext();
doGet()	out.println("form method=\"get\" action=\"/servlet/ WebSphereSamples.Incre ment. VisitIncrementSite\">");	out.println("<form method=\"get\" action=\"/servlet/ WebSphereSamples.Increment. VisitIncrementSite\">");

Compiling and running the client

When all editing is complete, compile `VisitIncrementSite` and copy the compiled class `VisitIncrementSite.class` into a directory structure according to the package statements (e.g. `%WL_HOME%\myserver\servletclasses\WebSphereSamples\Increment`).

Now you are ready to run the client. Start your browser, open `http://localhost:7001/servlets/WebSphereSamples/Increment/VisitIncrementSite` and observe "**Increment a Counter**....."

Note, that when we deploy a CMP Entity bean from IBM WebSphere 3.5 samples to BEA WebLogic 5.1, the underlying java source code changes (from EJB 1.0 specification to EJB 1.1 specification):

1. the returned value of ejbCreate method changes to `primaryKey class` in CMP EJBs,
2. `return` statement is added,
3. `transaction attributes` are changed (we have no `transaction attributes` in our sample)

Step By Step How To Deploy Session Bean from IBM WebSphere 3.5 Samples to BEA WebLogic 5.1

This topic takes you step by step through creating a Session bean, deploying it to WebLogic Server 5.1 and creating a client for it.

Creating and Deploying a Session Bean

Creating the project

To avoid compilation errors, create directory structure that corresponds to the package statements of the WebSphere samples. Copy EJB-related files from `WebSphere\AppServer\hosts\default_host\WSsamples_app\servlets\WebSphereSamples\HelloEJB` into `%TG_HOME%\samples\java\ejb\WebSphere\HelloEJB\com\ibm\ejb\cb\samples\hello\tier2`. Thus, the is placed example close to our delivered ejb samples:

```
Hello.java
HelloBean.java
HelloBeanResourceBundle.java
HelloHome.java
```

Start Together and create new project in `%TG_HOME%\samples\java\ejb\WebSphere\HelloEJB\com\ibm\ejb\cb\samples\hello\tier2`

Together automatically recognizes the structure of parent directories, so that upper folders correspond to the package statements. This helps avoid errors.

Modifying the Bean Properties

Now you are ready to edit the bean. Navigate to the Bean implementation class, go to SessionEJB and look at the properties. Together recognizes fully qualified names of the classes, interfaces etc.

Select HelloBean on the diagram and open its Object Inspector. On the *SessionEJB* page of the Inspector, choose *Properties* tab and set *JNDI name to bind to - HelloHome* (it was not recognized by Together since stored in XML file rather than in sources.)

Ok, we're now ready to take off!

Deploying to BEA WebLogic Server 5.1

Uncomment the string `weblogic.httpd.register.servlets=weblogic.servlet.ServletServlet` in `%WL_HOME%\weblogic.properties`. This string allows to run unregistered servlets from the servlet classpath.

Start WLS Server (manually from the Start menu, or using our starter plug-in).

Start deployment to BEA WebLogic Server 5.1. All checkboxes, except for *Generate a simple JSP client* and *Generate a command line file with instructions for deployment* should be on.

Specify correct paths / host / password / whatever required by the Expert and click *Finish*.

Edit `ejb-jar.xml` in the XML Editor. Go to `ejb-jar/enterprise-beans/session/Description (*)` and make sure that the description of HelloBean is retrieved directly from the sources, without any user's intervention! Check that EJBName is short "HelloBean". Click *Ok* to continue.

Next, edit `weblogic-ejb-jar.xml`. Do to `weblogic-ejb-jar/weblogic-enterprise-bean/ejb-name` and make sure it is "HelloBean", and the associated JNDI name in the field `weblogic-ejb-jar/weblogic-enterprise-bean/jndi-name` is "HelloHome". This name is taken from the properties, which were specified in step 3. Click Ok to continue.

After a while you'll see a message in Together's Message pane:

```
WLS51: Finished with 0 Errors, 0 Warnings
```

WLS console shows something like:

```
Fri Aug 25 12:06:44 GMT+02:00 2000: EJB home
interface:
'com.ibm.ejb.cb.samples.hello.tier2.HelloHome'
deployed bound to the JNDI name: 'HelloHome'
```

Ok, deployment is now complete. Let's make ourselves busy with the client.

Creating a Client

Copy client-related files into the proper directory structure (for example, `%TG_HOME%\samples\java\ejb\WebSphere\HelloEJB\WebSphereSamples\HelloEJB`).

Go to the *Project Properties* and add second source root path `%TG_HOME%\samples\java\ejb\WebSphere\HelloEJB\WebSphereSamples\HelloEJB`. Though our EJB and client files reside in different folders, it is very easy to access the sources since source paths of our project point to the very end of hierarchy!

Build the project (actually only the client has to be compiled). Make sure that `DEPLOYED_togetherEJB.jar` is added to the project classpath. If this jar file is missing, open *Project | Options* dialog and modify the Classpath on the *Builder* page.

Go to the output directory and copy compiled client classes (`%OUT_DIR%\WebSphereSamples\HelloEJB\*.class`) to the server's `servletclasses` folder (`%WL_HOME%\myserver\servletclasses\WebSphereSamples\HelloEJB`).

Open `http://localhost:7001/servlets/WebSphereSamples/HelloEJB/HelloEJBServlet` in your browser and observe "**Hello** from Hello EJB Sample!"

Step By Step How To Deploy CMP Bean from BEA WebLogic 5.1 Samples to IBM WebSphere 3.5

This topic shows you step-by-step how to deploy a CMP bean from WebLogic 5.1 to IBM WebSphere 3.5 and run a client for it. The sample resides in `%WL_HOME%\examples\ejb\basic\containerManaged`.

Create a project and edit CMP bean

Creating the project

First, you have to provide the source files for the project. Copy all .java files from `%WL_HOME%\examples\ejb\basic\containerManaged` to `%TG_HOME%\myprojects\cmp\examples\ejb\basic\containerManaged`.

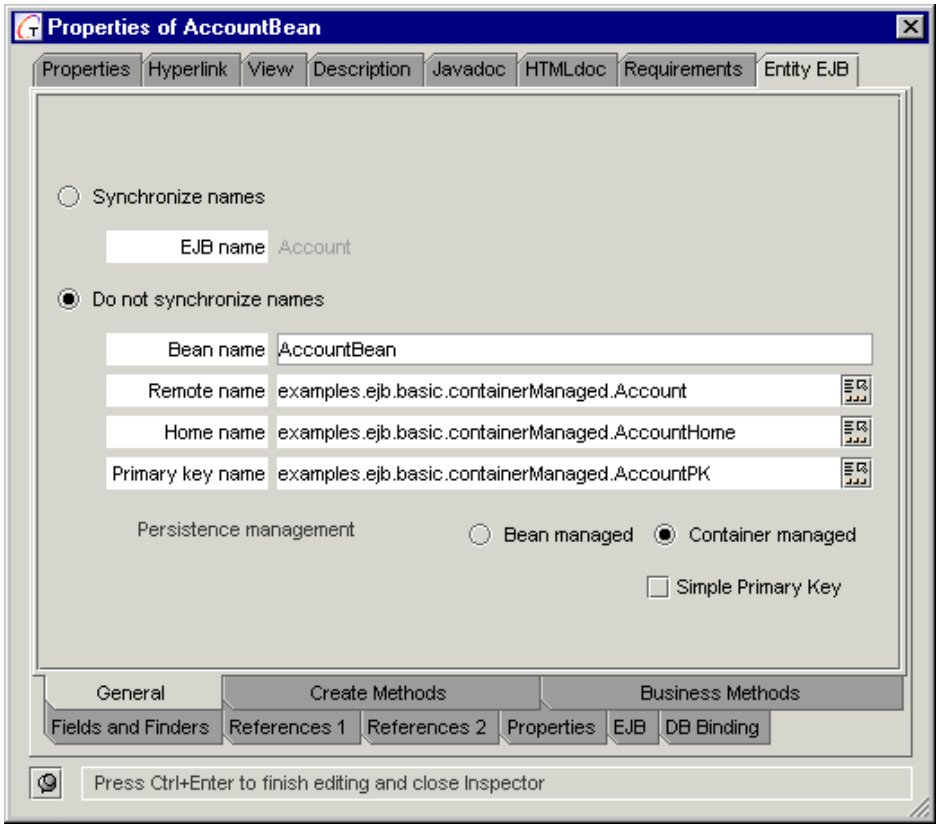
Next, start Together, create a New Project and set its location to `%TG_HOME%\myprojects\cmp`. Enter 'cmp' as the project name. Since the standard libraries are not required, you can remove them from the project. To do that, in the Advanced mode, select *Search / Classpath* and make sure that checkboxes *Include standard libraries* and *Include Classpath* are cleared. On the *EJB* tab select *IBM WebSphere AE 3.5* as the target server.

Modifying the Bean Properties

Now you are ready to edit the bean. Navigate to *containerManaged* node in the Explorer, select *Open Diagram* on the speedmenu and open `containerManaged` package.

Select *AccountBean* in the diagram, and open its Object Inspector. Make the following changes in the *Entity EJB page* of the Inspector.

On the *General* tab, choose *Don't synchronize names*. Next, create the Primary Key. Type `examples.ejb.basic.containerManaged.AccountPK` in the field *Primary Key Name* and click Enter. This will bring in a dialog, where you have to choose *Create* button to create the Primary Key Class.



On the *Fields and Finders* tab, choose *AccountId* as the primary key. Next, set the 'finder query' properties for finder methods as follows:

```
'findBigAccounts' : 'select * from accountbeantbl where
balance > ?',
'findAccount': 'select * from accountbeantbl where balance =
?',
'findNullAccounts': 'select * from accountbeantbl where
accountType is null '.
```

On the *Properties* tab, set JNDI name to AccountHome. Close the Object Inspector.

Editing the Bean

Select AccountBean in the Diagram pane and edit its source code. Comment out import statement:

```
import.javax.ejb.NoSuchEntityException
```

Create two constructors in the `AccountPK` class, one being a default constructor, and the other being a constructor with parameters. This is how it's done: select this class in the model treeview and create two new constructors (*New* | *Constructor* from the speedmenu). Open one of them in the text editor, add new constructor and edit methods as follows:

```
public AccountPK(String param) { accountId = param; }
```

As a result, we get constructors

```
public AccountPK(String param) { accountId = param; }
public AccountPK() { }
```

Edit `hashCode()` method in the `AccountPK`:

```
public int hashCode() { return accountId.hashCode(); }
```

Edit `equals()` method in the `AccountPK`:

```
    public boolean equals(Object other) {
       if (other instanceof AccountPK) {
          AccountPK otherKey = (AccountPK) other; return
            accountId.equals(otherKey.accountId);
       } return false;
}
```

Now you are ready to deploy this bean to WebSphere 3.5

Editing the Client

Before passing to the deployment, we have to edit the Client class.

- Replace all locations of `findByPrimaryKey(id)` with `findByPrimaryKey (new AccountPK(id))`
- In the main() method replace `String url = "t3://localhost:7001";` with `String url = "iiop://localhost:900";`
- In the `getInitialContext()` method replace `h.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.initialContextFactory");` with `h.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");`
- In the `lookupHome` replace `Object home = (AccountHome) ctx.lookup("containerManaged.AccountHome");` with `Object home = (AccountHome) ctx.lookup("AccountHome");`
- In the `findNullAccounts()` method replace `Enumeration enum = (Enumeration) PortableRemoteObject.narrow(home.findNullAccounts(), Account.class);` with `Enumeration enum = home.findNullAccounts();`
- In the `findNullAccounts()` method replace `Account nullAccount= (Account) enum.nextElement();` with `Account nullAccount= (Account) PortableRemoteObject.narrow(enum.nextElement(), Account.class);`

The client is now ready.

Deploying the Bean

Precondition

Start WebSphere server:

- Select *Control panel* | *Services from Start* | *Settings* menu.
- Select *IBM WS AdminServer* and click *Start* button.

Note: "IBM HTTP Administration" and "IBM HTTP Server" should be already started.

Deploying the bean

1. Run the J2EE Deployment Expert on the Tools menu.

2. Select "IBM WebSphere AE 3.5" as the target server and make sure that all available checkboxes, except *Process servlets, Generate WLM jar for Bean,Generate start and compile file for client, Generate simple JSP, Generate command file for deployment client* are set. Press Next.

3. Set paths to WebSphere home directory and to IBM JDK 1.2.2 root directory (e.g. "`c:\WebSphere\AppServer`" and "`c:\WebSphere\AppServer\jdk`"). Press Next.

4. Set the following parameters on the next page:

Parameter name	Setting	Comment
Admin Node Name	Name of the Node to deploy	Name of the computer
Dependent ClassPath	Fully qualified path to WebSphere	Check if it refers to existing jars
Application Server Name	Your Application Server	"Default Server" by default
EJB Container name	Your Container name	"Default Container" by default
Target WebSphere server directory	WebSphere home directory	for example `c:\WebSphere\AppServer`
Create Table for CMP EJBs	ON	
Stop & Remove Beans(s), Servlet(s) before deployment	ON	
StartBeans(s), Servlet(s) after deployment	ON	
Launch server in debug mode	OFF	

5. Press Next and Finish.

If you have checked Hot Deploy check box, the Message Pane will display a message about successful deployment completion:

```
//WAS35: Finished with 0 Errors, 0 Warnings.
```

Compiling and running the client

To provide proper compilation, specify JDK Home location as %WAS_Home%\AppServer\jdk on the *Run/Debug* tab of the Project | Options dialog.

Select 'Client' class on the diagram and choose Rebuild Node on its speedmenu. The Builder pane displays a message that the tool successfully completed.

Now, choose Run command on the Run/Debug menu and make sure that 'Class with main' is set to `examples.ejb.basic.containerManaged.Client`. Click *OK* to start. Observe the messages in the Run/Debug pane:

```
Beginning containerManaged.Client...
Starting Part A of the example...
Creating account 10020 with a balance of 3000.0 account type
Savings...
Account 10020 successfully created
...
Removing beans...
End Part B of the example...
End containerManaged.Client..
```

Note, that when we deploy CMP Entity bean from BEA WebLogic 5.1 samples to IBM WebSphere 3.5, the corresponding source java code changes appropriately (EJB 1.1 specification to EJB 1.0 specification):

1. `EJB Exceptions` are taken away;
2. The returned value of `ejbCreate` method changes to `void` in CMP EJBs,
3. `return` statement in the method's body is commented out,
4. `transaction attributes` are changed (there are no `transaction attributes` in our sample)

Step By Step How To Deploy Session Bean from BEA WebLogic 5.1 Samples to IBM WebSphere 3.5

This topic shows you step-by-step how to deploy a Session bean from WebLogic 5.1 to IBM WebSphere 3.5 Samples and run a client for it. The sample resides in `%WL_HOME%\examples\ejb\basic\statelessSession`.

Creating a project and editing Session bean and Client

Creating the project

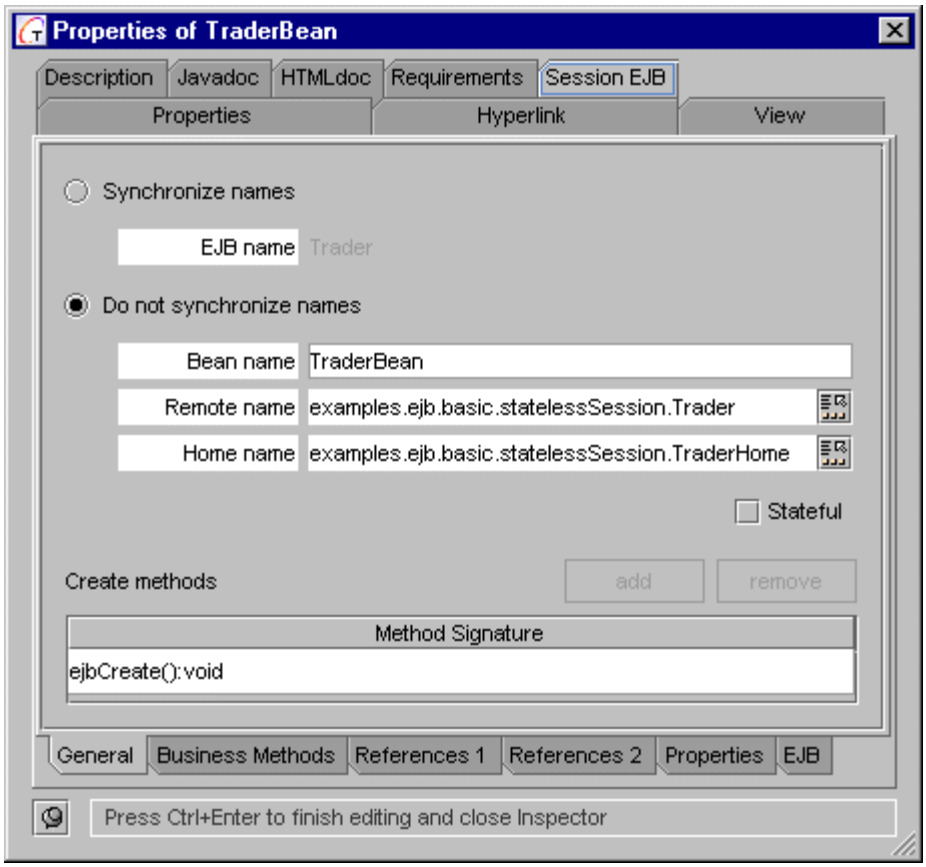
Copy all `*.java` files from `%WL_HOME%\examples\ejb\basic\statelessSession` to `%TG_HOME%\myprojects\session\examples\ejb\basic\statelessSession`.

In Together, create a New Project and set its location to `%TG_HOME%\myprojects\session`. Enter 'session' as the project name. As you won't need standard libraries, remove them from the project: press Advanced button, select *Search/Classpath* tab and clear the checkboxes 'Include standard libraries' and 'Include Classpath'. On the *EJB* tab, select IBM WebSphere AE 3.5 server.

In the Explorer, navigate to *statelessSession* node, choose Open Diagram on its speedmenu and open `statelessSession` package.

Modifying the Bean properties

In the Diagram pane, select TraderBean, and open its Object Inspector. Select *Session EJB* page, that provides a number of tabs. In the *Properties* tab, set JNDI name to "TraderHome". Next, in the *General* tab, select radio button 'Don't synchronize names'. Next, specify the Remote name of the EJB as `examples.ejb.basic.statelessSession.Trader`, and press Enter.



Having entered the remote interface name, you have a choice to rename the existing one, or to create a new one. Press *Create* button, to create the Remote interface for this session bean.

Editing the Bean source code

Edit constructor `ejbCreate()`:

Find	Replace

`try {`
`  InitialContext ic = new InitialContext();`
`  Integer tl = (Integer)`
`ic.lookup("java:/comp/env/tradeLimit");`
`  tradeLimit = tl.intValue();`
`  }`
`  catch (NamingException ne) {`
`  throw new CreateException("Failed to find`
`environment value "+ne);`
`  }`
	`tradeLimit = 300;`

You are now ready to deploy this bean to WebSphere 3.5

Editing the Client source code

Select Client class on the diagram, and make the following changes in the Editor pane:

Location	Find	Replace
JNDI_NAME parameter	`private static final String JNDI_NAME = "statelessSession.TraderHome";`	`private static final String JNDI_NAME = "TraderHome";`
`main()`	`String url = "t3://localhost:7001";`	`String url = "iiop://localhost:900";`
`getInitialContext()`	`h.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");`	`h.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");`

The client is now ready.

Deploying the Bean

Precondition

First, you have to start the WebSphere server:
- On the Start | Settings choose *Control panel* | *Services* command.
- Select *IBM WS AdminServer* and click *Start* button.

Note: *IBM HTTP Administration* and *IBM HTTP Server* should be already started.

Deploying the bean

In Together, invoke *J2EE Deployment Expert* from the *Tools* menu. Select *IBM WebSphere AE 3.5* as the target server and make sure that all available checkboxes, except *Process servlets*, *Generate WLM jar for Bean*, *Generate start and compile file for client*, *Generate simple JSP client* and *Generate command file for deployment* are set.

On the next page, set paths to WebSphere home directory and to IBM JDK 1.2.2 root directory (e.g. `c:\WebSphere\AppServer`" and "`c:\WebSphere\AppServer\jdk`). Press Next.

Set the following parameters on the next page:

Parameter name	Setting	Comment
Admin Node Name	Name of the Node to deploy	Name of the computer
Dependent ClassPath	Fully qualified path to WebSphere	Check if it refers to existing jars
Application Server Name	Your Application Server	"Default Server" by default
EJB Container name	Your Container name	"Default Container" by default
Target WebSphere server directory	WebSphere home directory	e.g. `c:\WebSphere\AppServer`
Stop & Remove Beans(s), Servlet(s) before deployment	ON	
StartBeans(s), Servlet(s) before deployment	ON	
Launch server in debug mode	OFF	

Press Next and Finish.

If you have selected Hot Deploy option, you will see the message in the Message Pane, which means that the deployment process successfully completed:

```
//WAS35: Finished with 0 Errors, 0 Warnings.
```

Now let us create a client for this bean.

Compiling and running the client

Set JDK Home to `%WAS_Home%\AppServer\jdk` on the *Run/Debug* tab of the *Options | Project* dialog.

For the 'Client' class on the diagram, choose Rebuild Node on its speedmenu. The Builder pane displays a message that the tool is completed.

To run the application, choose *Tools | Run/Debug | Run*. Make sure that *Class with main* is set to `examples.ejb.basic.containerManaged.Client`. The following messages appear in the Runner pane:

```
Beginning statelessSession.Client...

Creating a trader
Buying 100 shares of BEAS.
Buying 200 shares of MSFT.
Buying 300 shares of AMZN.
Buying 400 shares of HWP.
Selling 100 shares of BEAS.
Selling 200 shares of MSFT.
Selling 300 shares of AMZN.
Selling 400 shares of HWP.
Removing the trader

End statelessSession.Client...
```

Step By Step e-commerce: How To Create Web Application Diagram and Deploy it to an Application Server

This example shows how to use Web Application and Enterprise Application diagrams, based on the example supplied with Tomcat.

Creating Project

You can find this example in
`%TG_HOME%\bundled\tomcat\webapps\examples.war.`

Creating a Session Bean

Creating source directory structure

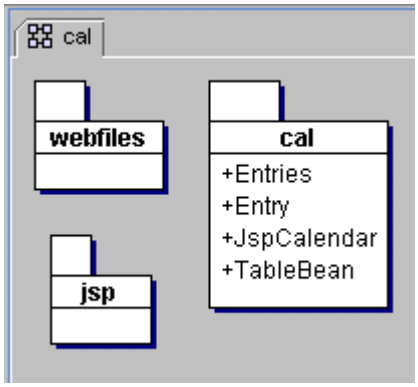
First, you have to create the source code directory structure. Extract the archive file `%TG_HOME%\bundled\tomcat\webapps\examples.war` to your work directory, for example, to `c:\temp\examples` (%EXAMPLES% macro is used to designate this path). Next, create a directory structure, where the source files will be placed:
`%TG_HOME%\myprojects\cal`
`%TG_HOME%\myprojects\cal\cal` for the java files.
`%TG_HOME%\myprojects\cal\jsp` for the jsp files.
`%TG_HOME%\myprojects\cal\webfiles` for the non-jsp files.

Next you have to copy all the necessary files to these directories. All jsp files should be copied from `%EXAMPLES%\jsp\cal to %TG_HOME%\myprojects\cal\jsp`. None-jsp files should be copied from `%EXAMPLES%\jsp\cal to %TG_HOME%\myprojects\cal\webfiles`. All java files should be copied from `%EXAMPLES%\WEB-INF\classes\cal to %TG_HOME%\myprojects\cal\cal`.

When the source directories are ready, start Together and create New Project (*Main Menu | File | New Project*) with the name *cal*. Specify location of the project as `%TG_HOME%\myprojects\cal`.

Make sure that `%TG_HOME%\bundled\tomcat\lib\servlet.jar` is added to the Search/Classpath of your project. It is important for jsp running and debugging.

Together displays the default project diagram:

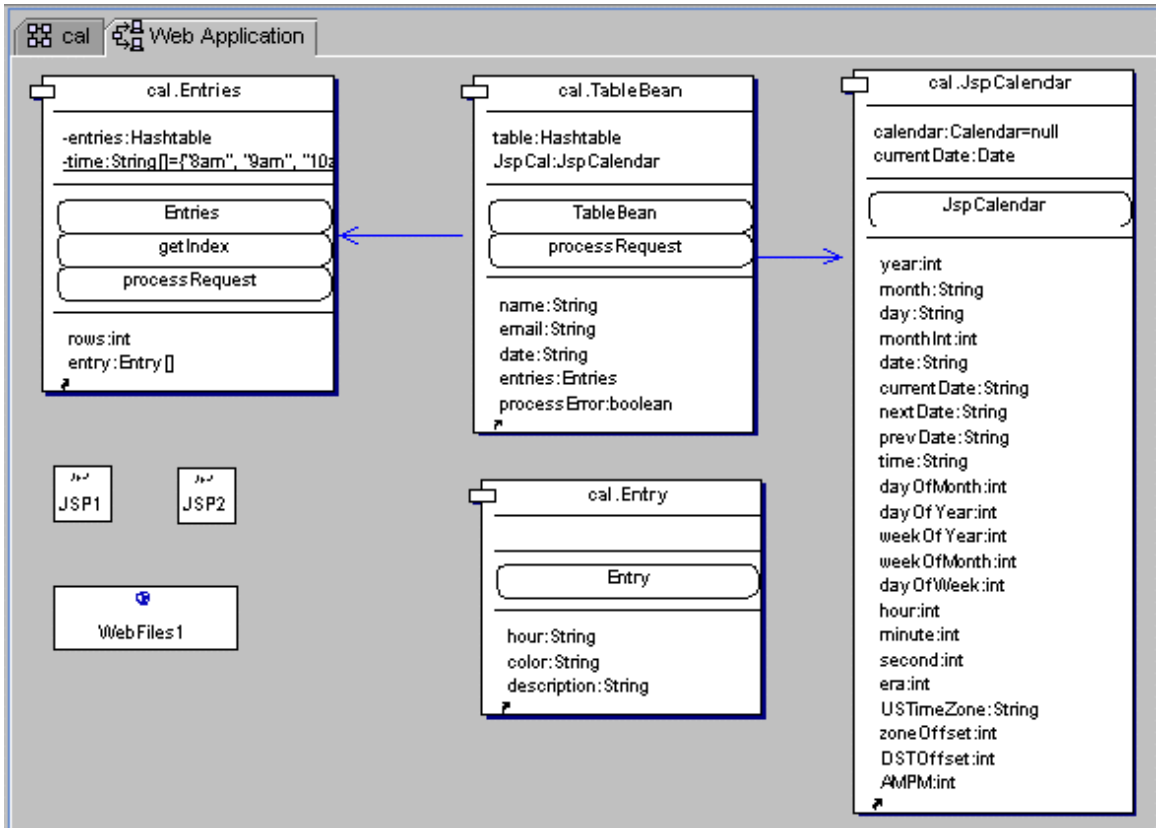


Creating Web Application and Enterprise Application diagrams

Create Web Application diagram (*File* | *New Diagram* | *Together* tab | *Web Application*). Set *Context Root* property of this diagram to *calendar*.

Add shortcuts to all classes from *cal* package to the diagram. Create two JSPs and one WebFile elements on the diagram.

Provide source codes for the created JSP and WebFile components. In the inspector, set *JSP Source* for JSP elements to `%TG_HOME%\myprojects\cal\jsp\cal1.jsp` and to `%TG_HOME%\myprojects\cal\jsp\cal2.jsp` respectively. Set *WebFiles source directory* to `%TG_HOME%\myprojects\cal\webfiles`.



Create Enterprise Application diagram and add shortcut to the created Web Application diagram.

Now you are ready to deploy this application to the Server.

Preview of the Example in Tomcat

You can preview the created application using Tomcat, prior to actual deployment. To do that, open your Web Application diagram, click *Run* button  in the main toolbar (or use Ctrl+F5 hotkey), and observe Tomcat deploying your application and running it automatically.

Deploying the created application

Start J2EE server. In Together, choose *J2EE Deployment Expert* command on the Tools menu. Select *Sun EE Reference Implementation* as a target server and make sure that the following options are only selected on the first page of the expert:

Add libraries required for deployment to the current project's Search/Classpath
Compile the classes referenced in the currently selected diagram
Generate Deployment Descriptor(s)
Pack modules for deployment
Deploy to application server
Clear temporary folder before starting the deployment process

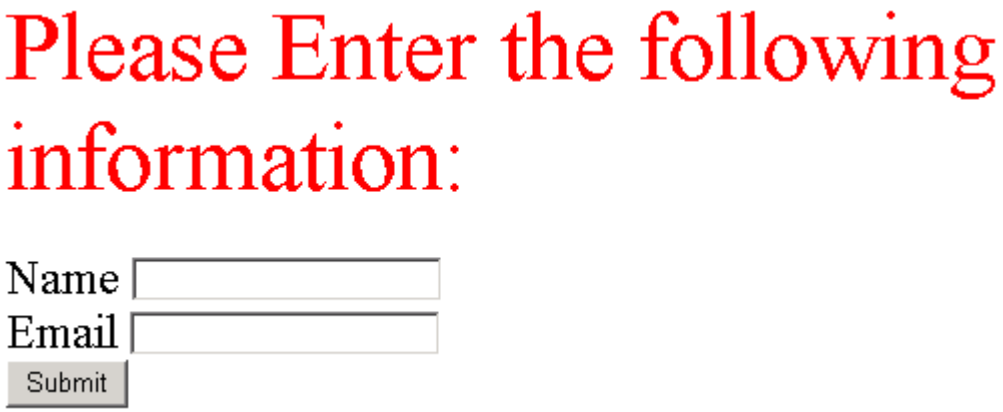
All other options should be cleared.

On the second page of the expert, specify correct paths to JDK 1.2.2 and to J2EESDK root directory (e.g. "c:\jdk1.2.2" and "c:\j2sdkee1.2.1").

On the next page of the expert, set 'localhost' as the host name. Press *Finish* to start deployment.

Running the application

Open your web browser and enter http://localhost:8000/calendar/login.html, Observe the result on the screen:



Note: See the sample in `%TG_HOME%\Samples\java\ecommerce\jsp\cal.`

Step By Step e-commerce: How To Create and Use MessageDriven Bean

This example shows how to create a message-driven bean in *Together* Class diagram and use it as an asynchronous message consumer.

To a client, a MessageDriven bean is a JMS message consumer that implements some business logic running on the server. The user accesses a message-driven bean through JMS by sending messages to the JMS Destination (*Queue* or *Topic*) for which the message-driven bean class is the MessageListener.

According to EJB 2.0 specification, MessageDriven bean has neither home nor remote interface. Therefore *Together* generates only a skeleton of its implementation class.

Here you can see a very simple example with one message-driven bean and one client for this EJB. The client sends messages to the Destination (Topic), for which the MessageDriven bean is the MessageListener. The MessageDriven bean initiates the output of the message (String) to the screen of display.

Creating a MessageDriven Bean

Create New Project and enter *MessageDriven* as the project name. As you won't need standard libraries, you can remove them from the project. Next, in the *EJB* tab choose *BEA WebLogic Application Server 6.0* and click *OK* to create the project.

Create New Package (using *Package* icon on the toolbar or *New | Package* command on the diagram speedmenu). Enter the name *messagedriven* and open in *New* Tab.

Click on *Message Driven EJB* icon and create MessageDriven bean in the *messagedriven* package. Enter the name *TogetherMessageDrivenBean*.

Set the following properties for this MessageDriven bean :

Destination Type to `javax.jms.Topic`

Destination JNDI Name to `TogetherTopic`

Choose `onMessage` method in the bean shape. In the Editor pane, add the following code to this method:

```
public void onMessage(javax.jms.Message msg) {
try {
TextMessage tm = (TextMessage)msg;
String text = tm.getText();
System.out.println("TogetherMessageBean : " +
text);
} catch (Exception ex) { ex.printStackTrace(); }
}
```

Add import statement `javax.jms.TextMessage` to the Java code generated for the MessageDriven bean (after `package messagedriven` statement and before class declaration)

Now you are ready to deploy this bean to BEA WebLogic 6.0

Deploying the MessageDriven Bean

Precondition

Prior to deploying the MessageDriven bean, create a JMS server in WebLogic Server. This requires adding the following statements to `config/mydomain/config.xml` file of your server in the <domain> tag:

```
<JMSServer
Name="TogetherJMSServer"
Targets="myserver"
>
<JMSTopic
JNDIName="TogetherTopic"
Name="TogetherTopic"
/>
</JMSServer>
```

Start BEA WebLogic Server 6.0

To do this, choose *Tools | EJB Deployment Expert*, and select *Start BEA Weblogic Application Server 6.0*.

Deploying to working server

To do this, choose *Tools | J2EE Deployment Expert,* and select *BEA WebLogic Application Server 6.0* as a target server. Make sure that the libraries required for deployment are added to the project Search/Classpath, and the following flags are checked on the first page of the expert:

- Compile the classes referenced in the currently selected diagram
- Generate Deployment Descriptor(s)
- Hot deploy to server
- Clear temporary folder before starting the deployment process

Note: Other checkboxes should be unchecked.

Press *Next* to proceed to the second page of the Expert.

Set paths to jdk1.3 (e.g. c:\jdk1.3), to WebLogic (e.g. d:\bea\wlserver6.0) and paths to the destination folders for resulting jar file and temporary files. Click *Next* to proceed to the last page of the Expert.

At the last step, specify server host (localhost), system password that you've set for WebLogic (e.g. *together*) and port number (usually 7001). Click *Finish* to complete.

The Message Pane displays messages, the last two being:

```
//WLS60: Finished with 0 Errors, 0 Warnings.//
```

It means the deployment process completed successfully.

Creating the Client

In the *MessageDriven* project create New Package *Client* on the default diagram and open in new tab.

In the Client package tab, create main class using Class by Pattern command on the diagram speedmenu or a toolbar icon. Change name to *Client* and press *Finish*. In the Editor pane, add necessary code to your class. After changes your class should look like:

```
/* Generated by Together */
package client;

import java.rmi.RemoteException;
import java.util.Properties;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;
import javax.ejb.CreateException;
import javax.ejb.RemoveException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class Client {

static private String TOPIC_NAME = "TogetherTopic";
private String myUrl = "t3://localhost:7001";
private Context myContext;
private TopicConnection myTopicConnection;

public Client() throws NamingException {

try {

myContext = getInitialContext();

// Create the connection and start it

TopicConnectionFactory conFactory = (TopicConnectionFactory)
myContext.lookup("javax.jms.TopicConnectionFactory");

myTopicConnection = conFactory.createTopicConnection();

myTopicConnection.start();

} catch(Exception ex) { ex.printStackTrace(); } }

public static void main(String[] args) throws Exception {

Client client = null;

try {

client = new Client();

} catch (NamingException ne) { System.exit(1); }

try {

client.sendMessage("Together ControlCenter supports EJB2.0
and Message-Driven beans.");
```

```
} catch (Exception e) { log("Client failed to log remotely:
"+e); } }
public void sendMessage(String message) throws
RemoteException, JMSException, NamingException {
Topic newTopic = null;
TopicSession session = null;
try {
log("creating topic session");
session = myTopicConnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
log("looking up topic in JNDI : "+TOPIC_NAME);
newTopic = (Topic) myContext.lookup(TOPIC_NAME);
} catch(NamingException ex) {
log("topic : "+TOPIC_NAME+" not found in JNDI.
Creating...");
newTopic = session.createTopic(TOPIC_NAME);
myContext.bind(TOPIC_NAME, newTopic);
log("topic : "+TOPIC_NAME+" bound to JNDI successfully.");
}
log("creating TopicPublisher");
TopicPublisher sender = session.createPublisher(newTopic);
log("creating TestMessage");
TextMessage tm = session.createTextMessage();
tm.setText(message);
log("sending message to JMS Destination");
sender.publish(tm);
}
private Context getInitialContext() throws NamingException {
try {
Properties h = new Properties();
h.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
h.put(Context.PROVIDER_URL, myUrl);
return new InitialContext(h);
} catch (NamingException ex) {
log("Error connecting to server.");
```

```
ex.printStackTrace(); throw ex;
}
}
private static void log(String s) {
System.out.println(s);
}
}
```

Now you are ready to compile and run the client

Compiling and running the Client

Add the following libraries to the project: `%WL_HOME%\lib\weblogic.jar` and home directory of WebLogic Server `%WL_HOME%` (e.g. `c:\wlserver6.0`). To do this, choose *File | Project Properties* on the main menu and select *Search/Classpath* page on the *Resource* pane of the Project Properties dialog. Make sure that the classpath is empty.

Press *Add Path or Archive* button and add `%WL_HOME%\lib\weblogic.jar`. This resource should be first in the list of available resources. Click *OK* to save the project properties.

Note: weblogic.jar should be the first in the list of libraries.

Compiling

On the speedmenu of the Client class select *Rebuild Node* command and observe the message notifying that rebuild is completed in the Message pane.

Running

With the Client class selected, choose *Tools | Run/Debug|Run* on the main menu and observe the following events:

- creating topic session
- looking up topic in JNDI : TogetherTopic
- creating TopicPublisher
- creating TestMessage
- sending message to JMS Destination
- message in the message pane and "TogetherMessageBean : Together ControlCenter supports EJB2.0 and Message-Driven beans." in the server console..

You can find this example in the `%TGH%\Samples\java\EJB20\messagedriven.tpr`'.

How to Use Taglibs in a Web Application

In this topic we shall consider a sample project that demonstrates usage of the Taglib diagram for a web application development.

Creating a tag library

Create a new project with the name *exampletag*. Next, create *tags* package, open it and create a new TagLib diagram. Add a tag *Extends TagSupport* and change its name to *SimpleTag*.

Defining tag library properties

In order to identify the tag library we have to create its descriptor file. In the diagram inspector open *TagLib Properties* page and set property *TLD File Name* to `$PRJ_DIR$\SimpleTagTLD.tld`. This file doesn't exist and will be created after user's confirmation.

Accept default values for the other properties.

Editing the taglib class

In the object inspector of the taglib class open *TagLib Properties* page and set the following properties:

```
BodyContent = EMPTY
Tag Name = tcc_tag
```

Provide contents for `doStartTag()` method. In the Editor pane enter the following code in place of "Write your code here..."

```
try {
JspWriter out = pageContext.getOut();
out.print("Together's simple tag.");
} catch(IOException e) {
System.out.println("Exception in SimpleTag: " + e);
}
```

Creating a Web Application diagram

Create a Web Application diagram. In the diagram inspector open *Web Properties* page and set *Module name* property to `TaglibWebApplication`.

Now the taglib should be added to the Web Application diagram. Choose *TagLib* button on the diagram toolbar and click on the diagram pane. This invokes *Select TagLib diagram* dialog in the form of Selection manager. Expand the Model node and choose TagLib diagram.

Creating a JSP

Add a JSP element to the Web Application diagram and change its name to *SimpleTagExample*. In the *JSP Properties* page of the object inspector set *JSP Source* as `$PRJ_DIR$\SimpleTagExample.jsp`. This file doesn't exist and will be created after confirmation.

Next, open JSP element in the JSP editor and enter the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
```

```
<%@ taglib uri="SimpleTagTLD.tld" prefix="tcc" %>
<TITLE><tcc:tcc_tag /></TITLE>
<LINK REL=STYLESHEET
  HREF="JSP-Styles.css"
  TYPE="text/css">
</HEAD>
<BODY>
<H1><tcc:tcc_tag /></H1>
<tcc:tcc_tag />
</BODY>
</HTML>
```

Deploying Web Application to BEA Weblogic Application Server 6.0

Now as the web application is ready, let us deploy it to WebLogic Server. Start J2EE Deployment Expert and choose *BEA Weblogic Application Server 6.0*

Make sure to set the following flags on the first page of the expert:

-Add libraries required for deployment to the current project's Search/Classpath
-Compile the classes referenced in the currently selected diagram
-Generate Deployment Descriptor(s)
-Pack modules for deployment
-Hot Deploy to server (BEA Weblogic Application Server 6.0 should be started)
-Clear temporary folder before starting the deployment process

On the second page of the expert specify paths to WebLogic Application Server 6.0 and to jdk1.3 and click *Finish* to complete deployment.

To observe the result, open your registered browser and enter `http://localhost:7001/TaglibWebApplication/SimpleTagExample.jsp`.

You should see

Together's simple tag.

How to Debug EJB's in IBM WebSphere 3.5

EJB's debugging in IBM WebSphere 3.5 is a rather complicated process that requires preliminary preparation.

> **Note**: Do not use debugging without urgent necessity, since it results in essential delay of IBM WebSphere's work.

Installing JPDA for IBM WebSphere 3.5

To use IBM WebSphere 3.5 in the debugging mode, install Java Platform Debugger Architecture (JPDA) extension for JDK 1.2.2 in such a way that the IBM WebSphere 3.5 server can be launched for remote debugging from the command line.

JPDA installation leaves the existing IBM JDK intact and uses a modified batch file to launch the WebSphere Admin Server in a 'jpda enabled' mode.

Install JDK 1.2.2

IBM WebSphere 3.5 is installed with JDK directory `%WS_HOME%\AppServer\jdk.` Create `%WS_HOME%\AppServer\jdk`1.2.2 directory and install JDK 1.2.2 in it:

Only the program files need to be installed. Thus you have to check *Program Files* checkbox solely:

Do not install *Java 2 Runtime Environment:*

Install JPDA Extensions

Unzip the `jpda1_0-win.zip` into `%WS_HOME%\AppServer\jdk1.2.2\jre`

Copy Files from IBM JDK

Copy the following, overwriting any existing files:

`%WS_HOME%\AppServer\jdk\jre\lib\ext\*.jar` to
`%WS_HOME%\AppServer\jdk1.2.2\jre\lib\ext\*.jar`

`%WS_HOME%\AppServer\jdk\jre\lib\orb.properties` to
`%WS_HOME%\AppServer\jdk1.2.2\jre\lib\orb.properties`

`%WS_HOME%\AppServer\jdk\jre\bin\ioser12.dll` to
`%WS_HOME%\AppServer\jdk1.2.2\jre\bin\ioser12.dll`

`%WS_HOME%\AppServer\jdk\lib\tools.jar` to
`%WS_HOME%\AppServer\jdk1.2.2\lib\tools.jar`

`%WS_HOME%\AppServer\jdk\bin\rmic.exe` to
`%WS_HOME%\AppServer\jdk1.2.2\bin\rmic.exe`

Create batch files

Copy

`%WS_HOME%\AppServer\bin\setupCmdLine.bat`

to

`%WS_HOME%\AppServer\bin\setupJPDACmdLine.bat`

Edit `setupJPDACmdLine.bat` and change the line

```
SET JAVA_HOME=%WS_HOME%\AppServer\jdk
```

to

```
SET JAVA_HOME=%WS_HOME%\AppServer\jdk1.2.2
```

Copy

```
%WS_HOME%\AppServer\bin\debug\adminServer.bat
```

to

```
%WS_HOME%\AppServer\bin\debug\jpda_adminServer.bat
```

Edit `jpda_adminServer.bat` and change the line

```
call ...\setupCmdLine.bat to call ...\setupJPDACmdLine.bat
```

Change the line

```
%JAVA_HOME%\bin\java -DDER_DRIVER_PATH=%DER_DRIVER_PATH% -Xmx128m -
Xminf0.15 -Xmaxf0.25 com.ibm.ejs.sm.server.AdminServer -bootFile
%WAS_HOME%\bin\admin.config %restart% %1 %2 %3 %4
```

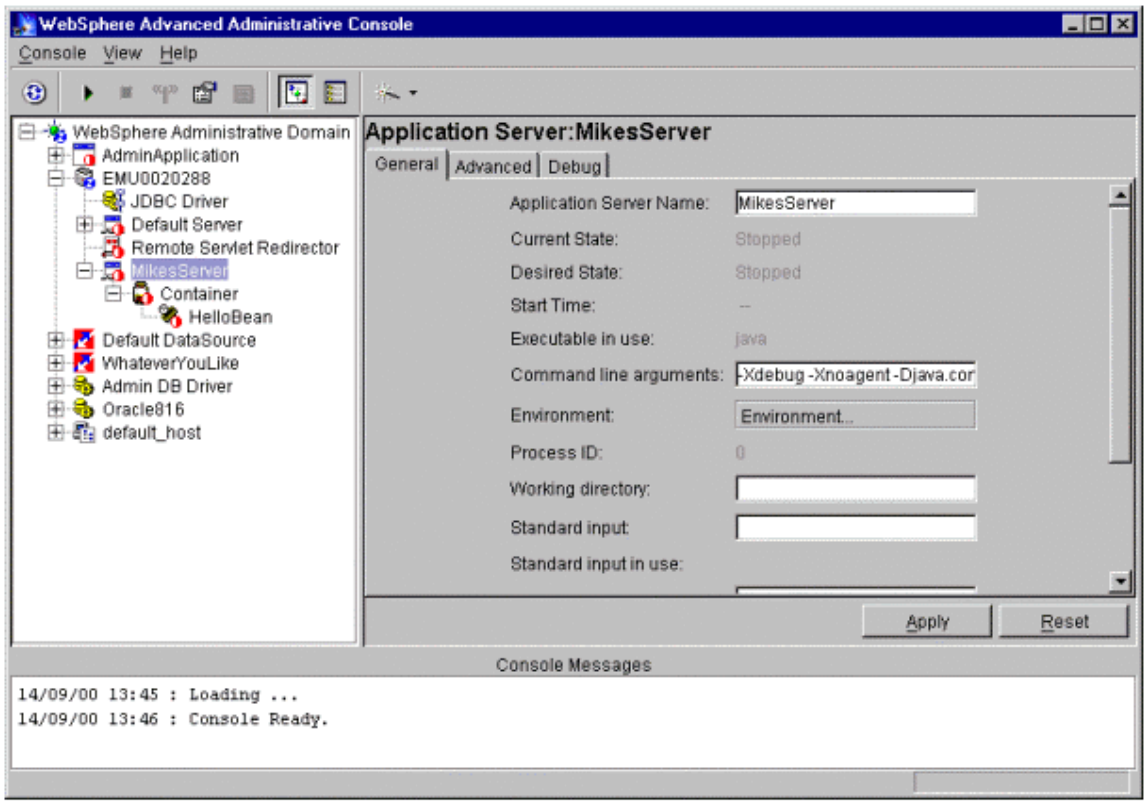
to

```
%JAVA_HOME%\bin\java -Djava.compiler=NONE -
DDER_DRIVER_PATH=%DER_DRIVER_PATH% -Xmx128m -Xminf0.15 -Xmaxf0.25
com.ibm.ejs.sm.server.AdminServer -bootFile
%WAS_HOME%\bin\admin.config %restart% %1 %2 %3 %4
```

To debug a running server use the following command line arguments:-

-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,address=8787,suspend=n

Check this in Avanced Administrative Websphere Console:



Note: Command line arguments text field should be empty, if IBM WebSphere 3.5 is started in the normal mode. So, clean this line after using IBM WebSphere 3.5 in the debugging mode.

The IBM Websphere server uses the 8787 port number for remote debugging by default. Make sure that any other server, which you wish to debug (a servlet container, for example), uses a different port number.

The Together's deployer automatically puts these parameters into the server attributes. Now you are ready to deploy and debug EJBs.

Debugging "Hello World" EJB sample

Start the IBM Websphere 3.5 service in debugging mode by running `jpda_adminServer.bat`.

Open the sample project `HelloWorld.tpr` under `%TOGETHER_HOME%/Samples/java/ejb/WebSphere` and start up "Hello World" EJB from Together, or using WebSphere console directly.

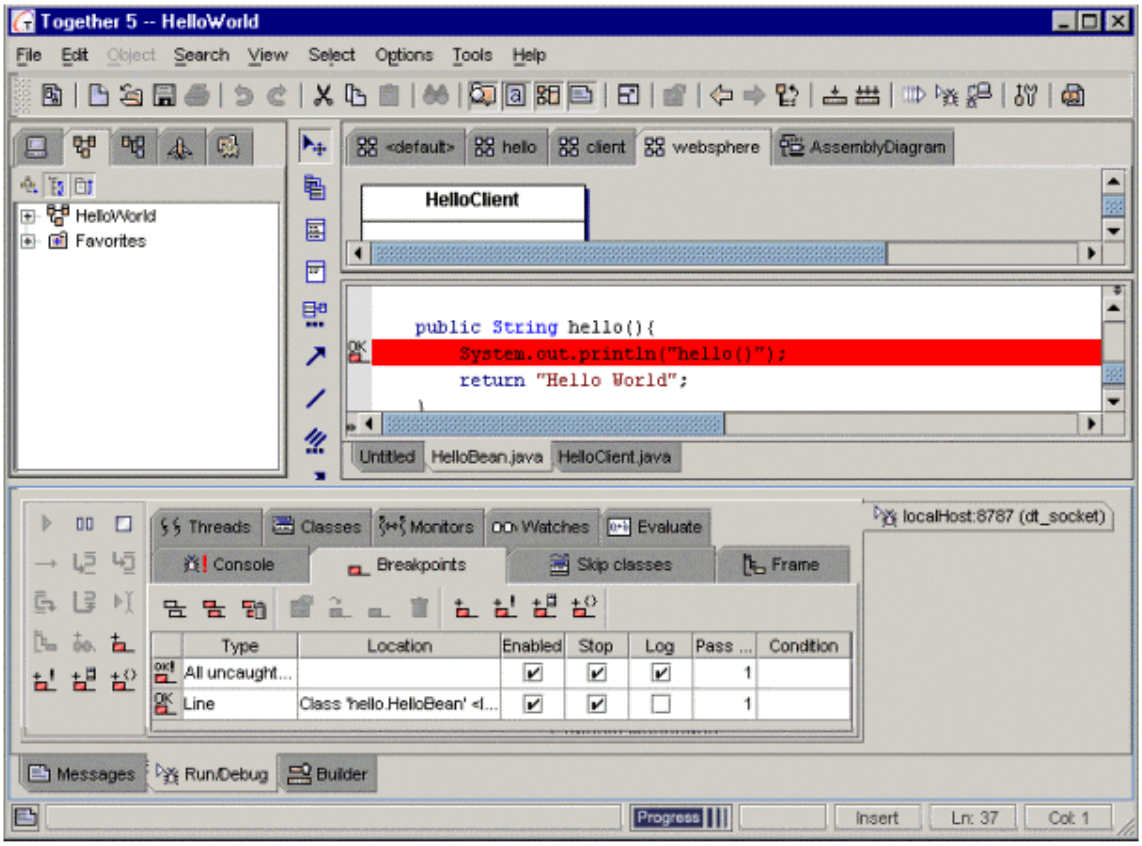
After deployment you have to point the deployed jar in *Project Options* | *Builder* | *Compiler options* | *Classpath* and to change the working JDK to IBM WebSphere JDK in *Project Options* | *RunDebug* | *JDK Home* edited field. Check that `%TOGETHER_HOME%/Samples/java/ejb/WebSphere` path is set in *Path to Sources* field of WebSphere Advanced Administrative Console.

If this path is not set, stop the application server, set the necessary path in *Path to Sources* field, click *Apply* and restart the server.

Set a breakpoint, for example, to the business method hello() in HelloBean.java source file:

```
    public String hello(){
        System.out.println("hello()");
        return "Hello World";
    }
```

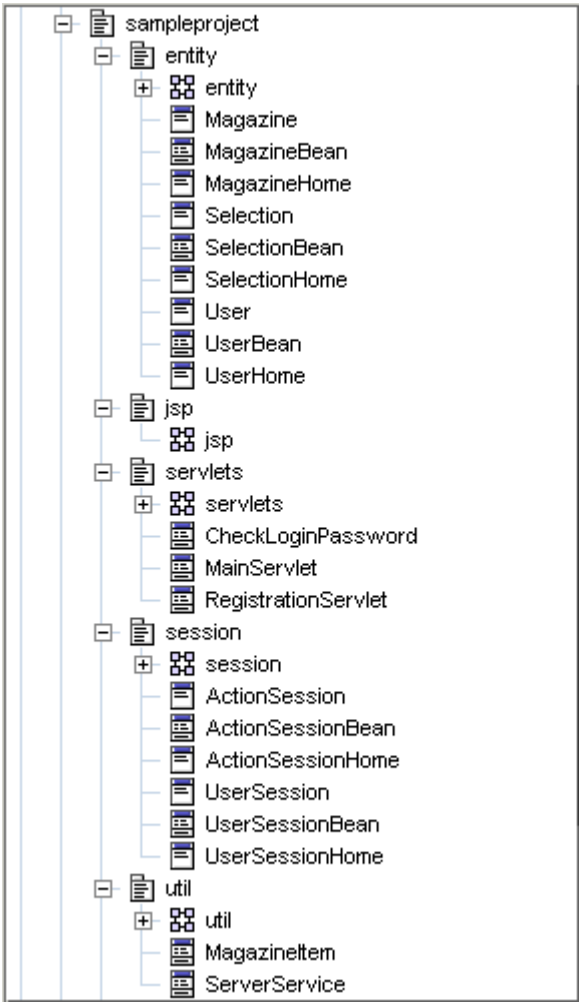
Then click *Attach to Remote Process* command and use the default settings:



Run the client application in the normal (not debugging) mode and wait, when the breakpoint becomes active.

J2EE Step by Step

This section will take you through the process of creating , deploying and running an application
using EJB Assembler, Web Application and Enterprise Application diagrams.
You can find the source files for this example in the
`%TGH%/Samples/java/ecommerce/MagazineSrc.zip` ,
create a project around them, and try it out for training purposes.
The application consists of the existing CMP Entity beans, Session beans, servlets, JSPs and utility classes:



Creating the Application

We have to create a project with three diagrams: EJB Assembler diagram, Web Application diagram, and Enterprise Application diagram

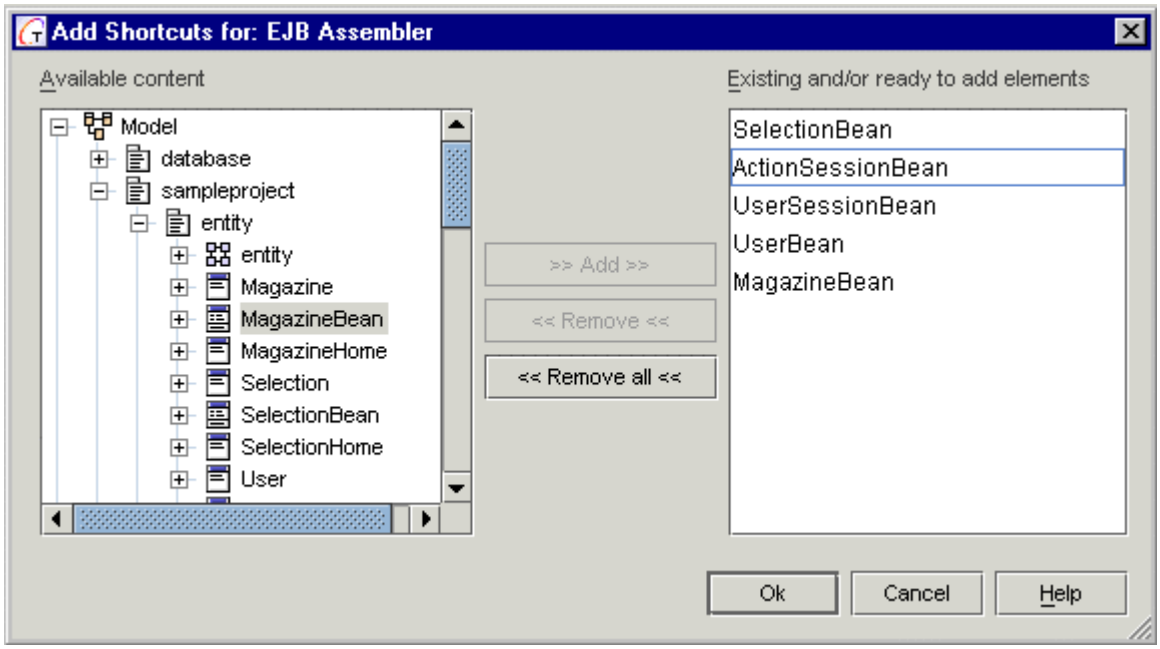
Creating the Project

Start Together and create a new project with the name *Magazine* around the code located at `%TG_HOME%\Samples\java\ecommerce\Magazine`. Since the standard libraries are not required, you can remove them from the project (in the Advanced mode, choose

Search / Classpath tab and make sure that checkboxes 'Include standard libraries' and 'Include Classpath' are unchecked). In the *EJB* tab, choose *BEA WebLogic Application Server 6.0*.

EJB Assembler diagram

Next, Create new EJB Assembler diagram (*New Diagram | Together | EJB Assembler*). Add shortcuts to the EJBs listed above. To do that, choose *Add Shortcut* command on the diagram speedmenu and select *SelectionBean, ActionSessionBean, UserSessionBean, UserBean, MagazineBean.*

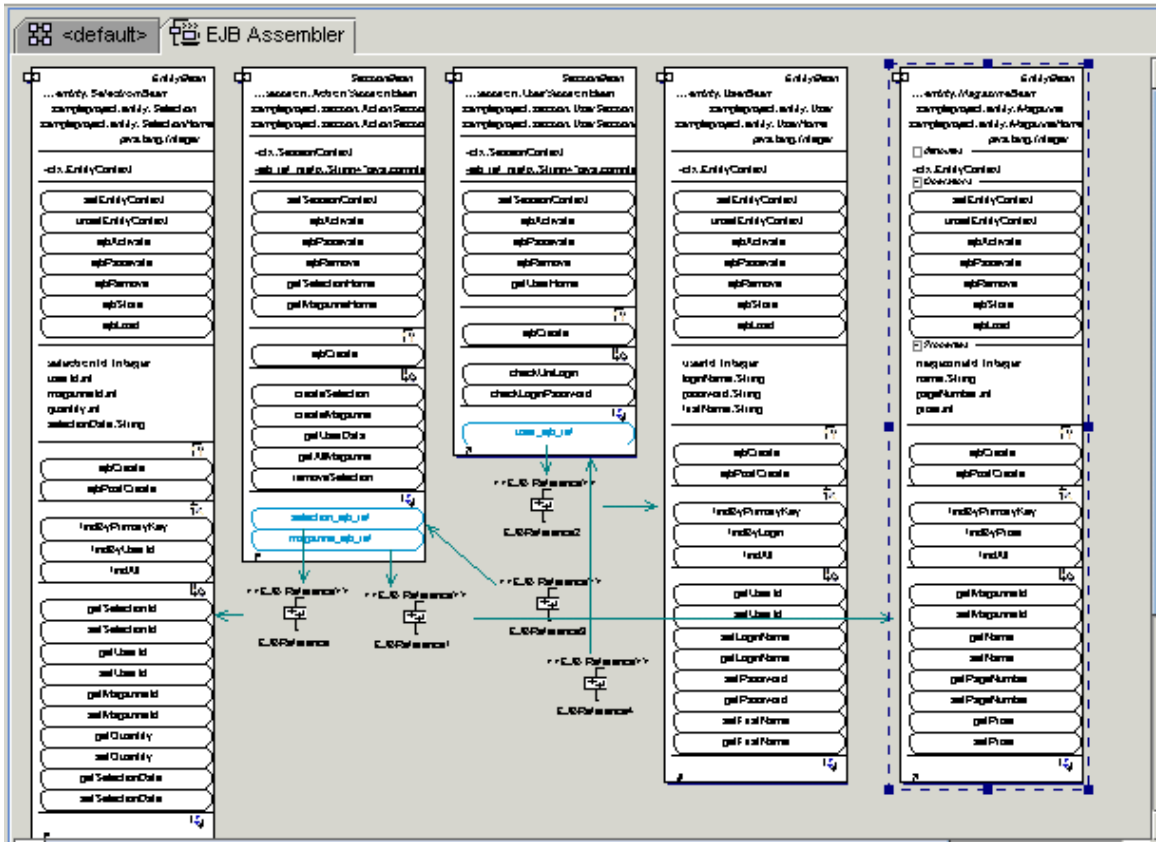


Observe the classes appearing on the diagram. Note that some of the classes contain EJB references. These references should be linked with the appropriate beans. To do that, create 5 EJBReference elements (select *EJBReference* icon on the tool bar and click on the diagram pane).

> **Tip:** Create one EJBReference element and choose Clone command on its speedmenu to produce four more instances.

Using *Assembly Link* icon  on the diagram toolbar, create assembly links between the following elements:

Source	Destination
ActionSessionBean / selection_ejb_ref	1st EJBReference element
1st EJBReference element	SelectionBean
ActionSessionBean / magazine_ejb_ref	2nd EJBReference element
2nd EJBReference element	MagazineBean
UserSessionBean / user_ejb_ref	3d EJBReference element
3d EJBReference element	UserBean
4th EJBReference element	ActionSessionBean
5th EJBReference element	UserSessionBean



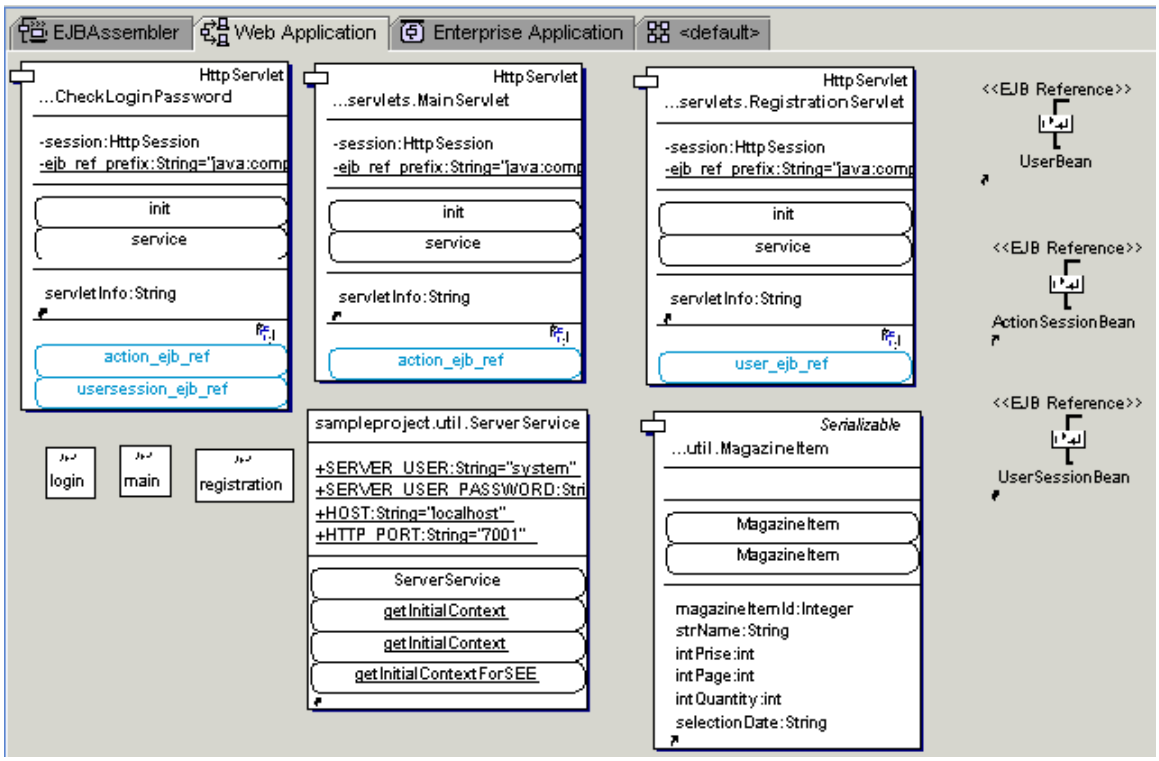
Web Application diagram

Create a new Web Application diagram (*New Diagram | Together | Web Application*) and add three JSP elements with the names *login, main* and *registration*.

Now you have to provide source code for the created visual components. To do that, you have to edit the properties of these components in the Inspector. On the *JSP Properties* tab of the inspector, enter the following values in the *JSP source* field:

JSP name	JSP Source
login	`%PRJ_DIR%\sampleproject\jsp\index.jsp,`
main	`%PRJ_DIR%\sampleproject\jsp\main.jsp`
registration	`%PRJ_DIR%\sampleproject\jsp\registration.jsp`

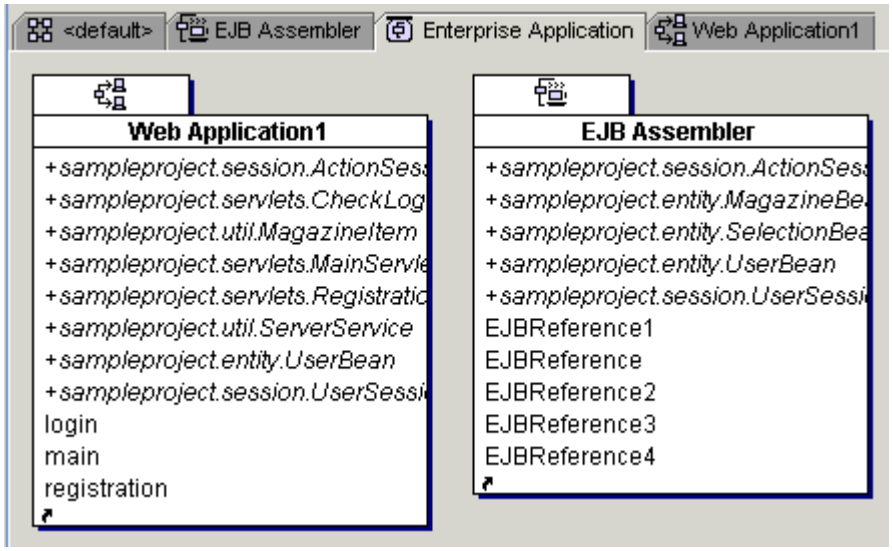
On the diagram pane, add shortcuts to all servlets, home and remote interfaces, utility classes, and to the EJBReference elements (*UserBean, ActionSessionBean, UserSessionBean*) from the EJB Assembler diagram.



Finally, set *Context Root* property of the Web Application diagram to *sample*.

Enterprise Application diagram

Create a New Enterprise Application diagram. Add shortcuts to the Web Application and EJB Assembler diagrams, using appropriate button on the diagram toolbar.



Now the application is ready to be deployed to BEA WebLogic Application Server 6.0

Deploying the Application

If you are working on a Windows platform, you will use ODBC data source. For the other platform, Cloudscape is used.

Before proceeding with the deployment, we have to create the database pool on the WebLogic server, and provide ODBC or Cloudscape data source.

Creating the database pool

To create a database pool, some editing of the `config.xml` file of your server is required:

```
<!--This is a connection pool for ODBC:-->

<JDBCConnectionPool

    CapacityIncrement="1"
    DriverName="sun.jdbc.odbc.JdbcOdbcDriver"
    InitialCapacity="1"
    MaxCapacity="2"
    Name="togetherPool"
    Properties="user=none;
    password=none;
    server=none"
    Targets="myserver"
    TestConnectionsOnRelease="true"
    TestConnectionsOnReserve="true"
    TestTableName="UserLogin"
    URL="jdbc:odbc:SampleBase"

/>

<!--This is a connection pool for Cloudscape-->

<JDBCConnectionPool

    CapacityIncrement="1"
    DriverName="COM.cloudscape.core.JDBCDriver"
    InitialCapacity="3"
    LoginDelaySeconds="5"
    MaxCapacity="10"
    Name="togetherPool"
    Properties="user=none;password=none;server=none"
    Targets="myserver"
    URL="jdbc:cloudscape:%PROJECT_DIR%/database/SampleBase"

/>

<JDBCTxDataSource

    JNDIName="TxTogetherPool"
```

```
    Name="TxTogetherPool"
    PoolName="togetherPool"
    Targets="myserver"

/>

<JDBCDataSource

    JNDIName="MyTogetherPool"
    Name="MyTogetherPool"
    PoolName="togetherPool"
    Targets="myserver"

/>
```

Here %PROJECT_DIR% refers to the absolute project folder path.
Note that the `Targets` property in xml files should be same as your WebLogic server name.

Creating OBDC datasource

This section refers to ODBC only. If you use Cloudscape, you can just skip it.
Now you have to provide an ODBC datasource, using the system Control Panel. Click *Start* and choose *Settings | Control Panel | ODBC DataSources*. Press *Add* in the opened dialog and choose *Microsoft Access driver*. Set DataSource name to *SampleBase*, click *Select* and choose `db.mdb`. Click *OK* to complete.

Setting connection parameters

Select `sampleproject.util.ServerService` class and make sure that user name, password and port number are specified properly.

Connection parameter	Default value
HOST	localhost
SERVER_USER	system
SERVER_USER_PASSWORD	together
HTTP_PORT	`7001`

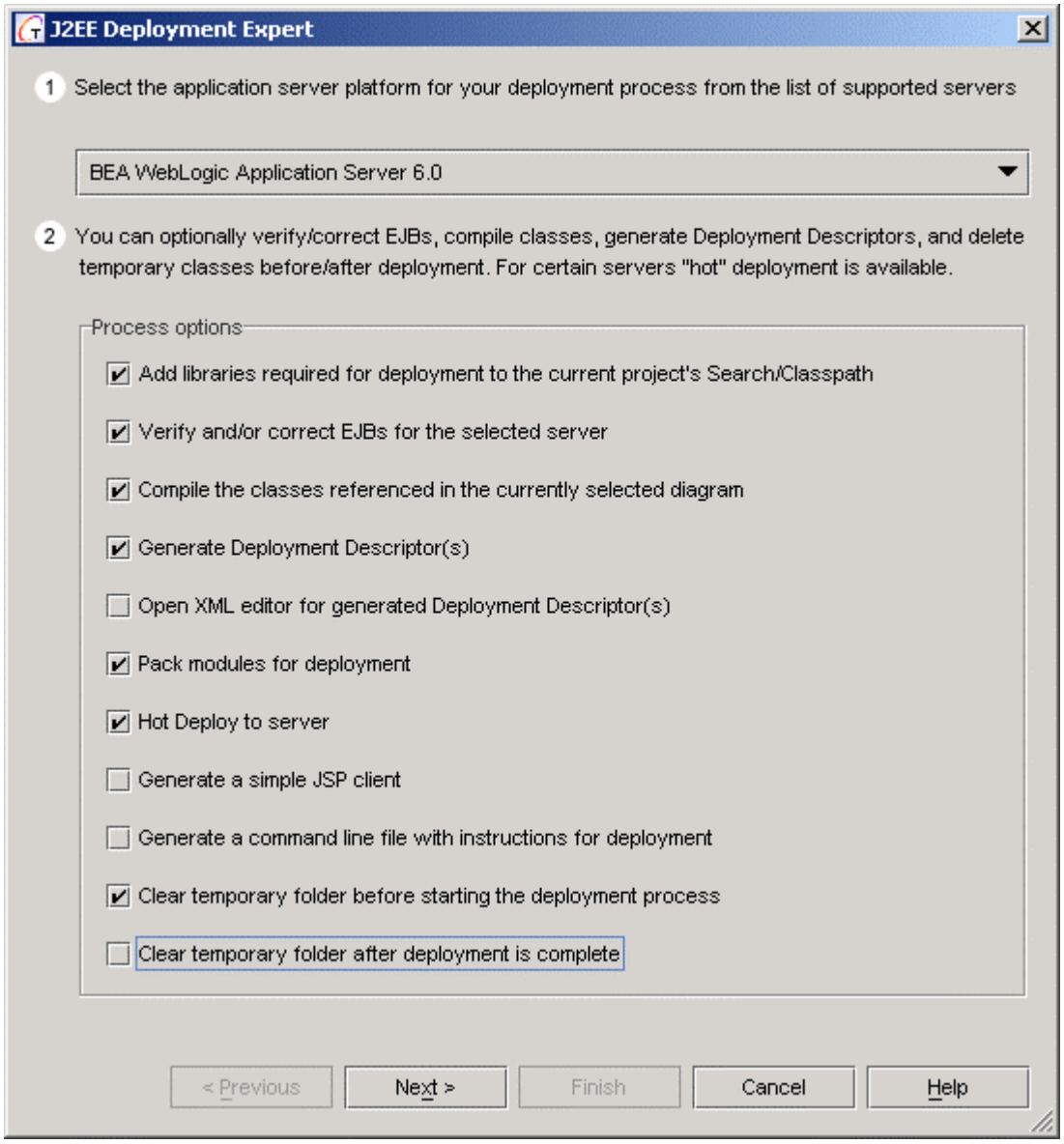
Deployment

First, we have to launch WebLogic Server 6.0 and set its root directory. Keep in mind that WebLogic 6.0 is delivered with the library `ejb20.jar` that provides J2EE support. However, by default this library is not specified in the launcher `startWeblogic.cmd`. Hence, you have to add this library to the classpath. Add the following line to the launcher:

```
set
CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar;.\samples\eval\clou
dscape\lib\cloudscape.jar;.\lib\ejb20.jar
```

On the *Together's* main menu, choose *Tools | J2EE Deployment Expert*. Select *Start BEA WebLogic Application Server 6.0* from the dropdown list. On the second page of the Expert, set the root directory of the WebLogic server using the File/Path Chooser button, and click *Finish* to complete.

Next, select *Enterprise Application diagram* to make it active, and choose *J2EE Deployment Expert* on the Tools menu. Again, choose the *BEA WebLogic Application Server 6.0* is the target server. Make sure to set the following flags only (all the other flags should be unchecked):



On the next page of the Expert, set paths to jdk1.3 (e.g. c:\jdk1.3), WebLogic (e.g. d:\bea\wlserver6.0) and paths to the resulting jar file and temporary files. Click *Next*.

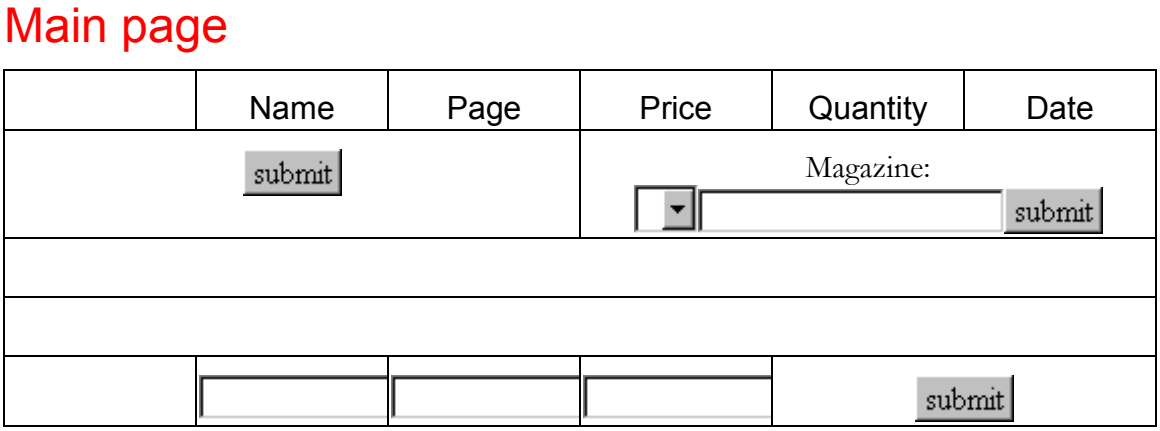
On the last page of the Expert, set server host (localhost), system password that you've set for Weblogic (e.g. together) and port (usually it's 7001), and click *Finish* to complete.

If 'Hot Deploy' checkbox was checked, and the deployment process successfully completed, the Message Pane displays:
//WLS60: Finished with 0 Errors, 0 Warnings.//
Now we are ready to compile and run the client.

Running the Application

Open your browser on `http://localhost:7001/sample/index.jsp`. The following screen shows up:

Enter *together* as a login and password. Click *login*. This will bring you to the Main Page:



On this page you can add magazines to the magazines' list (Add Magazine), or add/remove magazines that the user wants to buy.

Extensibility and Advanced Customization

Together Open API

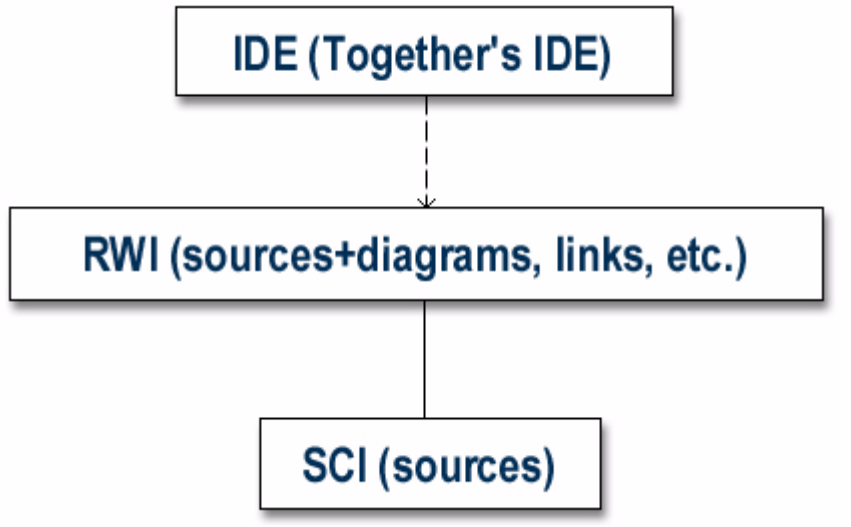
Together is highly extensible through an open Java API. You can extend Together's capabilities by developing Building Blocks that plug into the Together Platform as *modules*. Some of Together's features... Rose Import/Export and Documentation Generation, for example... are implemented as Building Block modules via the API (these are called System modules). You can view and run modules using the *Modules* tab of the Explorer.

This powerful API delivers the capability to externally use Together's internal functionality. The three main groups composing API are :

`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 API is composed of a three-tier interface that enables varying degrees of access to the native infrastructure. The top tier represents the highest degree of constraint and the lowest tier the least degree of constraint. The interfaces are very simply named:

IDE
Read-Write Interface (RWI)
Source Code Interface (SCI)



Basic *Together* API architecture

IDE

This is the API you need in order to generate 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 not change the model (accidentally or otherwise). IDE group provides the functionality related to the model's representation in Together's IDE and interaction with the user. Each package composing the IDE group has a description highlighting the areas of applicability of this specific package.

RWI

This API enables you to go deeper into the *Together* architecture. You can both extract information from, and write information to your models, and you can do some extensions of *Together*'s capabilities. RwiElements can represent more than packages, classes and members. In a RWI model they may represent different diagrams (class diagrams, use case diagrams, sequence diagrams and others), links, notes, use cases, actors, states etc.

SCI

As the name implies, the Source Code Interface takes you down to the source code level. This API is the most granular enabling even manipulation of single bytes. An SCI model is a set of sources (for Java, .class files are allowed) organized into packages. The SCI packages represent the Java packages (which can be stored even in .zip or .jar files) or directories for other languages. SCI model can contain parts written in different languages.

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++.

API Technical reference

Complete technical reference documentation for the Together API is provided separately from Help documentation. See **`$TOGETHER_HOME$/doc/api/index.html`**. To learn how to use Open API documentation, refer to **`$TOGETHER_HOME$/doc/api/help-doc.html`**.

See also

Working with Modules

Extension Modules

Together is highly extensible by means of an open Java API that enables you to write Java programs that use model information from Together and/or interact with Together itself to extend its native capabilities. Such programs are called the *building blocks* or *modules*. Many of Together's own features are implemented this way, and the architecture makes it possible for Together to include modules developed by strategic partners or other third parties.

Module development can take two forms:

Compiled: Written in Java and compiled using a Java compiler. Such modules use the same JVM as Together at runtime.

Written in TCL (or JPython): Such modules, referred to as *scripts* are interpreted by appropriate Together subsystems at runtime.

Compiled modules deliver better performance and more depth because you have the full capabilities of the Java language at your disposal. This topic addresses the basic concepts of module development.

Types of Modules

All modules are defined by their language, invocation time and type of deployment. By the time of invocation, the module are categorized into the following groups: User, User with pre-initialization, Startup, Activatable (OnDemand).

User module is added to the modules tree as an icon and can be invoked later from a popup menu (Run) of this icon.

User with pre-initialization is the same as "User", but differs with an additional feature that initialization method of such module executes before the first run of the module. This type makes sense for JAVA modules only and is assigned by default.

Startup modules are not added to the modules tree and always run on Together startup.

Activatable (OnDemand) module combines the advantages of both User and Startup types. Activatable modules are added both to the modules tree and to the list in Options|Activatable Modules menu.

To change Activatable module state, one can use main menu instead of browsing modules tree. Main menu command **Options->Activatable Modules** contains submenu where all existing Activatable modules are presented by checkboxes. State of a checkbox reflects the current state of an Activatable module. Checking a box activates a module, unchecking makes it deactivated.

State of an activatable module is persistent between Together sessions. Being activated, such module behaves as a startup module. By default activatable modules are deactivated.

Interfaces implemented by the Modules

Modules are Java classes that implement the `IdeScript, IdeStartup,`or `IdeActivatable` interfaces.

Modules of the User type implement `IdeScript` interface that provides the method `run()`. Startup modules implement the interface `IdeStartup` that provides `autorun()` method. The modules that should provide the possibility of on-demand invocation and deactivation, implement the interface `IdeActivatable` that extends `IdeStartup` with shutdown method. If a module is supposed to be used both as startup

and user module, it should implement `IdeScript` and `IdeStartup` interfaces, while startup module with the possibility of on-demand invocation, should implement both `IdeScript` and `IdeActivatable` interfaces.

All these interfaces are provided in the Together API.

Viewing and running Modules

Viewing modules

Modules are stored in the subdirectories under `$TOGETHER_HOME$\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 "in the works"...either not fully implemented, undocumented, or both.

The following table shows how modules are represented in the Modules tab.

	Java source file for a module. Can be compiled on the fly from speedmenu "Run" if a compiler is configured.
	Compiled Java module
	TCL script. "Run" executes the script in interpreted mode.

For more information about the Modules tab, see Explorer: Modules tab.

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' speedmenus.

You can run any module (or script) from the Modules tab speedmenu. For modules or scripts that refer to or handle model information (as most will), you should open a Together project before running.

To run a module (or script):

1. Navigate to and select the desired module or script
2. Right-click on the node and choose Run.

If you have defined a Java compiler in the Tools configuration options (Options | Default - Tools), you also can compile Java source code for a module "on the fly" when you choose *run* on the speedmenu.

You can also run modules using the command line interface. For more information, see Command Line Parameters.

For more information see Basic Guidelines for Developing Modules and Frequently Asked Questions.

Basic Guidelines for Developing Modules

You can write your own modules that access model information to generate model and code documentation in custom formats, export to different file formats, or develop patterns and experts.

If you create your own modules and save them to the relevant folders under `./modules/com/togethersoft/modules`. Newly created appear in the Modules tab and can run from there. You can also add commands for launching your own modules to the menu system by creating a Tool definition in the *Options* dialog (Options | Default | Tools), or by customizing the `./config/menu.config` and/or `./configaction.config` file.

Naming Conventions

Together's modules are classes implementing either `IdeScript` or `IdeStartup` interfaces, or both. The names of modules use mixed case: `GenerateDocumentaion`, `ImportFiles` etc.

Naming methods

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 a strong, active verb.

Examples:

```
processModel()
printCurrentDiagram()
recheckAllNames()
iterateCurrentNode()
```

This convention results in methods whose purpose can often be determined just by looking at the name. Although this convention results in a little extra typing by the developer, because it often results in longer names, this is more than made up for by the increased understandability of your code.

Getters

Getters are methods that return the value of an attribute. You should prefix the word 'get' to the name of the attribute, unless it's a Boolean attribute and then you prefix 'is' to the name of the attribute instead of 'get.'

Examples:

```
getFirstName()
getAccountNumber()
getLostEh()
isPersistent()
isAtEnd()
```

By following this naming convention you make it obvious that a method returns an attribute of an object, and for boolean getters you make it obvious that it returns true or false.

Setters

Setters are methods that modify the values of an attribute. You should prefix the word 'set' to the name of the attribute, regardless of the attribute type.

Examples:

```
setFirstName(String aName)
setAccountNumber(int anAccountNumber)
setReasonableGoals(Vector newGoals)
setPersistent(boolean isPersistent)
setAtEnd(boolean isAtEnd)
```

Following this naming convention you make it obvious that a method sets the value of an attribute of an object.

Naming attributes

You should use a full English descriptor to name your attributes to make it obvious what the attribute represents. Attributes that are collections, such as arrays or vectors, should be given names that are plural to indicate that they represent multiple values.

Documenting the module

Okay to use phrases instead of complete sentences, in the interests of brevity. This holds especially in the initial summary and in @param tag descriptions.

Use 3rd person (descriptive) rather than 2nd person (prescriptive). The description is in 3rd person declarative rather than 2nd person imperative.

```
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 description beyond the name. The best names are "self-documenting", meaning they tell you basically what the method does. If the doc comment merely repeats the method's name in sentence form, it is not providing more information. For example, if method description uses only the words that appear in the method name, then it is adding nothing at all to what you could infer. 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 - This 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

For now, a module defines its own subpackage with the name matching to name of the module, located in the `modules` package. For example:

```
package modules.InsertTags;

import com.togethersoft.openapi.ide.IdeContext;

import com.togethersoft.openapi.ide.IdeScript;

public class InsertTags implements IdeScript{

public void run(IdeContext context){

}

}
```

Modules FAQ

What is a module?

The module is a Java class that implements the **`IdeScript`** or the **`IdeStartup`** interface, or both. (You can find these interfaces in the `com.togethersoft.openapi.ide` package).

When implementing the `IdeScript` interface, you must define the `run(IdeContext)` method; when implementing `IdeStartup` interface, you must define the `autorun()` method.

What is the difference between these interfaces (and modules implementing them)?

The `IdeStartup` interface defines a module whose `autorun()` method will be invoked automatically during Together startup process. This method should perform 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 (com.togethersoft.openapi.ide.IdeContext)` instance being passed to the `run` method, contains information about selection at the moment you ran the module.

What's the difference between a module and a script?

The difference 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 and compiled using a Java compiler and executed using the same 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, using Java is recommended.

Where should my modules be located?

At this moment, a module defines its own subpackage with a name matching the name of the module, and 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){

}

}
```

How do I register a module?

- Compile a module. (Let's say you have compiled the `InsertTags` module shown above)
- Make a *manifest file* for your module. A manifest file is a simple text file which allows Together to identify the kind of a module.

The extension of a manifest file is `.def`, and the file name is the name of a module. It must be located in the classpath according to the package of the module. (For example, `InsertTags`'s manifest file is `New"%root%/modules/com/togethesoft/modules/inserttags/InsertTags.def`)

A manifest file consists of only one string:

For `IdeStartup` modules:

```
Startup=true
```

For `IdeScript` modules:

```
Script=true
```

(For example, since `InsertTags` is an `IdeScript`, it contains `Script=true`)

How do I run a module?

Startup modules implementing the `IdeStartup` interface: you can't run these manually. Together automatically invokes them during the session startup.

Modules implementing the `IdeScript` interface: use the modules tab of the Explorer pane. Navigate to the script, right-click on it, and choose *Run* from the speedmenu.

How about an example?

Here are two example modules for writing *Hello world;*.

Startup script. This script writes the message at Together's startup process.

```
package script.HelloWorldAutorun;

import com.togethersoft.openapi.ide.IdeStartup;

public class HelloWorldAutorun implements IdeStartup {

public void autorun(){

System.out.println("Hello world from HelloWorldAutorun
script!!!");

}

}
```

After compilation compose the file `%root%//script/HelloWorldAutorun/HelloWorldAutorun.def` with the line `Startup=true` in it.

Script implementing IdeScript interface.

```
package script.HelloWorld;
import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
public class HelloWorld implements IdeScript {
public void run(IdeContext context){
System.out.println("Hello World!!!");
}
}
```

After compilation compose the file `root%//script/HelloWorld/HelloWorld.def` with the line `Script=true` in it. Now open a project and locate this script in the *Modules* tab of the Explorer. Right-click on it and select *Run* command.

Module development "hands-on"

This section describes the full process of writing and deploying extension modules. We'll compose a simple module and show how to make Together recognize it. Finally, typical errors are outlined, and troubleshooting procedures described.

Let us develop a simple module, which can be used as a startup and as a regular module. This time we are not very interested in the module's functionality, so it will display just a short message, for the sake of simplicity.

There are two major steps in the module development. First, write the module source code. Second, declare this module. Module declaration and source code must comply. For example, if a module is declared OnDemand, but the main class of this module implements `IdeScript` interface, then this module will not work.

Source code for the module

Since we need to run the module as a startup and as a regular Together module, we must implement both interfaces: `com.togethersoft.openapi.ide.IdeScript` and `com.togethersoft.openapi.ide.IdeStart`

To output messages to the Together Message pane, we must 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` package.

The only thing left is the name of the module. Let's name it *MyFirstModule*. Below is the source code listing. If you want to try this out, create a directory

```
%TOGETHER_HOME%\modules\com\togethersoft\modules\myFirstModule
```

and save the following code as

```
MyFirstModule.java : //the MyFirstModule.java file

package com.togethersoft.modules.myFirstModule;

import com.togethersoft.openapi.ide.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 void run(IdeContext context){
```

```
//regular modules perform this method
IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMAT
ION,

"This is my first module called as a regular Together
module.");

}

public void autorun(){ //startup modules perform this method
IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMAT
ION,"

This is my first module. It was called as a startup
module.");

//this will appear in the Message pane when Together is
loaded

System.out.println("This is my first module, called as a
startup module.");

//this will appear in the console window at startup

}

}
```

Declaring a Module

If you run Together now and open Modules tab of the Explorer, you will see that *MyFirstModule* is not shown. This is because we haven't informed Together about what we're going to do with the module.

The most important thing that one should keep in mind writing a module is the set of properties that should be used for module declaration. Module declaration should cover module type (by invocation time), name, location in the modules tree, dependencies, additional libraries and other important properties. We do this through a text manifest file.

A manifest file is a **Manifest.mf** file or a file with the **.def** extension. It contains information about the usage of a module.

Declaring a module in Manifest.mf file

Below is the list of all properties available for the modules.

Main-Class (MainClassName)

This property is **mandatory** for all types of JAVA modules declared in Manifest.mf file (except for JAVA modules declared in the *.def files). Name in brackets is the alternative property name that should be used when module is declared in *.def file.

The class specified as a value of this property should exist and should implement Java interface that corresponds to its type (IdeScript, IdeStartup or IdeActivatable)

Example of property declaration in Manifest.mf file

```
Main-Class: com.togethersoft.modules.ejb.EJB
```

*Example of property declaration in *.def file:*

```
MainClassName=com.togethersoft.modules.helloworld.HelloJava
```

Name

This property specifies the module name to be used in the modules tree, in "Options->Activatable Modules" submenu (for "Activatable" modules) and in "Help->Modules" submenu (for HTML modules). Value of this property is case-sensitive.
Example:

```
Main-Class: com.togethersoft.modules.ejb.EJB

 Name: "EJB Support"
```

Or

```
Name: "XPTest Support"
HelpFile: index.html
```

Folder

This property specifies module location in the modules tree. It makes sense only for User, User with pre-initialization and Activatable module types. If this option is not specified, then Early Access folder is used by default. Value of this property is case-sensitive.
Example:

```
Name: "XPTest Export"

Folder: "System/XP/XPTest Export"
```

As a result of the declaration above, module XPTest will be found in the modules tree in All modules->System->XPTest and module node(s) will be named XPTest Export.

Time

This property defines module type by invocation time. It is **mandatory** for the JAVA modules. Value is case-sensitive.
Examples:

```
Time: User
```

Declares the module as User or User with pre-initialization (depends on the main class of the module).

```
Time: OnDemand
```

Declares the module as Activatable.

```
Time: Startup
```

Declares the module as startup.

Script, Startup, OnDemand

These properties are alternative for the Time property and are now deprecated.

ActivatedByDefault

This option can be used for Activatable modules only. Available values are true and false. By default this option is false. Setting this option to true means that if an Activatable module were not explicitly deactivated, it is in activated state.

Services

This property should be used when module allows to be used by some other modules. Services' names (case-sensitive) should be specified as property values. Other modules will in turn specify these names as the names of the services they depend on.
Example:

```
Name: EJB
```

Time: Startup

Main-Class: com.togethersoft.modules.ejb.EJB

```
Services: EJB, EJBClassDiagram
```

This declares that services EJB and EJBClassDiagram are available and modules that depend on them will be loaded.

DependsOn

This property is closely related to the above. It declares services that should be available to load this module. If at least one of the specified services is not available, module will not be loaded and an error message will be generated.
Example (module "A" depends on module "B"):i

```
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, then module A won't load.

Class-Path (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 fully qualified path, or relative path to this module home directory. It is also possible to use "TGH" macros. Property name in brackets should be used while declaring module using *.def file.

Example:

```
Class-Path:
libInHomeFolder.jar;$TGH$\lib\lib.jar;C:\libs\lib1.jar
```

Classes from the above-mentioned libraries are available for the module at runtime.

HelpFile

This property allows to add Help menu item to the module's popup menu in the modules tree. Path to the help file must be fully qualified.

Example:

```
Main-Class: com.togethersoft.modules.bolero.Bexport
```

Name: XP Test

```
HelpFile:
$TGH$\module\com\togethersoft\modules\xp\doc\index.html
```

Options

This option adds *Options* popup menu item to the module node in the modules tree. Value of this property is the name of the properties page, same as used in `IdeConfigManager.showConfigEditor` method.

Example:

```
Main-Class: com.togethersoft.modules.bolero.Bexport
```

Time: User

```
Name: XP Test Export
```

Options: "XP Test Options"

For "Activatable" modules this item shows when the module is in activated state only.

Known problem: only default level options can be shown this way. If options page with the specified name doesn't exist, empty dialog appears.

Hidden

Enables hide (true) or show (false) the User or Activatable modules in the modules tree.

Declaring a module in a def file

Specifying if 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 put *both* lines if you want the module to run automatically on startup and you want to be able to run it during the Together session.

The manifest file can contain some other lines that provide additional information to Together so that the module appears in, and is accessible from Together UI.

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 equality sign will be displayed as the name of the module in the Modules tab of the Explorer. In this example exercise, we'll specify a Visible Name.

Specifying the Modules tab location

You can specify where your module should appear in the Explorer's *Modules* tab. For example, you can display it in the existing *Sample* folder, or you can cause a new folder to appear. 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" in the Modules tab)

Adding the module name to menus for classes, interfaces, and members

You can cause the module to be displayed in the Modules submenu of the speedmenu for classes, interfaces, attributes and operations. Add the line:

```
PopupMenuItem = true
```

Adding the module name to menus of elements with specific shapetype

You can cause the module to show up in the Modules submenu of the speedmenus of elements having a specific *shapetype* (see the explanation of this term in 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 make the module appear in the "Modules" submenu only for operations, or only for attributes, or only for classes and interfaces. Two lines are required to accomplish this:

```
PopupMenuItem = true

PopupConstraints="shapeType=<shapetype identifier>"
```

Examples

Only for operations:

```
PopupMenuItem = true
PopupConstraints = "shapeType=Operation"
```

Only for attributes:

```
PopupMenuItem = true
PopupConstraints = "shapeType=Attributes"
```

Only for classes and interfaces:

```
PopupMenuItem = true
PopupConstraints = "shapeType=Class"
```

Note that since classes and interfaces have the same shapetype (i.e. Class), you can't selectively limit the appearance of a module to just interfaces via the manifest file. Do not worry, this *can* be done, but you will have to do a little more work in the module. To learn how to do it, refer to the module in
`%TOGETHER_HOME%\modules\com\togethersoft\modules\tutorial`
Only for actors:

```
PopupMenuItem = true
PopupConstraints = "shapeType=Actor"
```

Tip: The possible shapetypes are defined in the `RwiShapeType` interface (see the API documentation).

Rules for the manifest file

The manifest file must satisfy the following conditions:
- Manifest file name must be exactly the name of the module's main class (the class implementing `IdeStartup` or `IdeScript` interfaces).
- Manifest file location must be somewhere under the directories specified in `$TOGETHER_HOME$/config/scriptloader.config` file.
This file defines two possible root directories for manifest files:

```
1. %TOGETHER_HOME%\modules\com\togethersoft\modules

2. %TOGETHER_HOME%\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`). Invoke your text editor and create the manifest file now. Add the following lines:

```
Script = true
Startup = true
Name = My First Module
Folder = "MyFirstModuleFolder"
```

Save the file as:
`$TGH$\modules\com\togethersoft\modules\myFirstModule\MyFirstModule.def`

Compiling and storing the module

Now you have to compile the module's source code. You can do so using Together, or your favorite Java compiler. Make sure to include the `%TOGETHER_HOME%\lib\openapi.jar` file containing API classes in your compiler's classpath.

Where to store the compiled class

Keep 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 path matches the path of the manifest file counting from one of the following two directories:

```
%TOGETHER_HOME%\modules\com\togethersoft\modules
%TOGETHER_HOME%\classes\com\togethersoft\modules.
```

For example, if your manifest file is

`modules\com\togethersoft\modules\coolModule\CoolModule.def,`

then your .class files must be somewhere in the classpath under the directory

`com\togethersoft\modules\coolModule\.`

Once again, especially for your first modules, it is recommended to keep *.class files in the same directory as the manifest (.def) file and the sources. The best way is to keep them all together in a separate directory under

`%TOGETHER_HOME%\modules\com\togethersoft\modules.`

Evaluating the Results

Once you have compiled the module, run Together (if you compiled the module using Together, re-start). If the console window is on, you should see the module's startup message that marks when it was called at the startup, and when it performed `autorun` method.

Note, that although `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 will be 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, another is the compiled .class file. Select either file, right-click, and choose "Run" (for the source file this command will cause Together to try to compile and run it. If you wish to use your home-brewed compilation results, select the .class file). If the message pane is open, you should see the module's message.

Congratulations, you have just deployed your first Together module!

It is strongly recommended to sequentially comment out each line of the manifest file to feel that this makes the module work in only one mode. Try it:

```
Script = true
;Startup = true
Name = This is my first module.
```

and

```
;Script = true
Startup = true
Name = This is my first module.
(Restart Together after each change in the manifest file to
see the difference)
```

Troubleshooting

The declaration and .java files of the module described in this document are not provided with Together... that's to make *you* create your own first module :-). Once you create it, deploy it, and run it, you will find it a matter of minutes to create new modules.

After you successfully deploy and run this example (please do it. If you are going to write modules, this will save you a lot of time later), look through the sample modules located in the %TOGETHER_HOME%\modules\com\togethersoft\modules directory. Pay attention to the Tutorial folder, which contains a set of sample modules demonstrating the usage of Together's open API.

Try to take something from the first lesson modules in the tutorial folder, and use it in a new module (or, simply rename a tutorial's lesson to be your second module).

If you get some compilation errors, please check that you:

- import all the required API interfaces/classes
- added all the required libraries to the compiler's classpath
- use API methods properly. See API documentation in TGH/doc.

Also, you may face these common problems:

Everything compiles without errors, but you can't find your module in Together's Modules tab

Almost certainly, you didn't compose a .def file at all, or didn't place it under the directories specified by the scriptloader.config file.
By default, they are specified as

%TOGETHER_HOME%\modules\com\togethersoft\modules

%TOGETHER_HOME%\classes\com\togethersoft\modules

Everything compiles without errors, and .def file is located okay, but you still can't find your module in Together's Modules tab

Most likely, you made a spelling error in the lines Startup = true or Script = true in the .def file.

Or, the name of the manifest file is not identical to the name of the module. In this case Together's Message pane will display something like "The manifest file

c:\together\modules\com\togethersoft\modules\coolModule\CoolSript.def
 exists but Together either cannot find or cannot read its associated module files".

Everything compiles without errors, you see your module's folder in the Modules tab, but there are only Java sources (assuming you want to see and run the compiled module)

Make sure module's .class files are located in the right place so that Together can find them. It was discussed in this document earlier (if you keep .class files where the .def file is, you won't encounter such a problem).

Everything compiles without errors, you see the module's compiled class in the Modules tab, but your module doesn't seem to work

We strongly recommend to mark the start and finish of each of your modules (regular and startup) with appropriate messages in the Message pane. This instantly clarifies whether your module was actually running or not. For example:

```
IdeMessageManagerAccess.printMessage

(IdeMessageType.INFORMATION,"MyCoolModule:started");//start

IdeMessageManagerAccess.printMessage

(IdeMessageType.INFORMATION,"MyCoolModule:finished");//finis
h
```

Customizing System and UI

Advanced customization

This section is intended for administrators or managers who need to create shared custom configurations and/or modify Together itself in order to meet corporate standards. Power users interested in delving into the low-level customization capabilities of *Together* may also be interested in this information. The most commonly-needed customizations of the configuration properties can be done from the Options menu with the Options dialog.

The advanced topics cover several common low-level customizations. These are by no means the full range of customization possibilities. If you are interested in some customization that is not covered here, scan the underlying configuration files located in the `./config` directory of the installation. The names of these files can give you an idea of what kinds of properties they contain, and viewing the files and their comments can show you what, if any, customizations you can do through them. Customizations that are not possible through configuration properties may still be possible programmatically through the Together API. If you don't see any way to do the customization you want, contact a Together sales or support center.

User Interface customizations

Altering the user interface requires editing of some of the underlying configuration properties files. These are located in the `./config` directory of the installation. Configuration files have a `.config` extension.

It may also be necessary to edit the resource files referenced by lines in the configuration files. These are located in the `./lib/i18n` directory and have `.properties` extension. The files in this directory are of particular interest if you want to localize 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, make a backup of the original.*

Section topics:

Creating a shared multi-user configuration
Customizing property Inspectors
Customizing View Management's Show options (filtering)
Customizing patterns: See User's Guide: Working with Patterns: Developing and deploying your own patterns

See also

Configuring *Together*
Common customizations

Customizing View Management's *Show* options (filtering)

The *Show* options on the *View Management* node of the *Options* dialogs surface configuration properties contained in the `filters.config` file. These properties control what kinds of things are elided (hidden) in diagrams. By default, nothing is elided.

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 displays in the checked state in the Options dialog.

The Show options are all fully user-definable, not just those labeled "User Defined". However, the pre-defined filtering plus your customization of the "user-defined" options are usually sufficient for most needs.

Changing the display text of a Show option

You might want to change the name of the options that appear in the Options dialog... to accommodate international users for example. By default, the names of the options are extracted from a resource file. Thus, in the following line from the `filters.config` file:

```
optionsEditor.item.View
Management.item.Filters.item.A.item.shortName.name =
["filters/filter_shortName"]
```

...the text shown in blue is a reference to a string in a resource file. To change the name that displays in the Options dialog, you must either edit it the resource file, or replace the [reference] in the configuration file with a string literal in double-quotes.

To edit the resource file:

1. Open the resource file `$TOGETHER_HOME/lib/i18n/filters.properties` in a text editor.
2. Search for the name of the option... "All Packages" for example. The search should turn up a line similar to this: `all_packages=All Packages`.
3. Edit the name (shown in blue above) as desired.
4. Save and close the properties file.

The changes take effect the next time you start *Together*. Administrators with international users may want to make copies of the resource file for different language and replace the English default file in the installations of non-English speaking users.

Removing a Show option in 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 (e.g. "All Classes") in the `filters.config` file. Update the sequential information (as described in the next section) to compensate for the removal of the commented section from the file's section sequence.

Adding a Show option in 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 to get the elision 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 must 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 as shipped has 12 sections. So the above lines in the last section read as follows:

```
filter.l = hasProperty("$physicalPackage")
 ...
 optionsEditor.item.View Management.item.Filters.item.L.order = 120
```

Suppose you copy the first section and paste it at the end of the file to create a 13th section.* You would need to modify the lines as follows:

```
filter.m = hasProperty("$physicalPackage")
 ...
 optionsEditor.item.ViewManagement.item.Filters.item.M.order = 130
```

Then, in all other lines in the section you would 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, you need to 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, e.g.:

```
filter.m = hasProperty("$physicalPackage")
```

Text in blue 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 must add the property *`my_new_filter`* to the `filters.properties` file. Or you can use a literal instead:

```
filter.m.name = "My New Filter Options"]
```

Finally, you can update the other lines in the with references to resources or literals as required.

Customizing Properties' Inspectors

Properties' Inspectors are flexibly customizable. You can create custom tab pages, add new fields, change field names, add new stereotypes etc. There are several ways to do that. First, you can use Together Open API and do some coding job. The other option provides a handy way of visual customization, using the Custom Properties module. Finally, you can opt to enable config-based inspector, as in the earlier versions of Together.

Overview of the Inspector model

The new abstract model represents properties of an object in a structured way. Inspector varies its content 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.

Inspector consists of several components, each representing a group of properties, their names, and values. The model and the UI representation are independent of one another.

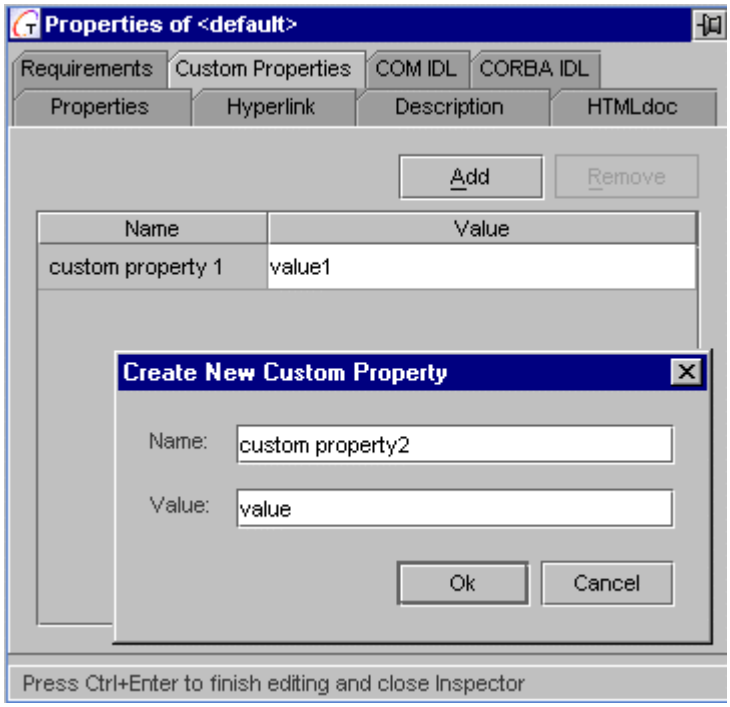
Adding custom pages and fields to the Inspector

Customizing Inspector by means of the Custom Properties Module

It is possible to visually customize inspector for a particular object. This is done by means of the Custom Properties module. Each new property added to the selected object adds 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.

1. Make sure that the module *Custom Properties* is activated. If this module is not activated, check the box Custom Properties on the menu Options | Activatable Modules.

2. Invoke the Inspector for the selected object. Additional tab *Custom Properties* appears in the Inspector:



3. Press *add property* button and specify property's name and value in the dialog window. Hit Enter to confirm. Add as many properties as required, and in the end press Control+Enter to apply the changes and close the Inspector.

4. Use the button *remove property* to delete selected properties. Press Control+Enter to apply the changes and close the Inspector.

API-based inspector customization

If you wish to create your own page or add new properties in an existing inspector, you can use Inspector API. Inspector is a startup module located in `%TGH%\modules\com\togethersoft\modules\`, and does not display in the Modules tab of Together. If you need to customize the inspector, you have to edit appropriate manifest files and classes. Updated startup module activates upon restart of Together

Classes that enable adding new page and fields to the Inspector are not included in the Inspector module, but reside in **%TGH%\modules\com\togethersoft\modules\inspector\examples**.

This is how it's done:

1. Open appropriate manifest file (*.def) in `%TGH%\modules\com\togethersoft\modules\inspector\examples` and uncomment the lines

```
MainClassName =
com.togethersoft.modules.inspector.examples.MainClassNam
e
Time = Startup
```

2. Create a Together project with the desired classes *AddPageToInspector.java* (or *AddFieldsToInspector.java*).

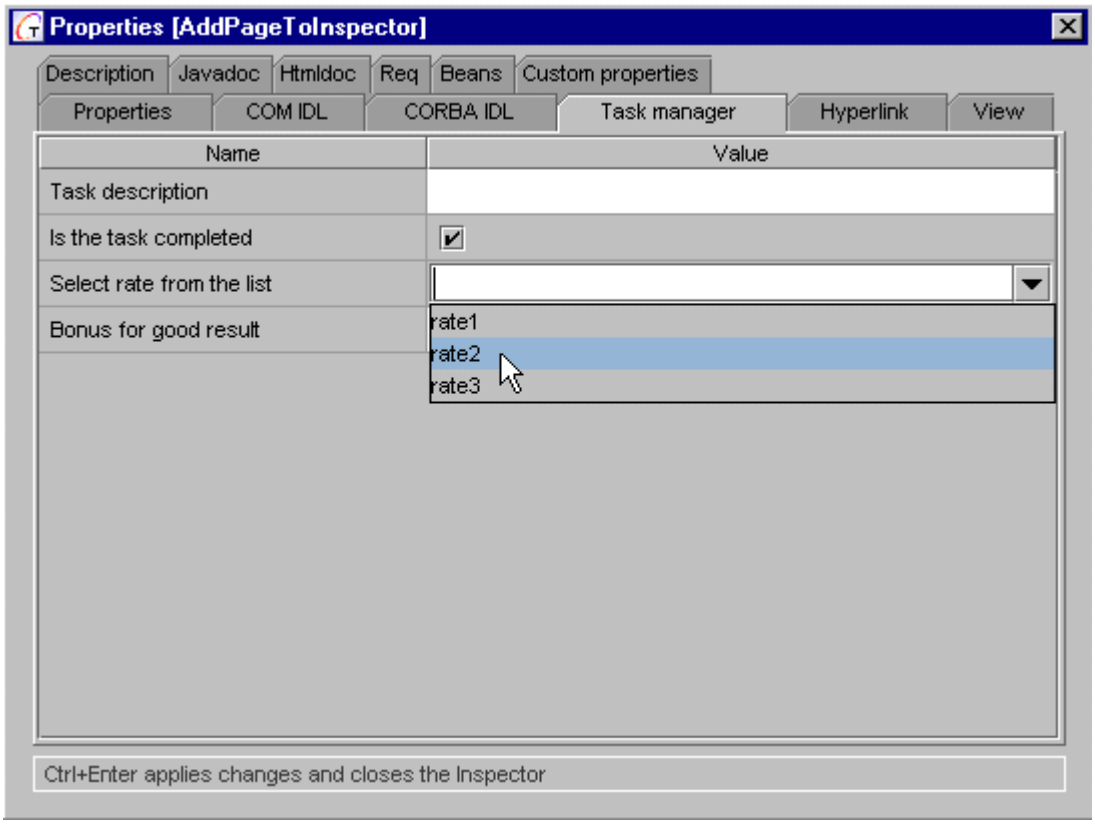
3. Create a backup copy of the class *AddPageToInspector.java* (or *AddFieldsToInspector.java*) and edit source code, specifying the desired page and field names:

```
...
IdeInspectorProperty property;
  // Replace "myPage" with the custom page name
  property = new RwiInspectorStringProperty(rwiElements,
"taskDescription");
  // Specify custom name to be shown in the UI
  property.setName("Task manager");
  page.addProperty(property, null);
  // Replace "myBooleanProperty" with the custom name of a
boolean property
  property = new RwiInspectorBooleanProperty(rwiElements,
"isReady");
  // Specify custom name to be shown in the UI
  property.setName("Is the task completed");
  page.addProperty(property, null);
  // Replace "myStringProperty" with the custom string
property name
  property = new RwiInspectorStringProperty(rwiElements,
"rate");
  // Specify custom name to be shown in the UI
  property.setName("Select rate from the list");
```

```
        // Specify custom names for the drop-down list
        SwingComboBoxEditor editor = new SwingComboBoxEditor(
        new DefaultComboBoxModel(
        new String[] {  "rate1", "rate2", "rate3"  }
        )
    ...
```

4. Compile the code. By default, the compiler puts class files to the Destination directory `%TGH%\out\classes\%PROJECT_NAME%`. Specify same location for the class file as for the source file on the Options | Project(Default) | Tools | Destination directory.

5. Restart Together. Now invoke class speedmenu and observe customized inspector:



6. If you don't need additional Inspector page any more, all you have to do is to comment out `MainClassName` and `Time` lines in the manifest file and restart Together.

More Documentation

The best way to learn how to write an inspector is to study the source code of the Inspectors delivered with Together. Together's open API provides some technical documentation of the key classes and interfaces, and a commented example of how to do a simple Inspector customization. Refer to the documentation for `com.togethersoft.openapi.ide.inspector` through the `%TGH%/doc/api/index.html`.

User-defined inspector customization

General tab of the Options dialog provides "Support user-defined inspector" flag, which toggles on and off building inspector pages from config entries, and enables visual creation of additional user-defined pages. When this flag is set, Tools menu displays Inspector Property Builder command, that invokes Inspector Property Builder dialog.

New page with the specified property name adds to the inspector for the selected elements. Together writes relevant entries to the file `changes.config` under `%TGH%/config` directory. See detailed description in the Inspector Property Builder topic of the Context Help.

Compatibility with the older versions

Presently, inspector customization through API is preferable. However, users of the versions 3.x can refer to `%TGH%\config\inspector.config` that provides detailed information on inspector customization. Besides that, uncommenting the line

```
HelpFile = "$TGH$\doc\guides\fcta.html"
```

in `%TGH%\modules\com\togethersoft\modules\inspector\examples\help.def` allows the help module "How to Create Custom Inspector" activate upon the next start of Together. This module becomes available on the menu Help | Modules.

See also

Advanced customization
Configuring Together

Configuring the New Diagram Dialog

The New Diagram dialog by default provides two tabbed pages, the first one with the standard UML diagrams, and the second with the Together modules' specific diagrams. You can edit %TGH%\config\diagram_group.config file to create your own pages with the customized set of diagram shapetypes.

Diagram shapetypes allocated in the same page form a *group*. A page is characterized by its group id, group name, weight and mnemonic:

Group id is a unique identifier for each group

Group name is a string that shows up as the page title.

Weight is a double value that defines the order, in which tabbed pages show up in the dialog (the page with the smallest weight is the first to get the focus when the dialog is invoked).

Mnemonic is a letter from the group name, not used by the other controls, that can be used for shortcut definition.

All diagrams that don't belong to a certain group, belong to the default group. To create a new page for the dialog, you have to define a new group with its properties, and specify the list of shapetypes you wish to add to the new page, conforming to the following syntax rules. The changes take effect on new start of Together, or on choosing Reload command from the Options menu.

Syntax

Element	Syntax
Default group	`diagram_group.default = <group name>`
Default group weight	`diagram_group.default.weight = <double value>`
Default group mnemonic	`diagram_group.default.mnemonic = <character>`
Custom group	`diagram_group.group.<group id> = <group name>`
Custom group weight	`diagram_group.<group id>.weight = <double value>`
Custom group mnemonic	`diagram_group.<group id>.mnemonic = <character>`
Diagram shapetype for a group	`diagram_group.<diagram_shape_type>.group = <group id>`

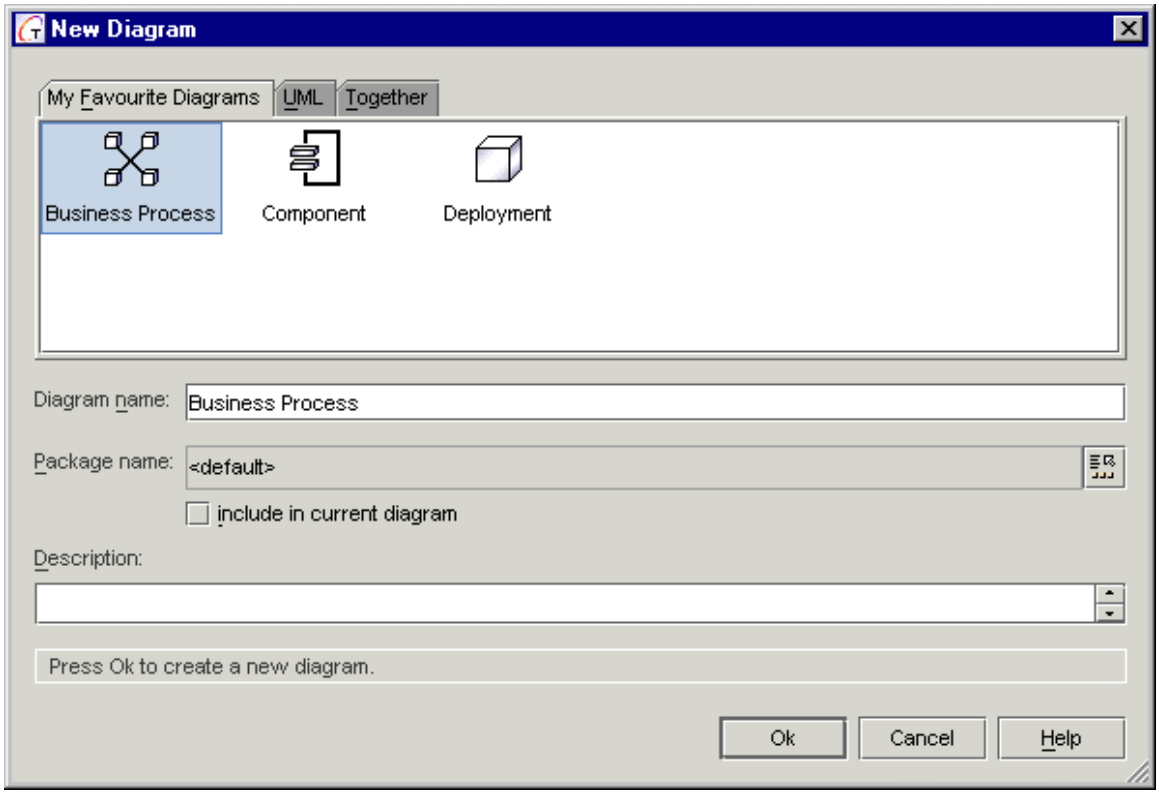
Example

The lines in the `diagram_group.config` file

```
diagram_group.group.fav = My Favourite Diagrams
```

```
diagram_group.fav.weight = 0.5
diagram_group.fav.mnemonic = F
diagram_group.ComponentDiagram.group = fav
diagram_group.DeploymentDiagram.group = fav
diagram_group.BusinessProcessDiagram.group = fav
```

produce the following result:



Defining Custom Diagram Types

Together supports the currently-defined UML diagram types, plus Business Process and Entity Relationship diagrams, but it doesn't confine you to using *only* those types. Together's Pixie (tm) technology lets you define your own custom diagram types.

Basic procedure for defining custom diagram types

1. Create icon images to display in the Together's user interface
2. Create a folder to store the custom diagram icons and add it to the paths in `Together.bat` file.
3. Create a new configuration file in %TGH%\config for each custom diagram type
4. In the configuration file, define the diagram entity and name, and create references to the icons
5. Define the diagram elements for the new diagram

Step 1: Creating diagram configuration file

To define a new diagram type, create a new file **{file_name}.config** in the folder %TGH%\config that stores all configuration properties files.

This file contains at least one line, which defines a new diagram type:
model.diagramType.{diargam_unique_ID}={diagram_unique_name}

Where:

diagram_unique_ID is any word or number which will be used to sort diagram icons in "New diagram" dialog box. If number is used it has to be greater than 9, because numbers from 1 to 9 are reserved for standard UML diagrams.

diagram_unique_name is any word which will be used to refer to this new type diagram

model.diagramType is a predefined construction to define new diagram type

Example:

```
model.diagramType.htd=Heffalump_Trap_Diagram
```

Step 2: Creating icons and icon references for UI

Two icons are required for each diagram type to be represented in Together UI: a small one for the Explorer treeview, and a big one for the New diagram dialog.

Editing Together.bat

Add path to the folder where the icons are stored to `-cp` variable of `Together.bat` file.

Example:

```
-cp "%TGH%\images;...".
```

Specifications for graphical icon images

Icon	Pixel Dimensions	File format
small (treeview)	16 x 16	GIF (transparent bg)
large (dialog)	32 x 32	GIF (transparent bg)

Referencing images in the config file

Add the following lines to 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 a name used for the new diagram

diagram_name_2 is a name used in the "New diagram" dialog box.

path_to_the_icon is a path relative to the folder specified in -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"
```

Defining element types for the custom diagram

Step 1: Defining treeview icons

First, define icons for the new diagram elements displayed in the Model treeview. This requires to create special line for each newly created diagram type in the configuration file:

resource.element.{element_unique_name}.icon.small = "{path_to_icon}"

Where:

element_unique_name is a word denoting the new element type

Example:

```
resource.element.sdEntity.icon.small = "Treeviews/tv-
sdEntity.gif"
resource.element.sdAttribute.icon.small = "Treeviews/tv-
attrib.gif"
```

Step 2: Defining toolbar icons

Toolbar icon buttons are dynamically constructed through their button definitions. Each button represents one element. Definition of each button has the following format:

```
diagram.toolbar.button.{diagram_unique_name}.{element_toolba
r_name} = \ node = createNode("{element_unique_name}")
diagram.toolbar.button.{diagram_unique_name}.{element_toolba
r_name}.condition = "{enabling_condition}"
diagram.toolbar.button.{diagram_unique_name}.{element_toolba
r_name}.icon = "{path_to_icon}"
```

Where:

element_toolbar_name is a unique name of a toolbar button

path_to_icon is a Path to the icon of the button

node is a Variable that stores the new element

createNode is an internal function that creates new node (entity). To create new association (link) use "createLink" function.

enabling_condition is any valid boolean expression that defines availability of the button

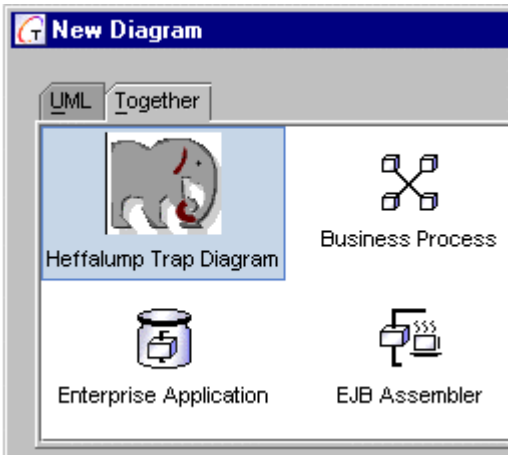
diagram.toolbar.button is a predefined construction to define new button on toolbar.

Example::

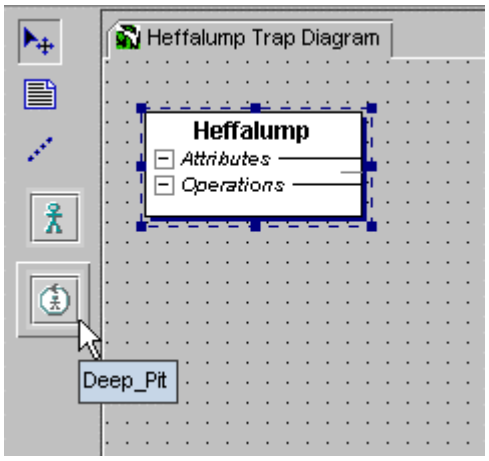
```
diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit = \
node = createNode("Deep Pit")

diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit =
%DEFAULT_LANGUAGE% != "idl"

diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit.icon
= "toolbar_icon_2.gif"
```



Custom diagram added to the New Diagram dialog



New buttons added to the toolbar

Defining Viewmaps

In order to create new elements in a diagram, it is necessary to describe graphical presentation of each one in the .config file for the new diagram type. A construction similar to that shown below must be added to the config file of a new diagram type. This section presents each step on the base of the sample config file

Step 1

Define graphical presentation starting from the header:

view.map.*.{element_unique_name}.isTopLevel() =

Where:

view.map.* is a predefined construction

element_unique_name is an element name, defined in the previous section

isTopLevel(): This construction mean that all following operations are performed if the element is on the diagram top level.

For example:

view.map..sdEntity.isTopLevel() =*

Step 2

Define the shape of the new element:

\ setGraphicObject("{shape_name}");

\ setLayoutConstraints({parameter}, [parameter], ...);

Where:

setGraphicObject: Function that sets "shape_name" of an element. The predefined shapes are:

Cube, RectangleVisible, Circle, RoundRectangle, Oval, Folder, Note, Actor etc.

setLayoutConstraints: Sets parameters of the element presentation. This function is optional.

parameter: Parameters are name-described. See the example below.

Example:

```
\ setGraphicObject("Cube");
\ setLayoutConstraints(minWidth(20),minHeight(20),
\                preferredWidth(100),preferredHeight(100),
\                canShiftX(true),canShiftY(true));
```

Step 3

Now we'll define useful options for 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 we register a new compartment with the name *Name* and shape *RectangleInvisible*., and set its layout.

Add element's name to the compartment:

\ *nameLabel = addToCompartment(label(getProperty("$name")),"Name");*

Here function `addToCompartment` adds a new label to the compartment *Name*. Function `getProperty` returns value of `$name` for this element, which was defined in the beginning of config file as *uniqueName* for the element.

Enable inplace editing

This line enables inplace editing of the element's name in the label. Expression `property:= "$name"` shows the element property to be updated.

\ *nameLabel->setInplaceEditor({property:="$name",default:=true});*

Defining label properties

The following lines set label properties:

\ *nameLabel->setAlignment("Center");*

\ *nameLabel->setLayoutConstraints(preferredHeight(16),fixedHeight(true));*

Defining links

To enable element's links to the other elements, add the following line:

\ *setCanHaveLinks()*

See the complete sample config file.

Example 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()
```

* (Support for all UML diagram types and custom diagrams varies by product. The latest information on product features is available at **www.togethersoft.com/together/**)

Web Services

Web Services

Having created a class, you might want to use it as a Web service. Together helps to accomplish this task. It is possible to transform a class into a Web service and deploy it to the target application server.

As of this writing, two plugins are provided: Apache-SOAP and BEA WebLogic 6.1 beta. In the future, this list will be extended.

> **Warning:** Keep in mind that Apache-SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It can be used as a client library to invoke SOAP services available elsewhere, or as a server-side tool to implement SOAP accessible services.
>
> To use Apache-SOAP it is necessary to install the server-side under an application server, such as Apache Tomcat v3.2, WebLogic Application Server 5.1, etc.
>
> For more detail, see Apache-SOAP documentation: **http://xml.apache.org/soap/index.html**

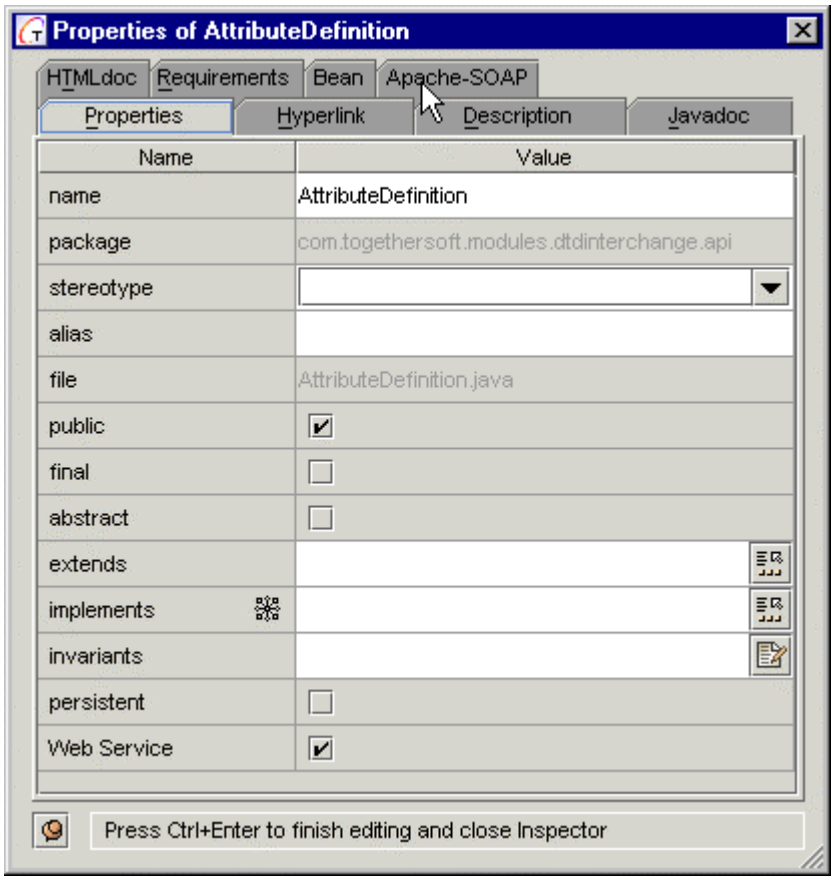
First, you have to choose the target application server. To do this, open the *Web Service* node of the *Options* dialog and choose the required application server from the drop-down list.

Creating a Web Service

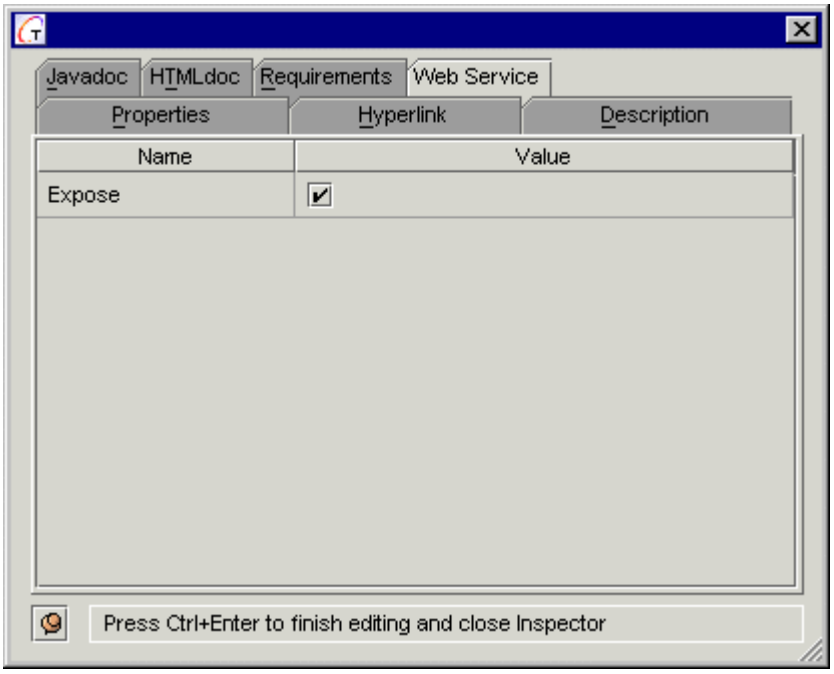
Apache-SOAP

You can create a Web Service from a *Class*. Open the necessary Class diagram, and set focus on the class that has to be transformed to a service. The Properties Inspector of a class contains a *Web Service* checkbox. When this flag is set, the new *Apache-SOAP* tab appears in the Inspector.

If you select the *Apache-SOAP* tab and set the *Message* flag on, this class will be defined as a message-style Web Service.



When the class is declared as a Web Service, its public operations become the methods of the Web Service and an additional *Web Service* tab appears in the Object Inspector with the property *exposed*:



BEA WebLogic 6.1 beta

You can use two types of Web Services: RPC-style (remote procedure call) Web Services and message-style Web Services.

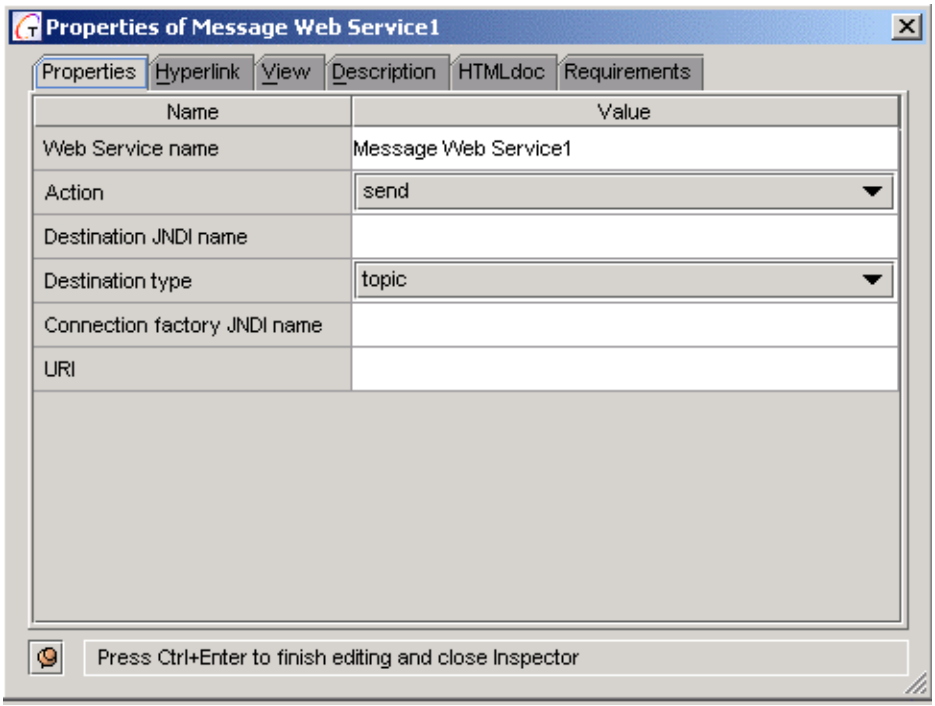
RPC-style Web Service

An RPC-style Web Service uses a stateless session EJB. You can create an RPC-style Web Service from a Session EJB. Open a Class diagram, and set focus on the Session EJB that is used as the base for the Web Service. The Properties Inspector of a session bean contains a *Web Service* checkbox.

When this flag is set, this Session EJB is defined as an RPC-style Web Service.

Message-style Web Service

Open an existent EJB Assembler diagram or create a new one. You can add a message-style Web Service using the *Message Web Service* design element  . You can see the Properties Inspector of a Message Web Service:



Action

You can specify whether the client that uses this message-style Web Service is a sender or a receiver of XML data to the JMS destination.

Select from the possible values: *send* or *receive*.

Destination JNDI name

Type the JNDI name of a JMS topic or queue.

Destination type

Select a type of JMS destination. There are two possible values: *topic* or *queue*.

Connection factory JNDI name

This field contains the JNDI name used to create a connection to the JMS destination.

URI

You can fill in this field with the URI used by clients to invoke the Web Service.

The full URL to access the Web Service is:

```
[protocol]://[host]:[port][context][uri]
```

Deployment Using the Web Service Expert

Having created a Web Service from a class, you can deploy this service to the selected application server. To do this, choose *Web Services Expert* on the *Tools* menu, which opens the expert dialog.

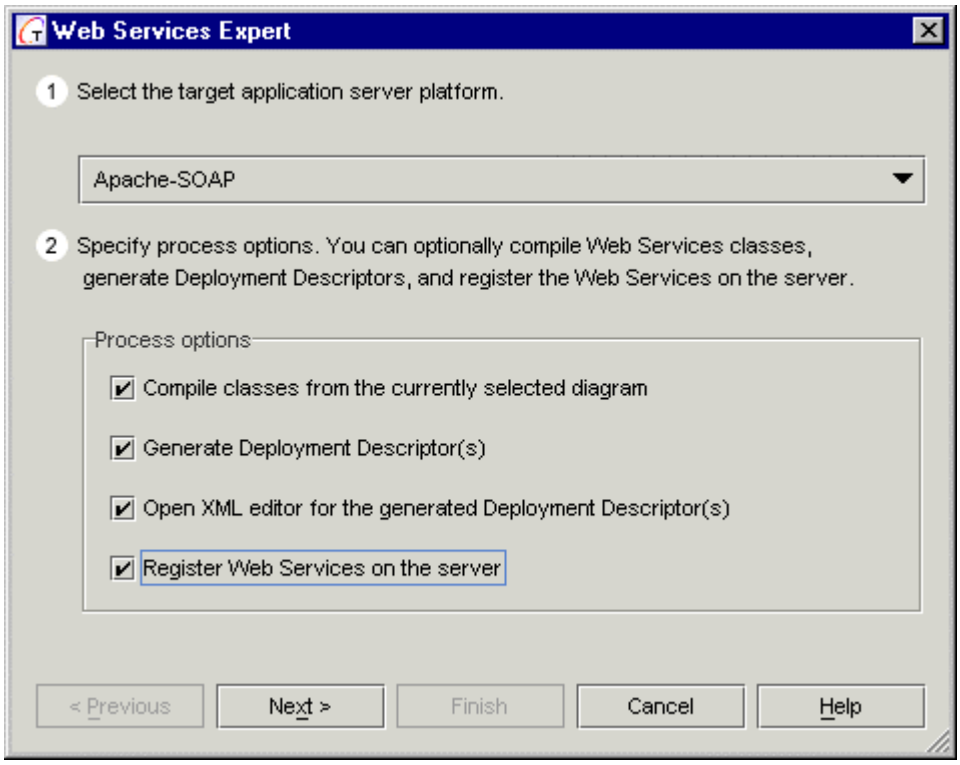
Together provides the Web Services Expert, a convenient GUI that greatly simplifies web services' deployment process. The interface of the Web Services Expert is almost the same as the interface of the J2EE Deployment Expert. Before using the Web Services Expert be sure that the server for the deployment of Web Services is installed on your computer. In this dialog, you can specify:

- the target server platform,
- what deployment-related actions you want to take place (compile, etc.),
- paths to the server, server tools, and the deployment output
- connection parameters for the server

To run the *Web Services Expert:*

1. Open the project and the class diagram containing the Web Services to be deployed.
2. On the Main menu, choose *Tools | Web Services Expert* to launch the expert dialog.
3. Choose the target server platform and set the other options as desired.

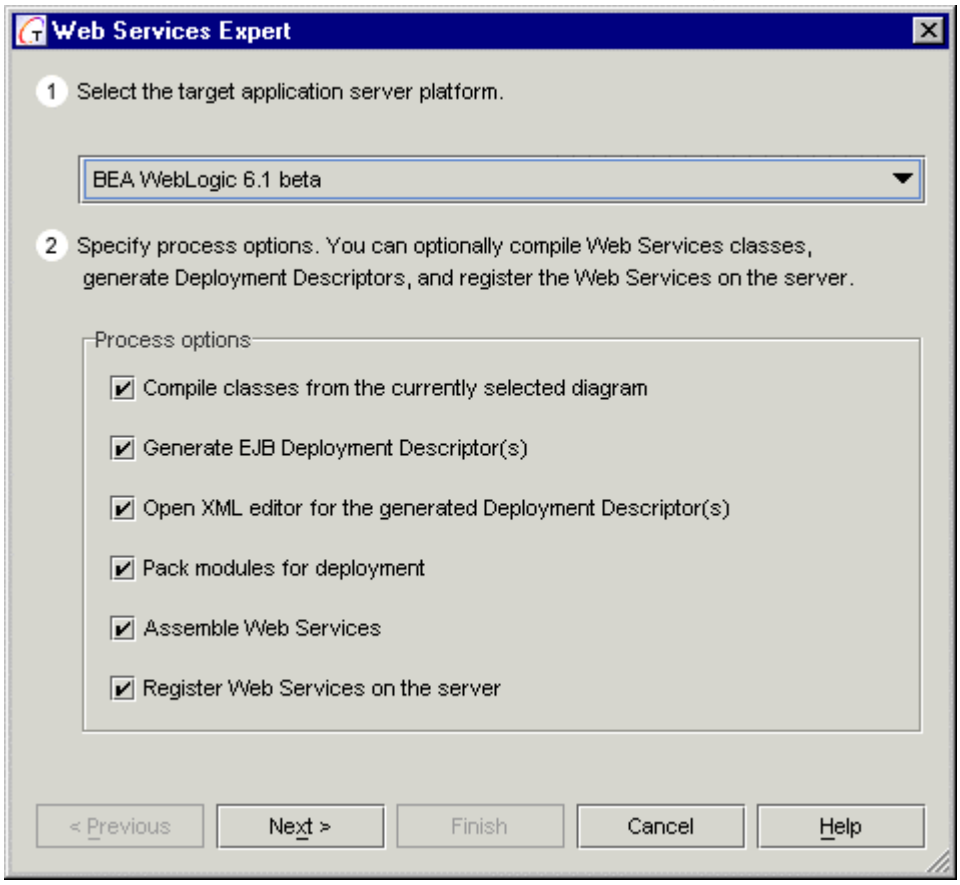
For Apache-SOAP:



If the classes need to be compiled, check the *Compile Classes* option (if they are already compiled, clear this option).

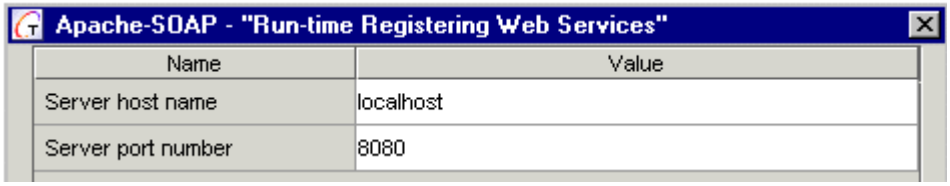
Set flags, if it is necessary to generate Deployment Descriptors, or Register Web Services on the application server.

For Weblogic 6.1. beta some additional information is required:

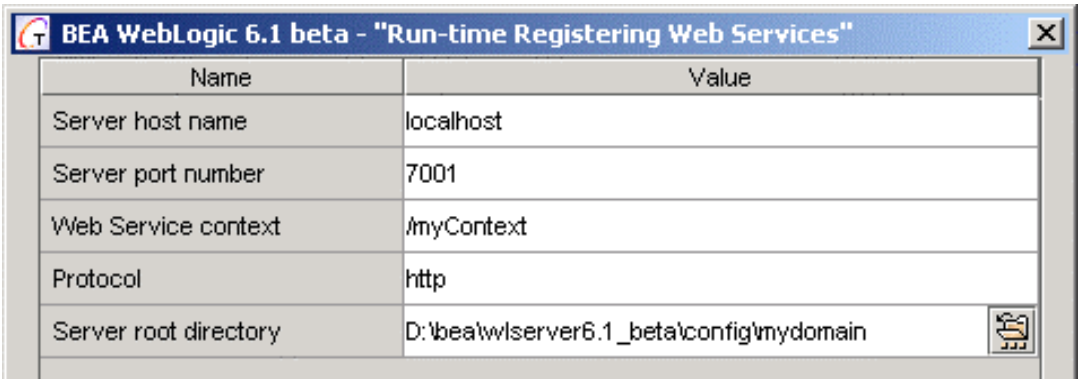


4. Click *Next* to advance through the page sequence of the Expert.
5. Specify the path to JDK 1.3, the path to the server, and the path to the folder for temporary files using the *Common Properties* page. Refer to Web Services Expert for details for the supported servers.
6. Click *Next* to continue.

You can see default settings for registering Web Services during runtime for Apache-SOAP :



and for Weblogic 6.1.beta Application Server:



7. Click *Finish*. According to your option selections, Together can handle interaction with the compiler to compile classes, generate the XML deployment descriptors, and register web services on the target application server.

See also

Web Services Expert

Index