
SilverStream eXtend Workbench

Tools Guide

Version 4.0

June 2002

SilverStream[®]

Copyright ©2002 SilverStream Software, Inc. All rights reserved.

SilverStream software products are copyrighted and all rights are reserved by SilverStream Software, Inc.

SilverStream and jBroker are registered trademarks and SilverStream eXtend is a trademark of SilverStream Software, Inc.

Title to the Software and its documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with SilverStream and its licensors, and you shall not take any action inconsistent with such title. The Software is protected by copyright laws and international treaty provisions. You shall not remove any copyright notices or other proprietary notices from the Software or its documentation, and you must reproduce such notices on all copies or extracts of the Software or its documentation. You do not acquire any rights of ownership in the Software.

Third Party Software:

Jakarta-Regexp Copyright ©1999 The Apache Software Foundation. All rights reserved. Ant Copyright ©1999 The Apache Software Foundation. All rights reserved. Xalan Copyright ©1999 The Apache Software Foundation. All rights reserved. Xerces Copyright ©1999-2000 The Apache Software Foundation. All rights reserved. Jakarta-Regexp, Ant, Xalan and Xerces software is licensed by The Apache Software Foundation and redistribution and use of Jakarta-Regexp, Ant, Xalan and Xerces in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notices, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "The Jakarta Project", "Jakarta-Regexp", "Xerces", "Xalan", "Ant" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of The Apache Software Foundation. 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.

Copyright ©1996-2000 Autonomy, Inc.

Copyright ©2000 Brett McLaughlin & Jason Hunter. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org). 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.

Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

IBM Jikes™ and Bean Scripting Framework (BSF) Copyright ©2001, International Business Machines Corporation and others. All Rights Reserved. This software contains code in executable form obtained pursuant to, and the use of which is subject to, the IBM Public License, a copy of which may be obtained at <http://oss.software.ibm.com/developerworks/opensource/license10.html>. Source code for Jikes™ is available at <http://oss.software.ibm.com/developerworks/opensource/jikes/>. Source code for BSF is available at <http://oss.software.ibm.com/developerworks/projects/bsf>.

This software contains code in executable form obtained pursuant to the Mozilla Public License, a copy of which may be obtained at <http://www.mozilla.org/MPL/>. Source code is available at <http://www.mozilla.org/rhino/download.html>.

This Software is derived in part from the SSLava™ Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved.

Contents

About This Book xi

- Purpose xi
- Audience xi
- Prerequisites xi
- Organization xi

Chapter 1 Workbench Basics 1

- What Workbench provides 2
- Workbench panes 3
- Basic Workbench operations 5
 - Starting and stopping Workbench 5
 - Using proxy servers 5
 - Opening, saving, and closing projects and files 6
- Workbench wizards 9
- Standard Workbench editors 10
 - About the Workbench source editors 10
 - Debugger 12
- Workbench viewers 12
 - Image Viewer 12
 - Class Viewer 12
- Web Service tools 13
- Setting preferences 13
 - General preferences 14
 - Build preferences 15
 - Display preferences 15
 - Text editing preferences 16
 - Printing preferences 17
 - Deployment preferences 18
 - Abbreviations preferences 19
 - File type preferences 20
 - Backup preferences 21
 - Version control preferences 23
 - Editor setup preferences 23
 - NetBeans directories preferences 24
 - XML Editor color preferences 24

Setting Workbench profiles	24
Server profile	24
Database profile	27
Registry profile	29
Using version control	29
Setting up access to version control	30
Accessing version control	36
Maintaining Todo lists	37
Working in the Todo tab	38
Working with generated items	42
Specifying a debugger	43
Specifying the command	43
Using Ant	44
What is Ant?	44
Using the Workbench Ant tools	45
Examples	48
Internationalization support	49
Specifying fonts	49
Extending the Workbench toolset and services	50
Chapter 2 Projects and Archives	51
About projects and archives	51
Organizing projects	53
Project design considerations	53
Project directory structure considerations	53
Creating projects and subprojects	56
Creating a deploy-only project	62
Working with existing source files	64
Populating projects	65
Creating source files	65
Adding to projects	68
Viewing projects	74
Maintaining projects	76
Opening a project	77
Managing general project settings	78
Managing project content settings	80
Removing files, directories, and subprojects from projects	85
Renaming a project	87
Compiling, building, and archiving	87
Setting up your Workbench environment	87
Using the commands	91
Validating archives	94

Chapter 3 Archive Deployment 95

- Workbench-supported J2EE servers 95
- Workbench deployment types 96
- Using Workbench to deploy J2EE archives 98
 - Archive contents 99
 - Creating deployment settings 102
- What Workbench does when you deploy a project 108
- Deploying Web Services 114
- Undeploying archives 115

Chapter 4 Component Wizards 117

- EJB Wizard 117
 - About the EJB Wizard 117
 - Starting the EJB Wizard 118
 - Panel sequence 118
 - Panel reference 122
- JSP Wizard 156
 - About the JSP Wizard 156
 - Starting the JSP Wizard 156
 - Specifying the JSP page name and other options 156
 - Specifying the project, directory, and package 158
 - Specifying imports 160
 - What happens 160
- Servlet Wizard 161
 - About the Servlet Wizard 161
 - Starting the Servlet Wizard 162
 - Specifying the class name and other servlet options 162
 - Specifying the project, directory, and package 163
 - Specifying which HttpServlet methods to override 165
 - Specifying which interfaces to implement 166
 - Specifying which classes and packages to import 166
- Java Class Wizard 167
 - About the Java Class Wizard 167
 - Starting the Java Class Wizard 167
 - Specifying the class name and other options 168
 - Specifying which interfaces to implement 169
 - Specifying which classes and packages to import 169
 - Specifying the project, directory, and package 169
- JavaBean Wizard 172
 - About the JavaBean Wizard 172
 - Starting the JavaBean Wizard 172

- Specifying the class name and other options 172
- Specifying the data fields 173
- Specifying which interfaces to implement 173
- Specifying which classes and packages to import 174
- Specifying the project, directory, and package 174
- Tag Handler Wizard 176
 - About the Tag Handler Wizard 177
 - Starting the Tag Handler Wizard 177
 - Specifying the class name and other options 178
 - Specifying the project, directory, and package 179
 - Specifying the tag library descriptor file 181
 - Specifying the body type 183
 - Specifying tag handler attributes 183
 - Specifying tag handler scripting variables 184
 - Specifying TagExtraInfo class 185
 - What happens 185

Chapter 5 Web Service Wizard 187

- About the wizard 187
- Using the wizard 189
- Panel sequence 189
- Panel details 191
 - Project location 191
 - WAR project selection 195
 - Class selection 196
 - WSDL file selection 197
 - Multiple namespace mapping 199
 - EJB home interface selection 200
 - EJB lookup information 201
 - Method selection 203
 - Class-generation and SOAP options 204

Chapter 6 Source Editors 211

- Common features 212
 - Standard editing features 212
 - Editor preferences 213
 - Searching across multiple files 213
 - Using text abbreviations 214
 - Changing case 215
 - Changing spaces, tabs, and indentation 215
- The NetBeans-based editors 216
 - Color coding 217

Code completion	218
Adding files types edited by NetBeans-based editors	221
Other editing support	222
The native editors	224
Changing line ending characters	224
Multiple clipboard support	225
Viewing and changing read-only and read-write attributes	225
Using the native Java, JSP, or HTML editor	225
Inserting custom tags in a JSP page	226
Chapter 7 XML Editors	229
About XML	229
XML support in Workbench	230
Using the XML Editor	230
Using the Source View	231
Using the Tree View	231
Creating and opening XML documents	233
Associating Schemas and DTDs with XML documents	235
Attaching a Schema or DTD to a document	235
Specifying a Schema or DTD in the XML document	237
Detaching a Schema or DTD	237
Converting a DTD to a Schema	238
Editing an XML document	239
About context support	239
Adding elements	242
Adding attributes	243
Adding namespace declarations	243
Editing objects	244
Using the Schema Guide	244
The Schema Guide window	245
Adding elements and attributes	248
Looking at different elements	248
Validating an XML document	249
Searching an XML document	251
Maintaining the XML catalog	252
Adding to the catalog	253
Using the XML Catalog Editor	254
Using the XSL Editor	256
Keyboard shortcuts	258
In Tree View	258
In Source View	261

In Catalog View, XML Catalog Editor	266
Chapter 8 WSDL Editor	267
About WSDL	267
About the WSDL Editor	267
Creating a new WSDL document	268
Adding elements to a WSDL document	269
Adding a message element	269
Adding a port type element	271
Adding a binding element	272
Adding a service element	274
Validating a WSDL document	276
Displaying a stylized view	277
Publishing to a registry	278
Generating Web Service files from WSDL	279
Chapter 9 Registry Manager	281
About UDDI	281
About the Registry Manager	281
Defining registry profiles	282
Browsing registries	284
Information displayed	284
Popup menus	286
Action buttons	287
Searching by business	287
Searching by service	290
Retrieving WSDL from the registry	292
Publishing to a registry	293
Chapter 10 Deployment Descriptor Editor	295
About deployment descriptors	295
About the Deployment Descriptor Editor	296
Using the Deployment Descriptor Editor	297
Chapter 11 Deployment Plan Editor	301
About Deployment Plans	301
Using the Deployment Plan Editor	301
Chapter 12 Debugger	307
Concepts you need to know	307
About the Debugger	309
Debugging server applications	312
Starting the server	312
Launching the Debugger	314

A sample debugging session	315
Debugging J2EE applications	320
Debugging client applications	321
Invoking the Debugger to start the application	322
Attaching to a running application	324
Managing program execution	326
Using breakpoints	327
Continuing execution	330
Stepping through the code	332
When the Debugger cannot locate source code	332
Analyzing the behavior of the application	332
Viewing the call stack	333
Viewing threads	333
Viewing variables	335
Debugger keyboard shortcuts	336

About This Book

Purpose

This book explains how to use SilverStream eXtend Workbench tools and facilities.

Audience

Use this guide if you are developing, assembling, or deploying J2EE and Web Service applications using Workbench.

Prerequisites

This book assumes that you are familiar with the Java programming language, the Internet, and Web applications. You can find learning materials on these topics readily available from a variety of public and commercial sources.

Organization

Here's a summary of what you'll find in this book:

Topic	Explains how to
Workbench Basics	Perform basic operations such as opening files, setting preferences, creating Workbench profiles, and using version control
Projects and Archives	Work with projects to create and maintain J2EE components and archives
Archive Deployment	Deploy J2EE archives using Workbench
Component Wizards	Use Workbench wizards to create EJBs, JSP pages, servlets, Java classes, JavaBeans, and tag handler classes (all wizards have built-in J2EE logic that facilitates the creation and deployment of well-structured J2EE components)
Web Service Wizard	Use the Web Service Wizard to generate the Java classes you need to create and access Web Services


Topic	Explains how to
Source Editors	Use the core functionality of Workbench source editors
XML Editors	Use the XML facilities provided by Workbench to create and edit XML files
WSDL Editor	Use the WSDL Editor to create and edit WSDL documents
Registry Manager	Use the Registry Manager to browse and publish to Web Service registries
Deployment Descriptor Editor	Construct and populate J2EE-compatible deployment descriptors that are used to assemble and deploy applications
Deployment Plan Editor	Create a deployment plan that performs all the tasks associated with deploying J2EE modules and applications to a SilverStream server
Debugger	Find runtime errors in Java applications (server-side objects such as J2EE applications as well as client-side objects) by controlling and monitoring the execution of Java code

1 Workbench Basics

SilverStream eXtend Workbench is an extensible IDE for developing and deploying **J2EE and Web Service applications**. Workbench automates and simplifies many tasks associated with J2EE and Web Service development, including creating and maintaining J2EE archives, populating and editing XML deployment descriptors, and deploying archives to a J2EE server.

This chapter introduces the Workbench tools and facilities. It describes how to perform basic operations such as opening files, setting preferences, setting profiles, and using version control. It includes the following sections:

- What Workbench provides
- Workbench panes
- Basic Workbench operations
- Workbench wizards
- Standard Workbench editors
- Workbench viewers
- Web Service tools
- Setting preferences
- Setting Workbench profiles
- Using version control
- Maintaining Todo lists
- Specifying a debugger
- Using Ant
- Internationalization support
- Extending the Workbench toolset and services

 For more information about performing project-level operations, such as creating a project, adding files to a project, building a project, and creating an archive, see Chapter 2, “Projects and Archives”.

What Workbench provides

Workbench is a file-system based development environment that provides:

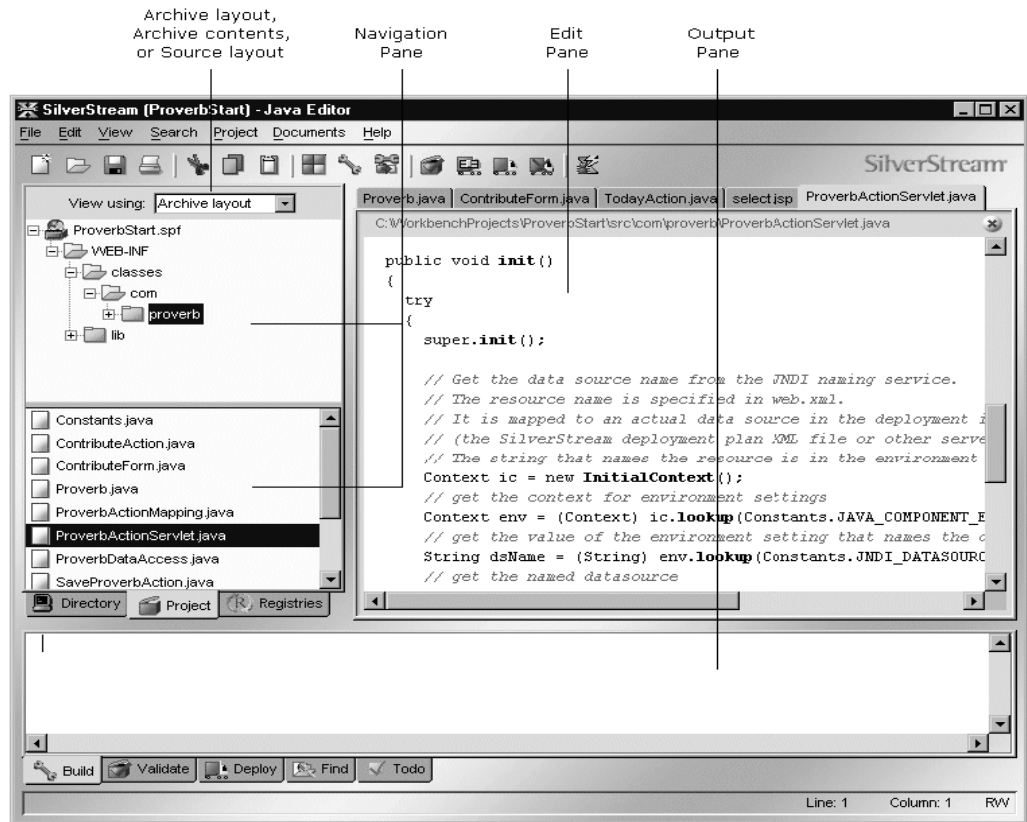
- **Component wizards** to help you create J2EE components such as JSP pages, EJBs, servlets, Java classes, JavaBeans, and tag libraries
- **Web Service facilities** including a wizard for creating Java-based Web Service components, a SOAP runtime environment, and a Registry Manager for searching and publishing to Web Service registries
- **Graphical and text-based editors** for working on Java files, JSP files, XML files, WSDL documents, deployment descriptors, and plain text files
- **Project views** that show the structure of a project's source files and the structure of a project's generated archives
- **Project tools** for building projects, creating and validating J2EE archives, and deploying archives to J2EE application servers
- **Version control integration** that provides access from Workbench to your version control system

Server independence Workbench is server-independent. When working on projects, you can create source files, build your project, and create archives all within Workbench, without any application server support. You do not need to run an application server until you are ready to deploy your project archives.

You can deploy archives built in Workbench to any J2EE-compatible application server. Workbench provides automated deployment to leading application servers, including SilverStream eXtend Application Server, BEA WebLogic, IBM WebSphere, Jakarta Tomcat, and Oracle9i. See the *Release Notes* for information about supported versions.

Workbench panes

Workbench consists of three resizable panes: Navigation, Output, and Edit. This section describes how to use these panes when creating projects.




The Navigation Pane The Navigation Pane lets you access various aspects of projects and registries. For example, the Project tab lets you view and select files from the source and archive directory structures.

At the base of the Navigation Pane are three tabs: Directory, Project, and Registries. When either the Project or Directory tab is selected, the Navigation Pane consists of two subpanes: the top one shows the directories and the bottom one lists the files in any directory that you select.

- The **Directory** tab lets you access files from the file system
- The **Project** tab shows the directory structure of source and archive layouts

- The **Registries** tab lets you browse and publish to Web Service registries

Here's how it works:

To do this	Use this	Details
Open files	Double-click	After you select a directory in the upper subpane, the lower subpane lists the files in the Directory and Project tabs
Manipulate projects, files, and directories in the Navigation Pane	Right-click	For example, depending on what you have selected, you can open files, compile files, add files to a project, remove files from a project, and so on
Switch between open files	Tabs in Edit Pane	Click the tab for the file you want to make active
Navigate to the next and previous open document	Documents menu	Use the Documents menu to navigate between the Next (Ctrl+F6) and Previous (Ctrl+Shift+F6) open document
Switch between source and archive views	Project tab	You can compare how directories and files are structured in the sources and in the generated archive  For details, see “Viewing projects” on page 74.

TIP You can see a file's complete name and path by positioning the mouse pointer over it in the lower subpane of the Directory or Project tab. Workbench tool tips are particularly useful in the Archive Layout or Archive Contents view of the Project tab for comparing a file as it conceptually exists in the archive (such as WEB-INF/web.xml) to its file system name (such as C:\dev\proj4\web.xml).

Edit Pane The Edit Pane is the file editor work area. It displays the contents of any file you have opened. You can use View menu items to hide the Output and Navigation Panes to give you additional work area.

Output Pane The Output Pane contains tabs that display information from any of the following processes: build, validate, deploy, find, and version control.

Status bar The status bar displays messages, such as when a file is saved.

About dialog Use the About dialog (**Help>About Workbench**) to check version information for Workbench and its components (editors, wizards, viewers, and so on). This dialog shows you what is installed in the product version you are running and which components have been individually revised.

Basic Workbench operations

This section tells you about:

- Starting and stopping Workbench
- Using proxy servers
- Opening, saving, and closing projects and files

Starting and stopping Workbench

Start Workbench using the appropriate command on your operating system, such as the Windows **Start** menu (**Programs>SilverStream eXtend>Workbench**).

Select **File>Exit** to exit Workbench.

Using proxy servers

If you are using a proxy server, you need to specify the proxy host and its port in **xwb.conf**, which is in the Workbench bin directory. Uncomment the following lines and specify your site's values.

```
vmarg -DsocksProxyHost=proxy-host
vmarg -DsocksProxyPort=proxy-port-number
vmarg -Dhttp.proxyHost=proxy-host
vmarg -Dhttp.proxyPort=proxy-port-number
```

If there are hosts that don't require a proxy, you can specify them (separated by |) with this property:

```
vmarg -Dhttp.nonProxyHosts=host1|host2...
```

Opening, saving, and closing projects and files

This section describes how to work with project files and source files in Workbench:

- Working with project files
- Working with source files
- Performing file system operations



For details about working with projects, see Chapter 2, “Projects and Archives”.

Working with project files

To work on an existing project, open its **project file**. Workbench project files have the extension .SPF (for SilverStream project file).

➤ To open a project file:

1. Choose **File>Open Project**. A file selection dialog appears.
2. Navigate to the project’s SPF file.
3. Select the SPF file and click open (or double-click the SPF file). Workbench displays the project in the Project tab of the Navigation Pane.

You cannot have multiple projects open (though you can work with multiple subprojects of an open project). If you have a project open and then open another unrelated project, Workbench closes the original project and any associated files before opening the second project.

Alternatively, you can:

1. Navigate to the project file in the Directory tab in the Navigation Pane of Workbench.
2. Either double-click the file or right-click it and then choose **Open File** in the popup menu that appears.

If you have opened the project file recently, you can also select it from the list under **File>Recent Files**.

➤ To save a project file:

No action is needed on your part to save a **project**. Whenever you modify the project contents or settings (for example, by adding a directory to the project), the project file is saved automatically.

The project file must be writable before you can make changes to the project in Workbench. Typically, this means that you must check out the project file from your version control system.

➤ **To close a project file:**

- Choose **File>Close Project**.

Working with source files

This section describes how to open, save, and close source files, such as Java, JSP, XML, and plain text files.

➤ **To open a source file:**

1. Choose **File>Open**.
A file selection dialog appears.
2. Navigate to the source file.
3. Select the file and click **Open** (or double-click the file). Workbench displays the file in the appropriate source file editor (Java, JSP, XML, or Text) in the Edit Pane.

Alternatively, you can:

1. Navigate to the file in the Directory tab in the Navigation Pane. If you have a project open and the file is included in that project, you can find it in the Project tab as well.
2. Either double-click the file or right-click it and choose **Open** in the popup menu that appears.

If you have opened the file recently, you can also select it from the list under **File>Recent Files**.

Working with open files One file is active at a time. By default, there is a tab for each open file in the Edit Pane. Simply click a tab to make that file active. (You can customize and turn off the display of tabs. See “Display preferences” on page 15.)

You can also make an open file the active file by selecting **Documents>More Documents** and selecting the file from the list of open documents

➤ **To save a source file:**

- Choose **File>Save**.

File>Save As enables you to save the contents of the currently open file to another file.

TIP You can also save a file by right-clicking its tab in the Edit Pane.

➤ To close a source file:

- To close the currently selected source file, select **File>Close** or click the **Close** button in the upper-right corner of the source editor.
TIP You can also close a file by right-clicking its tab in the Edit Pane.
- To close all open source files, select **File>Close All**.

If you have made changes to a source file, Workbench prompts you to save that file before closing it, closing its parent project, or exiting Workbench.

Performing file system operations

You can delete and rename files from within Workbench.

➤ To delete one or more files:

1. Go to either the Project or Directory tab in the Navigation Pane and select the directory containing the files to be deleted.
2. Select the files you want to delete. You can select multiple files using Shift+Click and Control+Click.
3. Right-click and select **Delete**.
4. Confirm the deletion.

The files are deleted from the file system.

If the files had been individually added to the current project (as opposed to being in the project because they are in a directory included in the project), you are asked whether you want to delete the entries from the project.

5. Click **Yes** to have the deleted files removed from the project.

➤ To rename a file:

1. Go to either the Project or Directory tab in the Navigation Pane and select the directory containing the file.
2. Select the file you want to rename.
3. Right-click and select **Rename**.
4. Specify the new name.

The file is renamed in the file system.


NOTE If you had multiple files selected, only the first one is renamed.





If the file had been individually added to the current project (as opposed to being in the project because it is in a directory included in the project), you are asked whether you want the project to use the new file name.

5. Click **Yes** to have the project use the new file name.

Workbench wizards

To speed project development, you can use Workbench wizards when creating J2EE projects or components. Workbench has several types of wizards:

Wizard type	Description
Project wizards	<p>Create Workbench projects associated with J2EE archives, including:</p> <ul style="list-style-type: none">• Enterprise archives (EAR)• Web archives (WAR)• EJB archives (JAR)• Application client archives (JAR)• Resource adapter archives (RAR)• Simple Java archives (JAR)• Deploy-only (nonbuildable) archives <p> For more information, see “Creating projects and subprojects” on page 56.</p>

Wizard type	Description
Component wizards	Create J2EE components, including: <ul style="list-style-type: none">• Enterprise JavaBeans• Servlets• JavaServer Pages• Java classes• JavaBeans• Tag handlers  For more information, see “Creating source files” on page 65 and Chapter 4, “Component Wizards”.
Web Service Wizard	Generate the Java classes you need to create and access Web Services.  For more information, see Chapter 5, “Web Service Wizard”.
WSDL Wizard	Create Web Services Description Language (WSDL) documents.  For more information, see Chapter 8, “WSDL Editor”.
Deployment wizards	Create deployment descriptors and SilverStream deployment plans.  For more information, see Chapter 10, “Deployment Descriptor Editor” and Chapter 11, “Deployment Plan Editor”.


Standard Workbench editors

As you create J2EE applications and components, Workbench source editors and the Debugger help you create well-structured archives that are easy to build, deploy, debug, and maintain.

About the Workbench source editors

When you open a source file, the appropriate editor starts automatically.

NOTE Opening an EAR, JAR, WAR, or ZIP file in Workbench lists the contents of the archive, along with some information about each entry. The listing is read-only.

 For a summary of the core functionality provided in all the source editors, see Chapter 6, “Source Editors”.

In addition, Workbench provides specialized functionality in the following editors.

Deployment Descriptor Editor

The Deployment Descriptor Editor lets you construct and populate J2EE deployment descriptors. A **deployment descriptor** is an XML document that provides information required for J2EE application assembly.


NOTE To open an existing deployment descriptor, right-click the project or archive (in the Project tab) and select **Open Deployment Descriptor**.

 For more information, see Chapter 10, “Deployment Descriptor Editor”.

Deployment Plan Editor


The Deployment Plan Editor lets you construct and populate deployment plans for deploying J2EE modules and applications to a SilverStream eXtend Application Server. A **deployment plan** is an XML document that describes how a J2EE module should run in the application server environment.

NOTE To open an existing deployment plan, right-click the project or archive (in the Project tab) and select **Open Deployment Plan**.

 For more information, see Chapter 11, “Deployment Plan Editor”.


XML Editor

The XML Editor lets you create, edit, and view XML files. It provides intelligent editing of XML files (by reading the XML DTD or Schema, it knows which elements and attributes are valid where) and a graphical tree view.

 For more information, see Chapter 7, “XML Editors”.

WSDL Editor

The WSDL Editor lets you create, edit, and view WSDL documents. WSDL (Web Services Description Language) is an XML vocabulary for describing Web Services.

 For more information, see Chapter 8, “WSDL Editor”.

Debugger

As part of application development, you can use the SilverStream Debugger to debug server-based applications (such as J2EE applications) and client applications. You can invoke the Debugger from within Workbench.



For more information, see Chapter 12, “Debugger”.

Workbench viewers

In addition to providing a set of source editors, Workbench provides the following viewers so you can view other files within Workbench:

- Image Viewer
- Class Viewer

Image Viewer

If you open a .gif, .jpg, .jpeg, or .png file, the file is opened in the Workbench’s Image Viewer. You can zoom the image:

- Left-click or press + to zoom in
- Ctrl+left-click or press - to zoom out
- Shift+left-click or press = to restore the image to its actual size

TIP If you want these files to open in an external program, specify the file extensions and the program in your preferences. For more information, see “File type preferences” on page 20.

Class Viewer

If you open a .class file, information about the .class file is displayed in the Class Viewer (exception: double-clicking a .class file in the Project tab’s Archive Contents view opens the corresponding .java file in the Java Editor).

The Class Viewer displays the following information:


- Name of the file, its corresponding source file, and the version of the compiler
- The class’s package statement
- The class’s declaration

- List of all fields, sorted by visibility
- List of all methods, sorted by visibility
- The same information for all inner classes

Web Service tools

Workbench supports Web Service development by providing:

- A **Web Service Wizard** to help you create Java-based Web Services and Web Service consumers from Java classes or WSDL files
- A **WSDL Editor** for creating, editing, or viewing WSDL files
- A **Registry Manager** for publishing and discovering Web Services

 For more information, see Chapter 5, “Web Service Wizard”, Chapter 8, “WSDL Editor”, and Chapter 9, “Registry Manager”.

Setting preferences

You can configure your Workbench development environment by setting:


- General preferences
- Build preferences
- Display preferences
- Text editing preferences
- Printing preferences
- Deployment preferences
- Abbreviations preferences
- File type preferences
- Backup preferences
- Version control preferences
- Editor setup preferences
- NetBeans directories preferences
- XML Editor color preferences

➤ **To specify preferences:**

1. Select **Edit>Preferences**.
The Preferences dialog appears.
2. Select the tab you want.
3. Set your preferences. See the following sections for information about specific preferences.
4. Click **OK**.

General preferences


Specify general preferences as follows:

Setting	Description
Number of recent files	Specifies how many recently open files appear in the Workbench File menu. Default is 10.
Number of recent projects	Specifies how many recently open projects appear in the Workbench File menu. Default is 5.
Reload open projects	When starting, automatically reloads the projects that were open when you last exited Workbench. Default is No.
Reload open files	When starting, automatically reloads the files that were open when you last exited Workbench. Default is No.
Web browser	Specifies the browser to use when running the Workbench help system. Choose a browser by typing the path or clicking the button.
Enable Todo	Specifies whether the Todo feature is enabled. When selected, Workbench displays a Todo tab in the Output Pane where you can maintain a Todo list of tasks.  For more information, see “Maintaining Todo lists” on page 37.

Setting	Description
Debugger command	<p>If empty, Workbench launches the SilverStream Debugger when you select Edit>Launch Debugger. See Chapter 12, “Debugger”.</p> <p>If not empty, specifies the command that Workbench invokes when you launch a debugger. See “Specifying a debugger” on page 43.</p>

Build preferences

Specify build preferences as follows:

Setting	Description
Always save modified files before compiling	<p>When set (the default), automatically saves all modified files before compiling, building, or rebuilding. When this property is not set, Workbench prompts for unsaved files.</p> <p> For more save option preferences, see “Backup preferences” on page 21.</p>
Compiler	Specifies the compiler. Default is Javac 1.3.
Compiler options	Specifies the command-line options to be sent to the Java compiler.

Display preferences

Specify display preferences as follows:

Setting	Description
Directory tab	<p>Specifies how directories and files appear in the Directory tab. Compact mode shows only the selected directory path and its related source files. Traditional mode (the default) shows all directories.</p>


Setting	Description
Project tab	Specifies how directories and files appear in the Project tab. Compact mode shows only the selected directory path and its related source files for the project. Traditional mode (the default) shows all directories.
Icons in the Navigation Pane	Specifies whether you want to show large or small icons, with or without text, in the Navigation Pane.
Edit Pane	Specifies the following: <ul style="list-style-type: none"> • Whether the Edit Pane displays tabs for each open file • Where the tabs are displayed relative to the editor • Whether, in stacked tabs, the tab for the open file is always in front. If you select this option, the row of tabs containing the active file moves if necessary to be in front (at the bottom when the tabs are above the editor and at the top if the tabs are below the editor)

Text editing preferences

Specify editor preferences as follows:

Setting	Description
Font size	Sets the screen font size in the Edit Pane. Default is 12. You can also set a print font size from the Print tab; see “Printing preferences” on page 17.
Spaces per tab character	Sets the number of spaces entered for each tab. Default is 4.
Show line numbers	Sets whether to hide (the default) or show line numbers in the Edit Pane. You can also use Ctrl+L in a source editor to toggle between hiding and showing line numbers for individual files.
Show vertical margin	Displays the margin wrap guide (set at 80 characters) at the right of the pane. Default is on.

Setting	Description
Highlight matching parentheses and braces	As you type, highlights text within a matching set of parentheses and braces. Default is on.
Use smart indenting	When you create a new line, sets the indentation level of the new line based on that of the current line. Default is on. (Not supported in the NetBeans-based JSP and HTML editors.)
Use spaces instead of tab characters	Uses spaces when the tab key is pressed. Default is off.
Use chromacoding	Color-codes the text. When deselected, all text is black. Default is on.
Show horizontal scrollbar always/only as needed	Default is only as needed.

 For additional text options, see Chapter 6, “Source Editors”.

Printing preferences

Specify printing preferences as follows:

Setting	Description
Printing mode	Sets mode to monochrome (the default) or color. For color printers, use color mode.
Font size	Sets print font size (default is 10). You can also set a screen font size in the Text Editing tab; see “Text editing preferences” on page 16.
Print line numbers	Sets whether or not to print line numbers. Default is to not print numbers.

Deployment preferences


Deployment preferences are used by the Deployment Descriptor Editor and the Deployment Plan Editor. When you open either of these editors, Workbench loads all of the project's classes (including subproject classes). The editor then uses information from the classes to populate the dialogs that display lists of classes, methods, member variables, and so on. If the classes are not up to date, the information the editors display can be incorrect or missing.

You can control whether Workbench automatically builds a project when you access the Deployment Descriptor Editor or Deployment Plan Editor. You can specify one of the following build settings:

Setting	Description
Always automatically build my project	<p>Builds the project automatically when the Deployment Descriptor Editor or Deployment Plan Editor is opened.</p> <p>This setting ensures that the editors always have access to all of the latest classes.</p>
Never automatically build my project	<p>None of the project's files are built when the Deployment Descriptor Editor or Deployment Plan Editor is opened. You must build the projects and subprojects manually.</p> <p>Use this setting when you don't need anything to be built (such as when you are editing in XML mode). In this case you can edit the XML files, but the list of project classes in the editor may be blank or out of date.</p>
Prompt me to build my project	<p>Prompts you to build the project each time you open one of the deployment editors.</p> <p>Use this setting when you want to specify when a project build should occur.</p>

NOTE You do not need to create the archives in order for the deployment editors to load class information, because the editors load the classes directly from the file system, not from the archives.

You can also specify the following deployment preference:

Setting	Description
Default server versions	<p>Specifies the server version initially selected when you create a new SilverStream deployment plan. You can specify a server version for J2EE 1.2 projects and a server version for J2EE 1.3 projects.</p> <p> For more information, see Chapter 11, “Deployment Plan Editor”.</p>

Abbreviations preferences

You can define abbreviations that can be expanded to one or more lines of text—such as a word that expands to a predefined language construct. Once you have defined an abbreviation, you can type its name in an editor and select **Edit>Text Tools>Complete Abbreviation** (or press **Ctrl+U**) to replace the abbreviation with the expanded text.

Use **%c** in an abbreviation’s definition to signify where the insertion point will be positioned when the abbreviation has been expanded.

For example, the abbreviation **main** is predefined as follows and is meant to be used in Java files:

```
public static void main(String args[])
{
    %c
}
```

➤ To define an abbreviation:

1. Select **Edit>Preferences** and click the **Abbreviations** tab.
2. Click **Add**.
3. Type the abbreviation (shortcut) in the Abbreviation text box and click **OK**.
Abbreviations must be single words and are case-sensitive.
4. Click inside the Definition text box and type the text you want the abbreviation to expand to.
5. Click **OK**.

➤ **To delete an abbreviation:**

1. Select **Edit>Preferences** and click the **Abbreviations** tab.
2. Select the abbreviation in the Abbreviations text box.
3. Click **Delete**.
4. Click **OK**.

➤ **To edit an abbreviation:**

1. Select **Edit>Preferences** and click the **Abbreviations** tab.
2. Select the abbreviation in the Abbreviations text box.
3. Click inside the Definition text box and change the abbreviation.
4. Click **OK**.

➤ **To use an abbreviation in your source code:**

1. Type the abbreviation shortcut in the editor.
2. Position the cursor within the shortcut text or highlight it.
3. Click **Edit>Text Tools>Complete Abbreviation** or press **Ctrl+U**. The shortcut text is replaced with the expanded text defined for that abbreviation.

NOTE If the abbreviation text is not defined, the **Complete Abbreviation** command is ignored.

File type preferences

Workbench lets you use third-party tools to edit specific file types. You can set preferences that let you launch files in an external editor rather than opening them in Workbench. Use the File Types tab to associate file extensions with external editors. For each file type you can choose whether to open the file in:

- Workbench
- The Windows default editor for that file type
- The Windows program you specify

➤ **To define how a file type is launched:**

1. Select **Edit>Preferences** and click the **File Types** tab.
2. Click **Add**.
3. Type the file extension in the dialog and click **OK**.

4. Specify one of the following preferences for the file type:

Setting	Description
Open in Workbench	(The default) Opens files using the Workbench source editor.
Open using the default Windows program	Opens files using the Windows default editor for that file type (same as double-clicking a file in Windows Explorer—for example, using Notepad to open files with the .TXT extension).
Open using this application	Opens files with the application you specify. You can type the path to the application, or click Browse and navigate to the application. NOTE Because some editors launch a new program instance each time you open a file, this setting is not always recommended.

5. Click **OK**.

CAUTION *Consider the following when associating an external editor with the **XML** file extension: if you use an external editor to edit SilverStream deployment plans, you will not be able to take advantage of the default setup that the SilverStream editor provides and your project will not be associated with a deployment plan.*

Backup preferences

You can set Workbench preferences that control:

- Autosave files—successive copies of a modified file (each successive save replaces the preceding one). Autosave files are copies of any file in the Edit Pane that has been modified.
- Backup files—the original file before it was modified and saved.

By default, autosave and backup operations are **not** enabled.

You can set global backup preferences that control how all projects are backed up. However, because your projects may contain files with identical names, you may want to store separate backup and autosave files for each project. To do so, specify a subdirectory relative to the file's source directory for both backup and autosave files. Files that are backed up in parallel backup directories won't be overwritten.

NOTE When you specify a relative name for a backup or autosave directory, it will be relative to the **source file**.

Specify autosave and/or backup preferences as follows:


Setting	Description and parameters	
Auto save enabled	While you make changes to a source file in Workbench, periodically save a copy of the file.	
	Auto save to same directory as source file (default)	—
	Auto save directory	Specifies another directory to contain saved files You can enter an absolute path or specify a path that is relative to the source directory. Use Browse to search for a directory on the file system.
	Auto save extension	Specifies the extension of autosave files (default is .SAV)
	Auto save interval (minutes)	Specifies how often you want Workbench to save files (default is every five minutes)

Setting	Description and parameters	
Backup enabled	When you save a source file in Workbench, make a backup copy of the previous version of the file.	
	Backup to same directory as source file (default)	—
	Backup directory	Specifies another directory to contain backup files You can enter an absolute path or specify a path that is relative to the source directory. Use Browse to search for a directory on the file system.
	Backup extension	Specifies the extension of backup files (default is .BAK)

➤ **To define how files are autosaved and/or backed up:**


1. Select **Edit>Preferences** and click the **Backup** tab.
2. Choose **Auto save enabled** and/or **Backup enabled**.
3. Specify autosave and/or backup file parameters as described above.
4. Click **OK**.

Version control preferences


 See “Using version control” on page 29.

Editor setup preferences

These preferences specify which types of files will be edited using the Workbench NetBeans-based editors.

 See “Adding files types edited by NetBeans-based editors” on page 221 and “Using the native Java, JSP, or HTML editor” on page 225.

NetBeans directories preferences

 See “Creating parser database files” on page 220.

XML Editor color preferences

You can specify the colors used in the XML Editor’s Source View to display different types of information in XML documents, such as tags, arguments, values, text, errors, and white space (listed in the dialog as **ws**).

For each type of information, you can specify a foreground and a background color. You can pick from a list of colors or define your own by clicking the ellipsis button. You can also specify whether to use a bold font.

➤ To specify colors used in the XML Editor:

1. Select **Edit>Preferences** and click the tab for XML Editor colors.
2. Select the type of information whose color you want to specify, then specify a foreground and/or a background color and specify whether you want to use a bold font.

Setting Workbench profiles


You can define the following types of Workbench profiles:

- Server profile
- Database profile
- Registry profile

Server profile


A **server profile** stores information about an application server, including the server’s host name and port. When selected at deployment time, the server profile tells Workbench what server to deploy to and provides the information required for deployment to that server. A server profile applies to a specific server. If you are deploying to multiple servers, you need to set up a separate profile for each.

Your server’s configuration determines how to specify the server profile information. For example, if your server uses security certificates you will specify the https protocol. The server configuration may also affect how you specify the server name, server port number, database name, and so on.

 For information about configuring a particular application server, see the product documentation for that server.

➤ **To create a server profile:**

1. Select **Edit>Profiles**.
2. On the **Servers** tab of the Profiles dialog, click **New**.
3. Specify settings in the Create a New Server Profile dialog as follows:

Setting	Description
Profile name	Enter a name to identify the profile. NOTE The server profile name cannot contain the period (.) character.
Server type	Select a server type from the list. Server types are organized by brand and version number. The version number indicates the lowest version supported by a given server type. A server type is often valid for multiple subsequent versions as well. As a rule, you should select the server type for your brand that is closest to the target server's version, without being higher. For example, if your target is Version 3.7.5 of the SilverStream eXtend Application Server, the server type to select is SilverStream 3.7.4 or higher.
Deployment tools directory	Specify the directory containing the executables used to deploy to the server.
Rapid deployment directory	For rapid deployment only. Enter the directory where you want Workbench to write the files for rapid deployment. Some servers require that files be written to a specific directory for rapid deploys. Make sure that you specify the appropriate location for your server's configuration. See "Setting rapid deployment directories" on page 26 for the directory listings.  For more information on rapid deployment, see "Workbench rapid deployment" on page 96.

Setting	Description
Server name	<p>Set the server name using the following formats.</p> <p>For servers running http:</p> <pre>servername http://servername[:port]</pre> <p>For servers running https:</p> <pre>https://servername[:port]</pre> <p>Specify the port number if the server is not listening on the default port.</p>
Database name	<p>For SilverStream eXtend Application Servers only.</p> <p>Enter the name of the database to deploy to.</p>
Target servers	<p>For BEA WebLogic servers only.</p> <p>Enter the names of the target servers.</p>

4. Click **OK** to close the Create a New Server Profile dialog
5. Click **OK** to close the Profiles dialog.

Setting rapid deployment directories This table shows the rapid deployment directory you should specify in the Server Profile dialog.

Server	Rapid deployment directory
SilverStream eXtend Application Server 3.7.2	Does not require that you specify a rapid deployment directory.
SilverStream eXtend Application Server 3.7.3 or higher	%INSTALL_DIR%\webapps
BEA WebLogic	%INSTALL_DIR%\config\targetname\application
IBM WebSphere	%INSTALL_DIR%\appserver\installedapps
Jakarta Tomcat	%INSTALL_DIR%\webapps

Server	Rapid deployment directory
Oracle9i AS	%INSTALL_DIR%\j2ee\home\applications Optionally, you can rapid deploy WARs to: %INSTALL_DIR%\j2ee\home\default-web-app
SUN RI	%INSTALL_DIR%\public_html

Using Workbench with secure servers Workbench connects to the target J2EE server at deployment time using the server profile. If the server profile indicates a secure server, then Workbench will make the SSL connection automatically. Workbench uses the set of commercial Certificate Authority certificates listed in agrootca.jar (located in Workbench's lib directory). If the server you are trying to deploy to uses a certificate that was issued by a CA certificate that is not listed in agrootca.jar, Workbench will not successfully connect to the server. You can add the CA certificate to agrootca.jar using any tool that allows you to modify the contents of a JAR file (for example, Sun's JAR utility or WinZip).

Database profile

You'll need to set up a database profile if you use any of the following Workbench components:

Workbench component	When use database profiles
EJB Wizard	When creating entity beans based on a database table
Deployment Plan Editor	When mapping the persistent fields of a container-managed entity bean to fields in a datasource

The database profile provides JDBC information that enables Workbench to connect to the datasource and retrieve table and field information. You can create multiple profiles to support different databases and JDBC drivers.

➤ To create a database profile:

1. Select **Edit>Profiles**.
2. On the **Databases** tab of the Profiles dialog, click **New**.

3. Specify settings in the Create a New Database Profile dialog as follows:

Setting	Description
Profile name	Enter any name to identify the profile.
JDBC Driver	<p>Enter the class name of the JDBC driver. You can specify any JDBC 2.0-compliant driver.</p> <p>To use the Sun JDBC-ODBC bridge driver (which is included in the JRE), specify sun.jdbc.odbc.JdbcOdbcDriver. If you specify a JDBC driver other than Sun's bridge driver, make sure that the driver class can be loaded by Workbench (see "To make the driver class available:" below).</p>
JDBC URL	<p>Enter an URL that specifies the database you want—for example, jdbc:odbc:TestDB</p> <p>NOTE The text you enter after the first colon is driver specific.</p>
Connection Catalog	<p>(Optional) Specify which SQL catalog (subset) of the database to connect to—for example, PayrollDb. If your database driver does not support catalogs, it will ignore this request.</p> <p>If supported, the connection catalog lets you set up which database tables are retrieved. Connection catalogs are useful when you are connecting to a very large database or only want to connect to a subset of database tables (for example, to exclude production database access).</p>
Datasource Name	<p>Specify the name of the data source to associate with this database profile.</p> <p>You can specify either the datasource name, like SilverBooks, or the full JNDI specification, like java:pm/JDBC/SilverBooks.</p>

4. Click **Test** to check the connection to the database specified by the JDBC URL. This test makes a JDBC connection to the database. The test will fail when a connection is not available or a setting is not correctly specified.

5. On the test popup, enter your database user name and password and click **OK** to verify access.
6. Click **OK** to close the Create a New Database Profile dialog.
7. Click **OK** to close the Profiles dialog.

➤ **To make the driver class available:**

1. Obtain the JAR or other archive file that contains the JDBC driver.
2. Do one of the following:
 - Put the JAR in the Workbench **lib\ext** directory.
 - Edit the Workbench configuration file (`bin\xwb.conf`) to point to the driver archive by including the line `addcp path/mydriver.jar`. For example:

```
addcp c:/sybase/SybJConnect.jar
```
3. Start Workbench.

Registry profile

Workbench provides a facility for defining profiles for Web Service registries. These profiles supply the information that allows you to search registries and deploy Web Services.



For more information on registry profiles, see “Defining registry profiles” on page 282.

Using version control

If you use a version control system, you can set up Workbench to access it. This enables you to perform version control operations on the files in your projects while working in the IDE.

- Setting up access to version control
- Accessing version control

Setting up access to version control

Before you can perform version control operations in Workbench, you need to adjust preference settings to enable version control and configure support for your version control system.

➤ To adjust version control settings:

1. Select **Edit>Preferences** to display the Preferences dialog, then go to the **Version Control** tab.
2. Check the **Enable Version Control** property.
This turns on the version control features of Workbench.
3. Select one of the available **Version Control Systems**.

In this property, you're actually choosing a version control system **definition** that tells Workbench the version control commands to support. Workbench comes with definitions for several popular version control systems (ClearCase, CS-RCS, CVS, PVCS, Visual SourceSafe). If you choose one of these, you can use it as is or edit the commands it defines to suit your needs and system configuration.

You also have the option of creating version control system definitions yourself. This lets you set up Workbench support for just about any version control system you might have.

Working with definitions The following topics provide more detail about working with version control system definitions:

- Editing a version control system definition
- Creating a version control system definition
- Distributing a version control system definition
- Deleting a version control system definition

Editing a version control system definition

A version control system definition specifies a list of version control **menu items** that Workbench is to display. Each menu item is mapped to a **command-line operation** of the chosen version control system and also specifies details about how that operation is to be executed. You can edit the list to modify, create, or delete menu items.

➤ To edit a definition:

1. From the Version Control tab of the Preferences dialog, select a definition from the **Version Control Systems** dropdown list.
2. Click the **Setup** button.

3. In the Setup dialog, make your changes to the list of version control menu items:

If you want to	Do this
Change the behavior of a menu item	Select that item from the Version Control Command listbox, then edit its Command properties.
Change the name of a selected menu item	Click the Edit button. The name can include letters, numbers, spaces, and special characters. You can also edit the mnemonic character to be used for keyboard access to the menu item (when pressed in combination with the Alt key).
Create a new menu item	Click the Add button, then specify the item's name and mnemonic character. Your new item will be added to the end of the list.
Delete a selected menu item	Click the Remove button.
Switch the order of menu items	Select an item you want to reposition, then use the arrow buttons to move it up or down in the list.

Command properties The following table describes the command-related properties you can specify in the Setup dialog for version control menu items:

Property	Description
Command	A command-line operation of your version control system that the menu item is to execute. You can include environment variables in the command by using the syntax <code>%varname%</code> or <code>\${varname}</code> . Workbench substitutes the values of these variables when the command executes. If the value of a variable can't be determined, an empty string is substituted. Predefined environment variables are available via the expand button next to the Command property. You can select a variable to insert it at the current cursor position.

Property	Description
Reload when done	Tells Workbench to try reloading the target file after the command executes. This is useful for commands that might modify the file (such as check in, check out, or get).
Wait for execution	Tells Workbench to wait until the command finishes executing before returning control to the user. Not waiting for execution can be appropriate for commands such as diff or history where there's no effect on the target file.
Execute command in directory of source file	Tells Workbench to execute the command relative to the directory of the target file. If you don't check this property, the command executes in the current directory.

Predefined environment variables The following table describes the predefined environment variables you can include in the command you specify for a version control menu item:


Variable	Description
<code>%_PATH%</code>	Full path and name of the target file. For example: <code>x:/com/myco/myfile.java</code>
<code>%_DIR%</code>	Directory of the target file. For example: <code>x:/com/myco</code>
<code>%_NAME%</code>	Name of the target file (without directory). For example: <code>myfile.java</code>
<code>%_BNAME%</code>	Base name of the target file (without directory and extension). For example: <code>myfile</code>

Variable	Description
%_EXT%	Extension of the target file. For example: java
%_PROMPT <i>prompt-text</i> %	Prompts the user for a value by displaying a dialog. The dialog includes any <i>prompt-text</i> you specify. The value of this variable is whatever the user types in the dialog input field. If the user clicks the dialog's Cancel button, the entire command is canceled.
%_COMMENT%	Prompts the user for a comment. The comment is saved to a temporary file. The value of this variable is the name of that temporary file.

Creating a version control system definition

If Workbench doesn't provide a definition for your version control system, you can create one yourself.

➤ To create a definition:

- From the **Version Control** tab of the Preferences dialog, click the **Add** button.
- When prompted, type a **name** for your version control system definition.
The definition name can include letters, numbers, spaces, and certain special characters. The name you specify is added to the **Version Control Systems** dropdown list.
Workbench also creates an XML file to store your definition. The name of this file matches the definition name you specify (except that spaces are replaced by underscores). Workbench saves your definition XML file in its **Resources\version_control_config** directory (along with the definition XML files it provides).
- When the **Setup** dialog displays, specify the details of your version control system definition.
 See Editing a version control system definition.

Distributing a version control system definition

Once you edit or create a version control system definition, you might want to copy it to other computers where Workbench is installed.

➤ **To distribute a definition:**

1. Find the XML file for your version control system definition in the Workbench **Resources\version_control_config** directory.
2. Copy that file to the corresponding directory on each target computer.
When Workbench is run on those computers, the **Version Control Systems** dropdown list (on the Version Control tab of the Preferences dialog) will automatically include your copied definition.

Deleting a version control system definition

If you don't need a particular version control system definition, you can remove it.

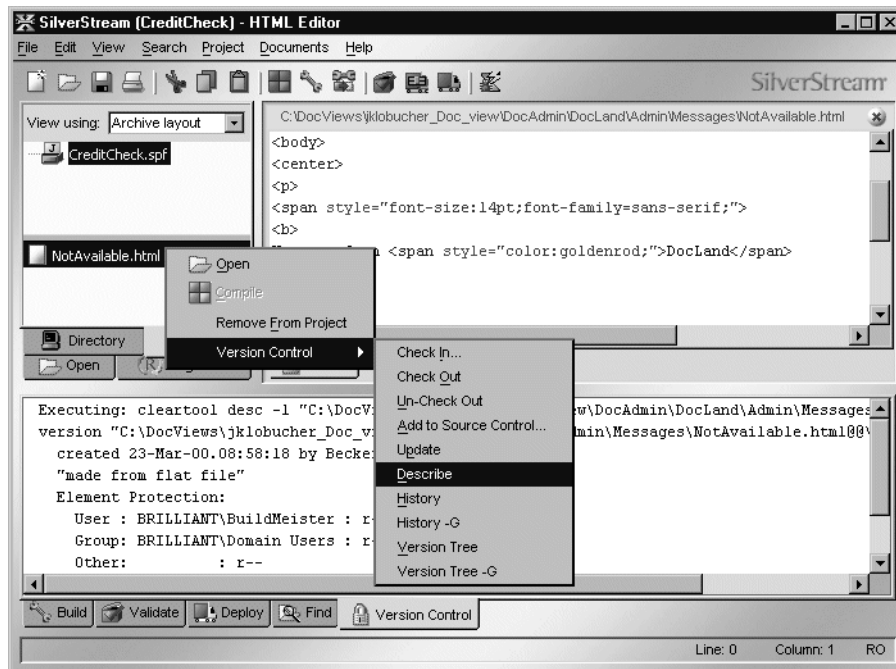
➤ **To delete a definition:**

1. From the Version Control tab of the Preferences dialog, select a definition from the **Version Control Systems** dropdown list.
2. Click the **Remove** button.
Workbench prompts you to confirm, then deletes that definition from the list. The definition's XML file is deleted from the Workbench **Resources\version_control_config** directory.

Accessing version control

When you use Workbench with version control access enabled, the commands specified by the active version control system definition are available via a popup menu. You just need to right-click one of the following:

- Any file name on the Project tab of the Navigation Pane
- An open file in the Edit Pane

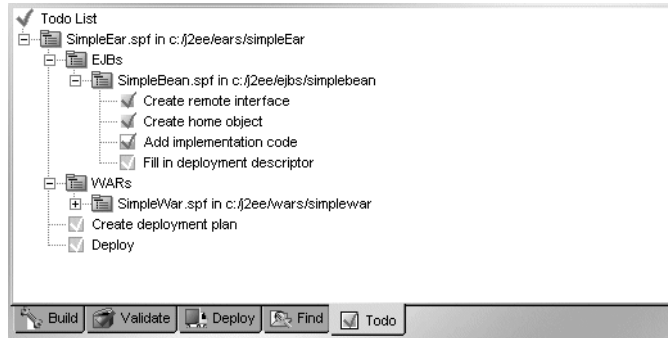


When you execute a version control command, resulting text messages display on the Version Control tab of the Output Pane.

Maintaining Todo lists

Developing J2EE and Web Service applications can be quite a complex undertaking. It is sometimes hard to manage the work. With that in mind, Workbench provides the ability for you to maintain a Todo list that organizes and tracks your tasks.

You maintain your Todo list in the Todo tab of the Output Pane.



You can:

- Create Todo items
- Associate items with Workbench projects or mark them as independent of particular projects
- Mark the completion status of items
- Create a hierarchy of items
- Reorganize the items in the hierarchy
- Delete items

In addition, various Workbench wizards and tools generate items in your Todo list to point you to areas where work needs to be done and to describe the nature of that work.

Working in the Todo tab

When you first click the Todo tab, the Todo list is empty (unless you have run a tool or wizard that populates the list; see “Working with generated items” on page 42).

Creating items The first thing you’ll do is add one or more items, which can be tasks or folders.

➤ To add an item:

1. Select the item following which you want to add an item and either press **Ins** or right-click and select **Add Item**.

TIP You can also use **Edit>Add Todo Item** to insert an item at the end of the list, or press **Shift+Ins** to add the item as a child of the selected item.

The Add Todo Item dialog displays.

2. Enter the following information:

Item	Description
Description	Text to display for the item in the Todo list
Note	(Optional) Additional information about the item. This text displays as part of the item’s tool tip when the mouse pointer is over the item
Add to open project	<p>If you want to associate this item with an open project, select the project from the list.</p> <p>If you select a project, the Todo item is added to the end of the list in the project’s folder (the folder is created if necessary). A project’s Todo folder is a top-level folder named:</p> <p style="text-align: center;"><i>projectFile</i> in <i>pathToProject</i></p> <p>For example, if a project file is in c:\WorkbenchProjects\myEAR\MyEAR.spf, the project folder will be named:</p> <p style="text-align: center;">MyEAR.spf in c:/WorkbenchProjects/myEAR</p>

3. Click **OK**.

The item is created.

If you associated the item with a project, it is created as the last item in that project’s list.

If you did not associate the item with a project, it is created as a sibling following the selected item (unless no item was selected when you added the item, in which case the item is added as the first item in the list, or unless you added the item with Shift+Ins, in which case the new item is a child of the selected item).

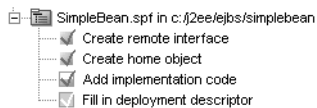
New items appear with the description you entered, along with a checkbox. The checkbox indicates the completion status of the item (see the next section).

Editing items You indicate an item's completion status, as well as revise its description and note, by editing the item.

➤ To edit an item:

1. Select the item.
2. Right-click and select **Edit Item**.
The Edit Todo Item dialog displays.
3. Update the information as appropriate. To indicate completion status, select a value from the Percent done listbox or type a value.
4. Click **OK**.

A faded checkbox indicates that the task has not begun. A half-filled checkbox indicates partial completion. A filled checkbox indicates completion.



TIP You can also toggle an item's completion status between 0 and 100 percent by selecting the item, right-clicking, and selecting **Toggle Item(s) Done**. If the completion status was zero, it is set to 100; if it was non-zero, it is set to zero.

Tool tips When you position the mouse pointer over an item, the item's tool tip displays as:

percent done; Notes: noteText

Creating a hierarchy Todo lists can be hierarchical—items can contain other items. For example, you can create a folder of related tasks.

➤ To create a hierarchy:

- Move one or more items under another one by selecting the item(s) and either pressing > or by right-clicking and selecting **Indent**.

The item becomes a child of its previous sibling, which is now a folder.

Similarly, to outdent one or more items, select them and press < or right-click and select **Outdent**. If an item no longer has children, it is no longer displayed as a folder.

Moving items You can move an item around with drag and drop: Press and hold the mouse button on an item and move the item within the list. A horizontal line indicates where the item will be moved to. Release the mouse button to move the item. Moving a folder moves all of its contents as well.

You can move an item anywhere in the list.

TIP You can drag more than one item at a time as long as you **drag** as soon as you have selected the last of the multiple entries. (If you **click** after selecting the last entry, it reverts to a single selection. This behavior is a limitation of the JDK 1.3 version of the control used in the Todo tab and has been addressed in JDK 1.4.)

Deleting items You can delete one or more items at a time.

➤ To delete items:

1. Select the items you want to delete. You can select a folder to delete it and all its contents. You can select multiple items anywhere in the list using Shift+Click and Ctrl+Click.
2. Press **Del** or right-click and select **Delete item(s)**.
You are asked to confirm your deletion.
3. Click **Yes** to delete the items.

Using keyboard shortcuts The following keyboard shortcuts are supported in the Todo tab:

Keys	Description
Up Arrow	Move up one item
Down Arrow	Move down one item
Home	Move to first item in list
End	Move to last displayed item in list
Right Arrow	Expand item if on a collapsed folder, otherwise move to next item

Keys	Description
Left Arrow	Collapse item if on an expanded folder, otherwise move to parent
Enter	Toggle the expand/collapse state for item
+	Expand all items
-	Collapse all items
Ctrl+A	Select all items
Ctrl+/ Ctrl+\	Select all items
Ctrl+\	Deselect all items
Shift+Up Arrow	Extend selection up
Shift+Down Arrow	Extend selection down
Shift+Home	Extend selection to start of list
Shift+End	Extend selection to end of list
Ctrl+Up Arrow	Move focus up one item without changing selection status of items
Ctrl+Down Arrow	Move focus down one item without changing selection status of items
Ctrl+Space	Toggle selection status of item
Shift+Space	Select range of items from currently selected item(s) to item having focus
>	Indent selected items
<	Outdent selected items
Ins	Add item as sibling
Shift+Ins	Add item as child
Del	Delete selected items

Disabling the Todo feature If you don't want to use the Todo feature, you can disable it by deselecting **Enable Todo** in General Preferences (**Edit>Preferences**). With Todo disabled, the Todo tab does not display and the Edit>Add Todo Item menu item is disabled.

NOTE Even after disabling the Todo feature, your Todo list remains intact and will be displayed when you later reenables Todo.

Working with generated items

Various Workbench wizards and tools generate Todo items and add them to the corresponding project folder in your Todo list (remember that the Todo folder for a project is a top-level folder named *projectFile* in *pathToProject*). For example, the Servlet Wizard adds items about processing the servlet's GET and POST requests and implementing any interface stub methods.

@todo comments In addition to populating the Todo list with items, the wizards include @todo javadoc-style comments in the generated source files. These comments are of a finer granularity than the items generated in the Todo list. The Todo list would be too cluttered if all the @todo comments appeared in the list, but the @todo comments can be helpful to you in your detailed work.

Actions in generated items Generated items are just like the items you create in the Todo tab, with one exception:

A generated item might have an **action** associated with it. If a generated item has an action associated with it, you can invoke the action by doing either of the following:

- Double-clicking the item
- Right-clicking the item and selecting the first menu item, which describes the action

NOTE If an item has no associated action, the first menu item is **Launch Action** and it is disabled.

Typically, the action is to open an associated file. For example, if you double-click the Todo item generated by the Servlet Wizard about specifying the servlet's GET request, Workbench opens the servlet's source file and positions the insertion point appropriately.

Specifying a debugger

By default, when you select **Edit>Launch Debugger** or click the **Launch Debugger** toolbar button, Workbench opens a dialog asking you for information, then launches the SilverStream Debugger provided with Workbench. (For more information about using the SilverStream Debugger, see Chapter 12, “Debugger”.)

Instead of using the SilverStream Debugger, you can also specify your own debugger so that when you select **Edit>Launch Debugger**, your debugger is launched with the proper command-line arguments.

➤ To specify your own debugger to launch from Workbench:

1. Select **Edit>Preferences**.
2. In the **Debugger command** field in the General tab, specify the command line to launch your debugger, as described next.
If this field is not empty, Workbench executes the specified command when you launch a debugger. If this field is empty, Workbench launches the SilverStream Debugger.
3. Click **OK**.

Specifying the command

Specify the operating system command that Workbench should issue when you select **Edit>Launch Debugger** or click the **Launch Debugger** button. See your debugger’s documentation for information about how to invoke your debugger from the command line.

You can include environment variables in the command by using the syntax `%varname%` or `${varname}`. Workbench substitutes the values of these variables when invoking the command.

In addition to environment variables set at the operating system, you can also use environment variables that are predefined by Workbench:

- You can use the same variables that the Workbench version control interface uses (see “Predefined environment variables” on page 33).

The file-related environment variables (such as `_%_PATH%`) refer to the file that is open and currently active in Workbench.

- Plus you can use the following two predefined variables:

Variable	Description
%_CLASSPATH%	The semicolon-delimited list of the classpath entries for the project and its subprojects
%_SOURCEPATH%	The directory containing the project file

Using Ant

Internally, Workbench uses Apache Ant when you build a project by selecting one of the Build commands on the Project menu (for more information about building Workbench projects, see “Compiling, building, and archiving” on page 87). You don’t need to know anything about Ant if you only want to do builds from the Workbench IDE. But Workbench also provides direct access to Ant so that you can accomplish the following:

- Do project builds from the command line
- Do your own customized Ant processing

If you want to do either of these tasks, read this section to learn about Ant and how to use it.

What is Ant?

Apache Ant is a Java-based build tool, much like make but without make’s foibles. A couple of key differences between Ant and make are:

- Instead of using makefiles, Ant uses XML-based **buildfiles**, which specify **targets** that define the processing that you want.
- Instead of using shell-based commands, Ant is extended using **Java classes**. It comes with a built-in set of **tasks**, each implemented through a Java class. To define a new task, you define a new Java class that extends the Ant Task class.

Ant is an open-source Jakarta subproject. To learn more about Ant, including details on defining your own tasks and creating buildfiles, see <http://jakarta.apache.org/ant>.

Using the Workbench Ant tools

You can use Workbench tools to invoke Ant from the command line. There are two Ant-based executables in the Workbench bin directory:

- **xwbbuild**, which allows you to build a Workbench project
- **xwbant**, which allows you to perform customized processing based on buildfiles (and possibly task classes) that you have created

The difference between the two executables is that xwbbuild takes a Workbench project file as input, and xwbant takes an Ant buildfile.

You invoke the tools from the command line.

xwbbuild syntax Here is the command syntax for xwbbuild:

```
xwbbuild projectFile WorkbenchTarget options
```

where:

Argument	Description
projectFile	Path to the SilverStream project (.spf) file. This file specifies, among other things, the name of the Ant buildfile that builds and creates the archive(s) for your project.
WorkbenchTarget	Specify one of these project buildfile targets: <ul style="list-style-type: none"> • build—Builds and creates the archive(s) for the specified project (equivalent to selecting Project>Build and Archive) • rebuild—Rebuilds and creates the archive(s) for the specified project (equivalent to selecting Project>Rebuild All and Archive) • clean—Removes all files from the project's build directory and deletes the archive(s) (no equivalent in the Workbench IDE)
options	See below for information on the options.


xwbant syntax Here is the command syntax for xwbant:


```
xwbant CustomizedTargets options
```


where:

Argument	Description
CustomizedTargets	Specify one or more of the targets you have defined in your buildfile
options	See below for information on the options.

Options Here are the options you can provide with `xwbuild` and `xwbant`:

Option	Description
-help	Prints usage information
-projecthelp	Prints the description of the project (if one exists), followed by the targets defined in the buildfile
-version	Prints the version of Ant
-quiet	Be extra quiet
-verbose	Prints detailed information about the processing
-debug	Prints debugging information, including a mapping of tasks to Java classes and a listing of properties and their values
-emacs	(<code>xwbant</code> only) Prints logging information without adornments
-logfile <i>file</i>	Sends output to <i>file</i> , instead of to the screen. This option creates <i>file</i> if it doesn't exist or overwrites <i>file</i> if it does exist.
-logger <i>class</i>	Specifies the class to do the logging. The default logger is <code>org.apache.tools.ant.DefaultLogger</code> . You can also specify another built-in logging class (look in <code>ant.jar</code> in the Workbench lib directory for provided classes) or specify a logging class you wrote yourself.  See the Ant documentation at http://jakarta.apache.org/ant for details.

Option	Description
-listener <i>class</i>	<p>Adds <i>class</i> as a listener. A listener is notified when one of the following events occur:</p> <ul style="list-style-type: none"> • A build is started • A build is finished • A target is started • A target is finished • A task is started • A task is finished • A message is logged <p>There is no default listener. You can specify a built-in listener class (look in ant.jar in the Workbench lib directory for provided classes) or specify a listener class you wrote yourself.</p> <p> See the Ant documentation at http://jakarta.apache.org/ant for details.</p>
-D <i>property=value</i>	<p>Overrides property value set in the buildfile. Properties are defined as <property> elements in the buildfile.</p>
-buildfile <i>file</i>	<p>(xwbant only) Specifies the buildfile to use. If this option is not specified, Ant uses build.xml in the current directory.</p> <p>(This option applies only to xwbant, because xwbbuild always uses the project buildfile that Workbench creates for you automatically.)</p>
-find <i>file</i>	<p>(xwbant only) Searches for buildfile <i>file</i> starting at the current directory. If it doesn't find it in the current directory, it searches the parent directory, up to the root directory, until it finds <i>file</i>.</p> <p>If <i>file</i> is not specified, it searches for build.xml.</p>

Workbench modification to Ant The version of Ant shipped with Workbench is exactly the same as the version from Apache (use the -version command-line option to see the version number), with one exception: the Workbench Ant javac task supports a **sourcepath** attribute, which allows you to specify the -sourcepath argument to the compiler. Workbench generates buildfiles with this attribute, so Workbench buildfiles will not work with an unmodified Ant.

Examples

xwbbuild examples The following command builds and creates the archive(s) for the myApp Workbench project (if changes have been made since the last time the project was built and archived).

```
xwbbuild myApp.spf build
```

The following command rebuilds all the files and creates the archive(s) for the myApp project.

```
xwbbuild myApp.spf rebuild
```

The following command removes all files from the build directory and deletes the archive(s).

```
xwbbuild myApp.spf clean
```

xwbant examples The following command performs the tasks defined for the default target in build.xml in the current directory.

```
xwbant
```

The following command performs the tasks defined for the purge target in build.xml in the current directory.

```
xwbant purge
```

The following command performs the tasks defined for the purge target in test.xml. If test.xml isn't found in the current directory, Ant searches for it in parent directories until it hits the root directory.

```
xwbant purge -find test.xml
```

Internationalization support

This section describes Workbench's support for internationalization.

Specifying fonts

If some international characters are not displaying correctly in Workbench (for example, they are displaying as boxes) or if the font mapping on your system is poor, you can specify different fonts for Workbench to use for its menus, labels, dialogs, and so on (note that the editors themselves are not affected by changes you make as described next).

➤ To change the fonts used by Workbench:

1. Exit Workbench.
2. Specify alternate font names (and optionally sizes and colors) in the following lines in **ide.props**, which is in the Workbench Resources\Preferences directory:

```
font-name-standard = font-name
font-size-standard = font-size
font-name-big = font-name
font-size-big = font-size
output-font-name = font-name
output-font-size = font-size
output-background-color = font-color
output-font-color = font-color
```

Font sizes are specified in points. Colors are specified as R,G,B integer values; for example, 255,255,255 is white and 0,0,0 is black.

- The **standard font** is used to display all standard-sized text, menus, labels, and so on. The default is 11-point Arial.
- The **big font** is used to display title text in wizards as well as buttons in wizards and dialogs. The default is 18-point Arial.
- The **output font** is used to display text in the Output Pane. The default is 12-point Monospaced, black on a white background.

Sun recommends that you use **Serif** as the font name to provide the best font mapping on most systems.

Extending the Workbench toolset and services

SilverStream eXtend Workbench is an extensible IDE for developing J2EE applications and Web Services. The standard toolset described in this documentation can be extended using the Workbench framework API. Contact your SilverStream representative for more information about extensibility.

2

Projects and Archives

SilverStream eXtend Workbench helps you create J2EE components (including EJB JARs, JSP pages, servlets, and Java class files) to produce well-structured J2EE archives. Your work in Workbench is organized into **projects**.

Working in a project involves editing sources (such as Java and data files), building classes, generating the archive, and deploying the archive. This chapter describes how to work with projects to create and manage J2EE components and archives. It includes the following sections:

- About projects and archives
- Organizing projects
- Creating projects and subprojects
- Populating projects
- Viewing projects
- Maintaining projects
- Compiling, building, and archiving
- Validating archives

About projects and archives

A **project** is a collection of source files that you work with in Workbench to create J2EE modules. A project can also be thought of as a series of rules that define how parts come together to create an archive.

An **archive** is what gets generated from a completed project. A Workbench project can represent any of the following types of archive:

- Enterprise archive (EAR)
- Web archive (WAR)
- Application client archive (JAR)
- EJB archive (JAR)
- Java class archive (JAR)
- Resource adapter archive (RAR)
- Deploy-only (non-buildable) archives

Workbench does not limit you to creating J2EE projects and archives. You can also develop and build nonarchive projects (projects that simply build other files) and utility projects (such as class files stored in a ZIP or JAR file) using Workbench.

What a project includes A project can include:

- Source code that will be compiled (the resulting files will be put into an archive)
- Content files that you put directly into the archive (JSP pages, HTML pages, images, and so on)
- A deployment descriptor for the project archive
- Server-specific deployment information
- Other project files, called **subprojects**

Project file Each project and subproject has a SilverStream **project file** (SPF file) that defines it. Workbench automatically creates this project file to store settings that you specify in Workbench. The project file defines how the project references subprojects, where files are stored on disk, and how files will be structured in the generated archive—and stores classpath entries and deployment settings. Changes you make to a project are automatically reflected and saved in the project file. When you add or move a component in a subproject, the change is updated in the subproject's project file.

CAUTION *There is no reason to directly edit the project file. All settings can be defined within Workbench. If you manually change the file incorrectly, you could compromise your ability to open the project associated with that file in Workbench.*


Organizing projects

When you create a project, you must specify what directories (or files) in your file system are to be included in the project and where to save the Java archive that is to be built by the project.


You must also decide how to structure subprojects within a project. For example, a top-level EAR project might contain various subproject modules such as WARs and EJB JARs that define an application's user interface, business logic, database access, and so on.

Project design considerations

J2EE and Web Service applications can be extremely complex, with many project design issues to consider. Your design decisions affect how to create the projects, subprojects, and components in Workbench that make up your application.

 For more information about design considerations for J2EE and Web Service applications, see the chapter on developing applications in the *Development Guide*.

Workbench supports almost any method for creating projects and components, including bottom-up (creating components first and then Workbench projects and subprojects) and top-down (creating projects and subprojects first and then components). In most cases, you should follow a top-down approach—first create the project and subproject structure and then create new components and add them (and any existing components) to your project.

 For information about creating an entirely new project, see “Creating projects and subprojects” on page 6. For information about creating a project that contains existing source files and components, see “Working with existing source files” on page 14.

Project directory structure considerations

Workbench provides a lot of flexibility in defining the directory structures for your project's source files and the archive built from those source files.

Directory structure of your source files The directory structure of the source files on your file system does not need to match the directory structure of the generated files in the archive. For example, files in different source directories can be assigned to the same directory in the archive. To simplify development, however, you may want to set up your project directories to mimic the directory tree structure that will group J2EE components into archives.


You could create your project source file directory structure so that the project (SPF) file is located at the root of that directory structure and then create a project **src** directory (at the same level as the project file) in which you can place all of the project source code. For example:

```
myWebProject\  
  myProject.spf  
  src\  
    dbAccess\  
      addItem.java  
      changeItem.java  
      deleteItem.java  
      queryDB.java  
    loginProcessing\  
      login.java  
      user.java  
    userInterface\  
      intro.jsp  
      login.jsp  
      loginError.jsp  
      welcome.jsp
```

When creating an enterprise archive (EAR) project with multiple subprojects (JARs, WARs, EJB JARs, and so on), it may be easiest to have all the project files at the same level, and have the sources of each subproject in separate **src** subdirectories. For example:

```
myWebProject\  
  myProject.spf  
  myProjectDB.spf  
  myProjectLogin.spf  
  myProjectUI.spf  
  src\  
    dbAccess\  
      addItem.java  
      changeItem.java  
      deleteItem.java  
      queryDB.java  
    loginProcessing\  
      login.java  
      user.java  
    userInterface\  
      intro.jsp  
      login.jsp  
      loginError.jsp  
      welcome.jsp
```

If your project package structure becomes too cumbersome, you can always move the subproject components into separate subdirectories. You can structure Workbench projects using a single or a combined source tree.

 For more information on project settings, see “Managing project content settings” on page 30.

Directory structure of an archive The internal directory structure of your J2EE archive depends on the archive type. Each type of archive has an XML descriptor that conforms to a particular DTD.


For example, when creating a Web archive (WAR), you must specify which files are accessible directly through an URL (such as JSP pages and servlets) and which files are not (such as supporting class and archive files). J2EE specifies that you locate files that are not to be made accessible through an URL in a WEB-INF directory in the archive directory structure. This WEB-INF directory should be located beneath the archive root directory and typically includes:


File or directory	Contents
web.xml	A required deployment descriptor file that tells the application server how to interact with the Web application
WEB-INF/classes/	A directory containing the compiled Java class files for the application
WEB-INF/lib/	A directory containing the JAR files used by the application

The JSP pages that are URL-accessible typically are located in the root directory of the archive. You may want to hide some JSP pages (such as those used by Struts) from URL access. Files under the WEB-INF directory are by default not accessible via URL, although you can configure them for URL access. The locations of other files are up to you.

CAUTION *When you create the WEB-INF directory, you must ensure that the directory name is in all uppercase text.*

For more information This section has provided only a glimpse into some of the issues you may encounter when designing your source file and archive directory structures.


 For more information about specifying archive directory structure and packaging archives, see the Sun J2EE Blueprints document.

 For information about how you can specify source and archive directory structures in Workbench, see “Managing project content settings” on page 30.

Creating projects and subprojects

A project (or subproject) can be an EAR, EJB JAR, WAR, RAR, JAR, deploy-only archive, or application client. As you create a project, you define a project name and a location for your source files. Workbench maps all of these source files to names and locations that you define for the archive.

The following steps (using the New Project Wizard) apply to each type of project, with exceptions noted.

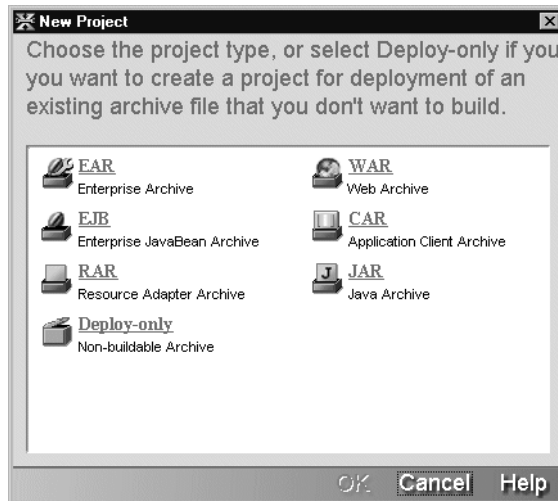
 For information on organizing the development workspace before beginning a project, see “Project directory structure considerations” on page 3.

➤ To create a project:

NOTE If you are creating a subproject, you must open the parent project in Workbench before starting this procedure.

1. Select **File>New Project**.

The New Project dialog appears:



2. Choose a project type and click **OK**.
 - If you want to create a nonbuildable archive, select **Deploy-only**. For detailed instructions, see “Creating a deploy-only project” on page 12.

- If you are creating an EJB JAR and EJB client JAR pair, you should first create the parent WAR or EAR for both so that both projects can be open at once. For detailed information on the relationship between the EJB JAR and the client JAR, see “Specifying the EJB JAR configuration” on page 125.
- If you want to create a project that includes a completed archive (along with its source code) from a third-party source, choose a project type and follow the instructions in “Working with existing source files” on page 14.

NOTE The following New Project Wizard panels (to create a WAR) apply to each type of project.

New Project

Enter the name and location (directory path) for the project, the archive file, and the deployment descriptor (if relevant). If this is a J2EE project, select the desired J2EE version. (To use an existing archive as-is, create a deploy-only project instead.)

Project Name:
Forecast

Project Location:
c:\WorkbenchProjects\Forecast

Archive Name (e.g. office.war):
Forecast

Archive Location (directory):
c:\WorkbenchProjects\Forecast


Deployment Descriptor Name:
web.xml


Deployment Descriptor Location:
c:\WorkbenchProjects\Forecast\WEB-INF



Project J2EE Version: j2ee 1.3 (war 2.3)

< Back Next > Cancel Help

3. Specify project information as follows:

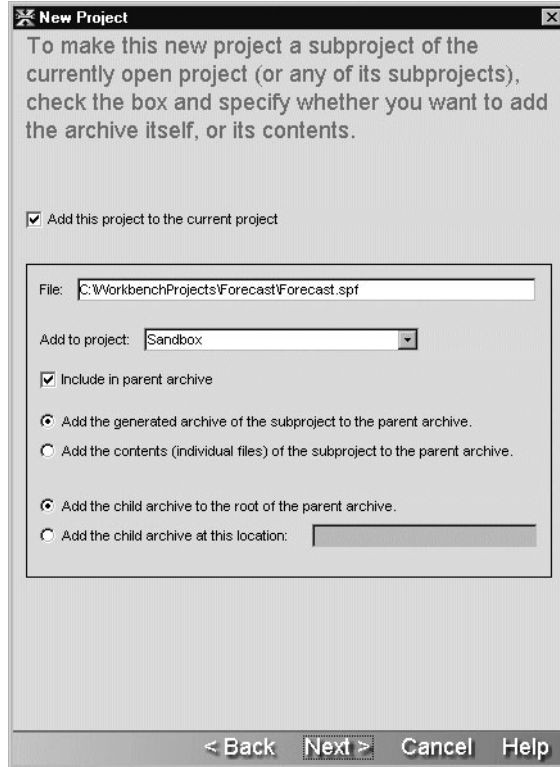
New project setting	What you do
Project Name	<p>Specify the name you want to use for the project (the .SPF extension is automatically appended). This name appears in the Source Layout.</p> <p>As you enter a project name, the archive name is filled in automatically. You can keep the same name for your archive or enter another one.</p>
Project Location	<p>Specify the directory where you want the project (and other source files) to be located. Workbench creates a SilverStream project file (with an .SPF extension) in the project location.</p> <p>As you enter a project location, the rest of the new project settings are filled in automatically. You can change these settings.</p> <p>You can click the ellipses beside the Project Location field to select a location, or type the project directory.</p> <p>If you specify a project location directory that does not exist, the wizard prompts you to create it.</p> <p>If you do not specify an absolute path, the wizard locates the project under the Workbench bin directory.</p>
Archive Name	<p>Specify the name of the archive file that will be generated. The resulting name will appear in the Archive Layout. An extension based on the archive type will automatically be appended to the name. You can keep the default archive name (that matches the project name) or enter a new one.</p> <p>To create a project based on an existing archive or to create a deploy-only project, enter the name of the existing archive that you want to include.</p> <p> For information about creating projects based on existing archives, see “Working with existing source files” on page 14. For details about deploy-only archives, see “Creating a deploy-only project” on page 12.</p>

New project setting	What you do
Archive Location	<p>Enter the location of the project archive or accept the default (the project root directory).</p> <p>The archive location appears in the Archive Layout of the Navigation Pane after the project has been created.</p>
Deployment Descriptor Name	<p>The wizard fills in a deployment name (based on the project type) after you enter a project location. Each archive stores its own set of deployment information in this XML deployment descriptor file source file. Workbench creates the default deployment descriptor name and location (based on archive type) when you build and archive the project.</p> <p>In most cases you should accept the default name and location.</p> <p>If you are converting an existing archive project (by creating a Workbench project file), enter the name of the deployment descriptor file on disk.</p> <p>If you want to have multiple J2EE subprojects of the same type sharing the same deployment descriptor directory location, see Deployment Descriptor Location (just below).</p> <p>The deployment descriptor name you enter here affects only the source file name—not the file name that is used in the JAR. When Workbench builds the archive, it includes this deployment descriptor file in the archive using the standard location defined by the J2EE specification for the archive type.</p> <p> For more information on deployment descriptor names, see Chapter 3, “Archive Deployment” and Chapter 10, “Deployment Descriptor Editor”.</p>

New project setting	What you do
Deployment Descriptor Location	<p>Enter the location of the deployment descriptor or accept the default. Each archive type uses a required J2EE default directory location.</p> <p>If you are converting an archive project, enter the location of its deployment descriptor.</p> <p>In most cases you should accept the default name and location. However, if you want to have multiple J2EE subprojects of the same type sharing the same deployment descriptor directory location, you should either enter a different source file name for each deployment descriptor or create a separate directory structure beneath the root directory for each descriptor.</p> <p>If you specify (or if Workbench finds) a deployment descriptor in the project source location matching the one you specify, it prompts whether or not you want to use the existing deployment descriptor. If you answer no, you will need to change the deployment descriptor name or location before continuing.</p> <p> For more information on deployment descriptor default names and locations, see Chapter 3, “Archive Deployment” and Chapter 10, “Deployment Descriptor Editor”.</p>
Project J2EE Version	<p>Specify the version of J2EE for this project.</p> <p> For information on targeting your application at an appropriate version of J2EE, see the chapter on handling J2EE versions in <i>Getting Started</i>.</p>

NOTE All settings on this wizard panel are required—except the two deployment descriptor fields and the J2EE version, which are not required (or displayed) for the Java or deploy-only archive.

4. Click **Next**.
5. If you have a project currently open in Workbench, the wizard asks if you want to add the new project as a subproject to that project or one of its subprojects.



If no project is currently open in Workbench, this panel does not appear.

If you do not want to create this project as a subproject, deselect **Add this project to the current project** and click **Next**. (You can proceed to Step 6.)

To create the project as a subproject of another project:

- Select **Add this project to the current project**.
- Select the parent project under **Add to Project**. This list contains the currently open project and all subprojects associated with it.
- Select **Include in parent archive**.
- If you want to add the generated archive of this project to the parent archive (as opposed to adding all of the generated files), select **Add the generated archive of the subproject to the parent archive**.

If you want to add the generated files (instead of the generated archive) of this project to the parent archive, select **Add the contents (individual files) of the subproject to the parent archive**.

- The wording of the next two options vary, depending on whether you choose to add the archive or the individual files to the parent archive.

In either case, you are asked whether to add the archive or files to the root of the parent archive or to specify some other location in the parent archive.

- Click **Next**.

6. The wizard summarizes the project details. Click **Finish** to create the project.

You can see the new project in the Project tab in the Navigation Pane. If necessary, you can view or change project names and locations using the Project Settings dialog.

Once you have defined how your Workbench projects and subprojects will be structured, you can start adding source directories and files to a project, as described in “Adding to projects” on page 18.

Creating a deploy-only project

Workbench allows you to validate and deploy an archive for which you have no source code by first creating a deploy-only project for the archive.

For example, if you received a completed EJB JAR archive from a third party without any source code, you could create a deploy-only project for it. You cannot add to a deploy-only project.

NOTE If you receive a completed archive along with its source code, you should create a regular Workbench project, **not** a deploy-only project.

An EAR can contain both deploy-only and regular projects. For example, you can create an EAR containing an EJB JAR that you don't have the source for and a regular WAR that calls that EJB JAR.



For more information, see “Working with existing source files” on page 14.

How you tell that a project is deploy-only When you open a deploy-only project:

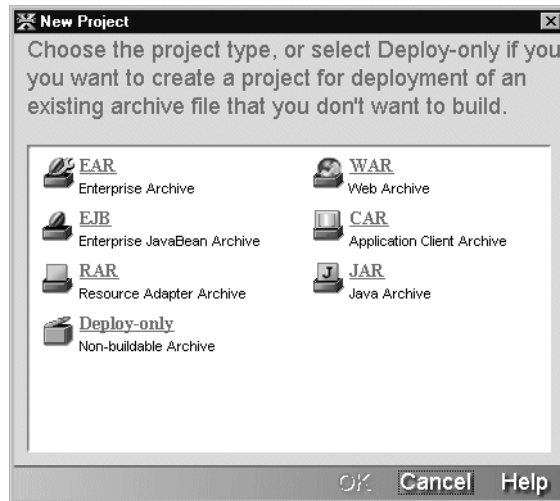
- The build commands on the **Project** menu are disabled. This prevents you from accidentally overwriting the archive—which you would be unable to recreate.
- The **Contents** tab of the Project Settings dialog is replaced by the following message:

The archive is deploy only. Its contents cannot be modified or examined.

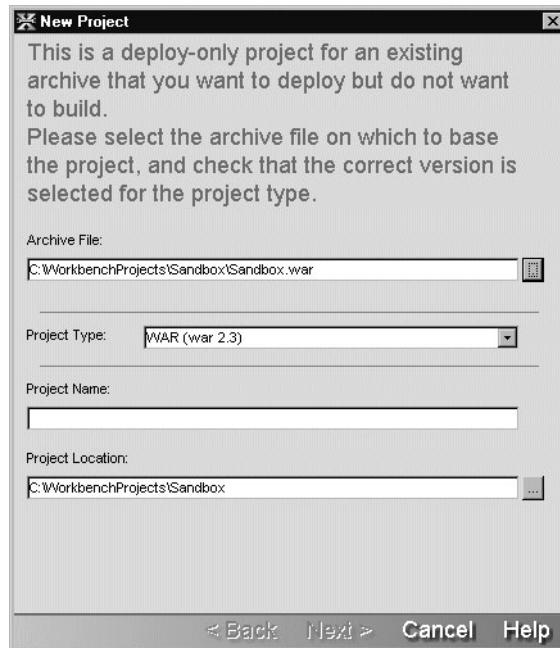
➤ To create a deploy-only project:

1. Select **File>New Project**.

The New Project dialog appears:



2. Select **Deploy-only** and click **OK**.



3. In the New Project Wizard, specify project information as follows:

Deploy-only project settings	What you do
Archive File	Enter or browse to the deploy-only archive file on which you wish to base the project. By default, the Project Location is set to the directory of the specified file when you select the archive.
Project Type	Make sure the archive type and J2EE version are correct.
Project Name	Enter a name to identify the deploy-only project. Workbench creates a SilverStream project file (with an .SPF extension) in the project location.
Project location	Specify where you want the project to be located. The location identifies the project root directory. The Project Location is set to the directory of the specified file when you select the archive, but you can change it.

4. Click **Next**.
5. If you have a project open, the wizard asks if you want to add the new project as a subproject of that project or one of its subprojects. For more information, see Step 5 under the preceding procedure for creating a project.
Otherwise, the wizard summarizes the project details.
6. Click **Finish** to create the project.

You can see the new project in the Project tab in the Navigation Pane, but you cannot edit its contents. If necessary, you can view or change project names and locations using the Project Settings dialog.

Working with existing source files

There are several ways you can use J2EE components and modules created with third-party tools in Workbench:

- If you want to create a nonbuildable archive that you validate and deploy in Workbench, you must create a deploy-only archive (a completed archive without any source code). For detailed instructions, see “Creating a deploy-only project” on page 12.

- If you have source code files and want to build an editable archive, you should create a regular Workbench project file as described below.

The following procedure describes one (directory-centric) approach where the resulting archive structure mirrors the directory structure of the source files. You can create a new project in a deploy-only archive using this same approach. The only difference is that you will not be able to later add source files to this type of archive.

➤ **To create a project that includes existing source files:**

1. Create a source directory structure and locate all your source files there.
When including existing archives, you may want to add the entire directory structure, since it is easier to maintain your project source files if you add directories rather than individual files. Once you have set up a project directory, files you add to it later will be automatically included in the resulting archive.
2. Create a Workbench project, as described under “Creating projects and subprojects” on page 6.
3. Add the source directory you created in Step 1 to the project, as described under “Adding to projects” on page 18.

Once you have added the source directory to the project, any changes you make later are automatically included in the archive and you avoid possible duplication of files.

Populating projects

Once you have a project, you can add components and subprojects to it. How you begin a project depends on whether you are creating a completely new project (no files, directories, or modules exist yet) or bringing existing J2EE components (created using an external IDE) into Workbench so you can add them to the project and deploy the archive.

 For more information, see “Project design considerations” on page 3 and “Working with existing source files” on page 14.


Creating source files

As you create source files in Workbench, you can group them under whatever project directories you want. You can create the source file and then add it to a project or open a project and then add the source file. Your project settings specify where files are located in the archive.

Source files include:

- Source code, such as Java files that will be compiled into an archive
- Content files that will be put directly into the archive, such as JSP pages, XML files, HTML pages, images, and so on

Typically, you want to create a directory and add that to the project before creating source files. When you add a directory to the project, any source files you create in that directory and in its subdirectories are automatically added to the project.

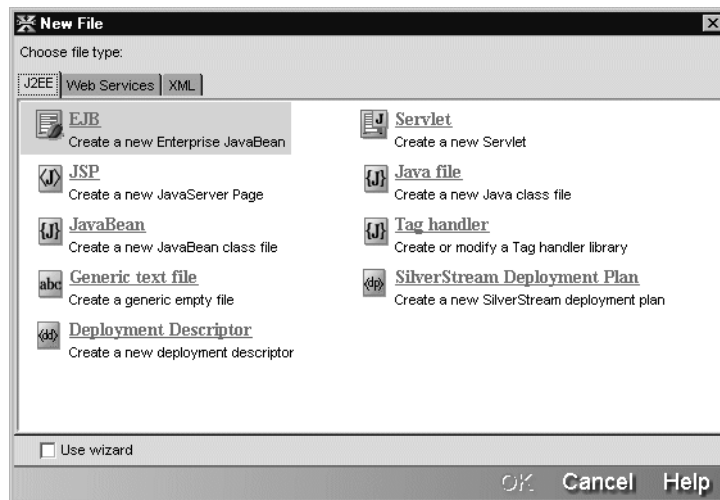
 For details about adding directories to a project, see “Adding an entire directory” on page 21.

When creating source files using the component wizards, any directories you specify are automatically added to the project.

➤ To create a source file:


1. (Optional) Open the project you want to add the file to.
2. Select **File>New**.

The New File dialog appears.





3. On the J2EE tab, choose a file type and click **OK**. The wizard for that file type starts. The wizards have built-in J2EE logic that facilitates the creation and deployment of well-structured J2EE components.

When the wizard finishes, a source editor containing the wizard-generated file opens in the Edit Pane.

 For more information see:

- “EJB Wizard” on page 117
- “Servlet Wizard” on page 161
- “JSP Wizard” on page 156
- “Java Class Wizard” on page 167
- “JavaBean Wizard” on page 172
- “Tag Handler Wizard” on page 176

 For information about source editors in Workbench, see Chapter 6, “Source Editors”.

 For information about the deployment-related wizards, see Chapter 10, “Deployment Descriptor Editor” and Chapter 11, “Deployment Plan Editor”.


 For information about the Web Service-related wizards, see Chapter 5, “Web Service Wizard” and Chapter 8, “WSDL Editor”.

 For information about the XML-related wizards, see Chapter 7, “XML Editors”.

➤ **To create a source file without using a wizard:**

1. (Optional) Open the project you want to add the file to.
2. Select **File>New**.
The New File dialog appears.
3. On the J2EE tab, choose **Generic text file** or **Java file**.
4. Deselect **Use wizard** and click **OK**.

A text editor containing a blank file opens in the Edit Pane.

 For information about source editors in Workbench, see Chapter 6, “Source Editors”.


Adding to projects

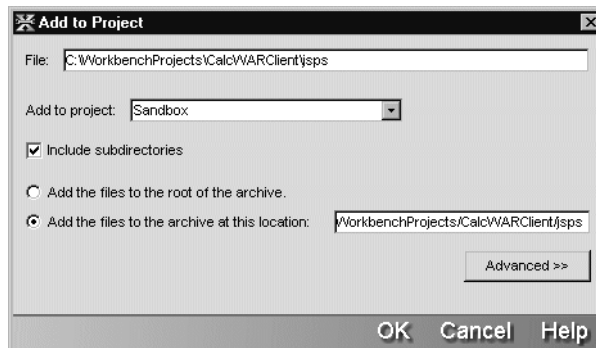
You can add source files, directories, and subprojects to an existing project.

Adding source files to a project


The following procedure describes how to add files and directories to a project.

➤ **To add files and directories to a project:**

1. Open the project you want to add to.
2. Select **Project>Add to Project**.
 For other methods, see “Other ways to add files and directories to a project” on page 20.
3. Choose whether you want to add a file or directory.
Typically, you add directories to your project rather than individual files.
4. Navigate to and choose the file or directory you want to add.
5. Click **Open** or **OK**.



6. Set the following options to specify how the file or directory will be added to the project and where you want it to be located in the archive:

File and directory setting	Description or action
File	Shows the (editable) path of the directory or file that you are adding to the project.
Add to project	Select the project that the specified item will be added to. Only the top-level open project and associated subprojects appear on the menu.
Include subdirectories	When adding directories, select to add the contents of the subdirectories as well as those of the specified directory.
Add the file(s) to the root of the archive	Select to add the specified files to the root of the archive. Clicking this option means you cannot remove the contents from the project without manually deleting the contents from the file system.
Add the file(s) to the archive at this location:	<p>Select to add the specified item to a specified location other than the archive root.</p> <p>You can also use relative paths or environment variables when locating shared project files or referring to files located outside the project's directory structure.</p> <p> For more information, see “Using environment variables” on page 30 and “Using relative paths” on page 31.</p>

7. If you are adding a directory to your project, click **Advanced**.

The following project entry settings let you specify how to include Java sources (of the files or directories) in the generated archive.

Advanced setting	Description or action
Include Java source files in archive	Select if you want to include sources files in the generated archive. For most production environments, you will not want to include Java source code.
Add the files to the root of the archive	Select to store source files in the archive root directory. Clicking this option means you cannot remove the contents from the project without manually deleting the contents from the file system.
Add the files to the archive at this location	Select and then specify a directory in which to store source files.

NOTE You can also edit these project entries in the **Edit archive entry** dialog (by clicking **Edit** in the Contents tab of the Project Settings dialog).

8. Click **OK** to add the file or directory to the project.

To see (or edit) how contents have been added to your project, click the **Contents** tab of the Project Settings dialog.



For information about editing project contents, see “Managing project content settings” on page 30.

Other ways to add files and directories to a project Using **Project>Add to Project** is only one way to add files and directories to a project. Other ways include:

- Clicking **Add Entry** or **Add Directory** on the Contents tab of the Project Settings dialog
- Using the popup (right-mouse) menu on the file or directory you want to add in the Directory tab of the Navigation Pane and choosing **Add to Project**.

Using this technique you can add multiple files at the same time: press **Ctrl+Click** to add multiple noncontiguous files; press **Shift+Click** to add multiple contiguous files.

Notes about adding individual files You typically add entire directories to your project. However, you can also add:

- The entire contents of a directory
- Individual file(s) in a directory

If you add a subproject as contents rather than as an entire project to a top-level project, the name will appear grayed out and within parentheses in the Archive Layout view of the Navigation Pane.

Refreshing the Navigation Pane Workbench automatically updates the contents of the Directory and Project tabs in the Navigation Pane when you make changes in Workbench. If you make changes outside Workbench, select **View>Refresh** or press **F5** to see the changes.

Adding an entire directory

When you add a directory or directory tree to a project (as described in “Adding source files to a project” on page 18), the structure of the files and directories in the archive matches the layout of the files and directories of your (on-disk) source directories.

When you specify an entire directory, anything you later change, add, or remove within that on-disk directory is automatically reflected in the project. To relocate archive files, you can simply move them from the existing source directory structure on your file system. Any such changes will be automatically reflected in your project, provided that you keep them within the source directory structure used by the project.

What gets excluded When you add the entire directory to a project, Workbench excludes the following types of files from the generated archive:

- Any file ending in **~**
- Any file starting and/or ending with **#**
- Any file starting and/or ending with **%**
- Any file named **cvsignore**
- Any files in a directory named **CVS**
- Any files ending in **JAVA** (by default, though you can choose to include Java files when adding the directory)
- Any autosave and backup files ending in **SAV** and **BAK**

These are generally backup or version control information files and don't belong in the generated archive.

Adding subprojects to a project


The following procedure describes how to add a subproject to a project.



For details about creating subprojects, see “Creating projects and subprojects” on page 6.

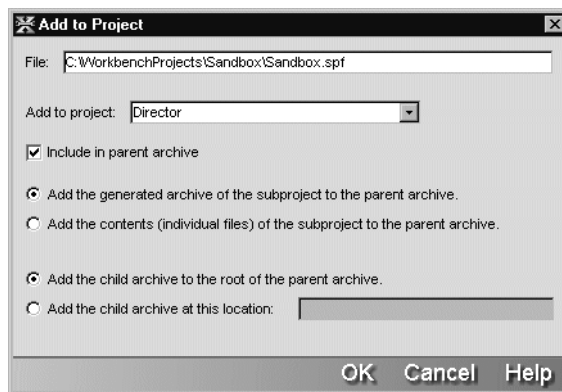
➤ **To add a subproject to a project:**

1. Open the project you want to add to.
2. Select **Project>Add to Project>Subproject**.

 For other methods, see “Other ways to add files and directories to a project” on page 20.

A file selection dialog appears.

3. Navigate to and choose the subproject file you want to add.
4. Click **Open**. The Add to Project dialog appears.



5. In the **Add to project** field, select the project that the specified archive will be added to.

NOTE Only the top-level project and any associated subprojects appear as choices.

6. Select **Include in parent archive** to add the contents of the subproject to the parent archive.


If **Include in parent archive** is not selected, the subproject will still be built before the parent project, but none of its contents will be included in the parent archive.

7. If you want to add the generated archive of this project to the parent archive (as opposed to adding all of the generated files), select **Add the generated archive of the subproject to the parent archive**.

If you want to add the generated files (instead of the generated archive) of this project to the parent archive, select **Add the contents (individual files) of the subproject to the parent archive**.


8. The wording of the last two options vary, depending on whether you choose to add the archive or the individual files to the parent archive.

If you selected **Add the generated archive of the subproject to the parent archive**, set one of the following options to determine how the specified archive will be added to the parent archive:

Subproject setting	Action
Add the child archive to the root of the parent archive	Select to add the specified archive to the root directory of the parent archive.
Add the child archive at this location	<p>Select (and enter a location) to add the specified archive to a location other than the root directory of the parent archive.</p> <p>You can also use relative paths or environment variables when locating shared project files or referring to files located outside the project's directory structure.</p> <p> For more information, see “Using environment variables” on page 30 and “Using relative paths” on page 31.</p>


If you selected **Add the contents (individual files) of the subproject to the parent archive**, set one of the following options to specify how the subproject contents (rather than the subproject's generated archive) will be added:

Subproject setting	Action
Add the files to the root of the parent archive	Select to add the archive contents to the root directory of the parent archive.

Subproject setting	Action
Add the files to the archive with this prefix:	Select and then enter a prefix to add the archive contents to a directory with the specified prefix. You can also use relative paths or environment variables when locating shared project files or referring to files located outside the project's directory structure.  For more information, see "Using environment variables" on page 30 and "Using relative paths" on page 31.

9. Click **OK** to add the child archive (or files) to the parent archive.

TIP To see how contents have been added to your project, click the **Contents** tab of the Project Settings dialog.

 For more information about adding project contents, see "Modifying project entries" on page 31.

Viewing projects

You use the Project tab in the Navigation Pane to view projects. You can view projects in three ways to see how directories and files are organized on the file system and in the archive:

- Source Layout view
- Archive Layout view
- Archive Contents view

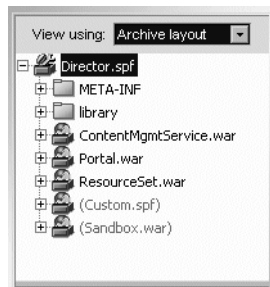
Source Layout view The **Source Layout view** reflects the organization of the project's files and directories on your hard disk. Subprojects are listed at the top level as folders.

- Subprojects added as archives and as contents are both shown using the project name
- Subprojects excluded from the parent archive are shown grayed out using the project name

The Archive views The **Archive Layout view** and **Archive Contents view** both reflect the organization of the archive that will result from building the project. The Archive Layout view presents a development-oriented picture of how the project files and directories will be organized in the resulting archive, while the Archive Contents view is the closest representation of what will be in the generated archive. The differences between the two views are:

- Archive Layout shows the project's source files (.java files), even though the archive actually contains compiled files (.class files). Archive Contents shows compiled files since they are what is in the archive (double-clicking a .class file opens the corresponding source file in the Java Editor for editing, unless the .java file can't be found, in which case the .class file is opened in the Class Viewer).
- Archive Layout lists all subprojects, even those subprojects that have been added as contents (as opposed to being added as archives) and those subprojects excluded from the parent archive, to give you an idea of how the projects are organized.
 - Subprojects added as archives are listed as the archive that they generate
 - Subprojects added as contents are displayed grayed out using the name of the project
 - Subprojects excluded from the parent archive are shown grayed out, using the name of the archive or the project name, depending on which state would result from reincluding the subproject in the parent archive

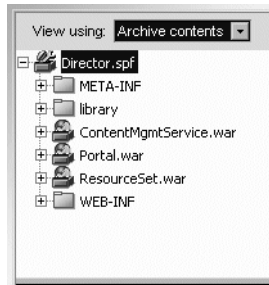
In the following screen, the ResourceSet subproject has been added as an archive, so displays with its archive name. The Custom subproject has been added to the project as contents, so is grayed out. The Sandbox subproject has been excluded from the parent archive, so it too is grayed out.



Archive Contents lists subprojects as follows:

- Subprojects added as archives are displayed using archive names
- Subprojects added as contents are represented by the content itself, since that is how they will appear in the parent project's archive
- Subprojects that are excluded from the parent archive are not represented at all

The following screen shows the Archive Contents view of the same project shown above.




- Archive Contents shows each inner class in a .java file, in addition to the file's primary class.

TIP You can see a file's complete name and path by positioning the mouse over it in the lower subpane of the Directory or Project tab. Workbench tool tips are particularly useful in an Archive view for comparing a file as it exists in the archive (such as WEB-INF/web.xml) to its location on disk (such as C:\dev\Aries\web.xml).

Maintaining projects

Your open project may be your top-level project or it may be a subproject. Workbench allows you to manage the settings of any open project by adding or modifying files, directories, subprojects, paths, classpaths, and so on. You can modify a project by:

- Opening a project
- Managing general project settings
- Managing project content settings
- Removing files, directories, and subprojects from projects
- Renaming a project

 For more information on team development and design considerations, see the *Development Guide*.

Opening a project

To open a project, open the project file (with the .SPF extension). Changes you make to a top-level project file are automatically saved in that file along with any other subprojects that are part of the same top-level project.

You can open multiple projects at a time, as long as they are all part of the same top-level project. For example, you can simultaneously open an EAR, a WAR, an EJB JAR, and a application client provided they are all part of the same top-level EAR.

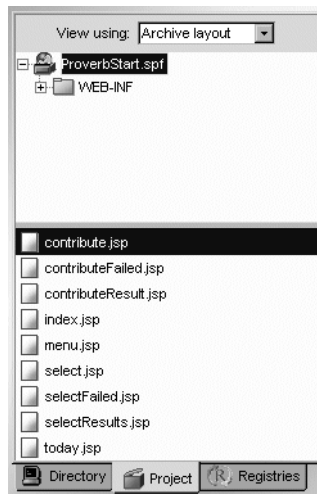
NOTE Whenever you add a component or subproject, the project file is automatically saved. The only time you need to explicitly do a save is when you make changes to a source file using one of the editors.

➤ To open a project:

1. Select **File>Open Project**.
2. Navigate to the project directory.
3. Select the project file (.SPF) and click **Open**.

In the upper left, the Navigation Pane displays the Archive Layout of the project. The files are displayed in the lower subpane of the Navigation Pane.

TIP You can also navigate to the project file in the **Directory** tab and double-click the file to open it.

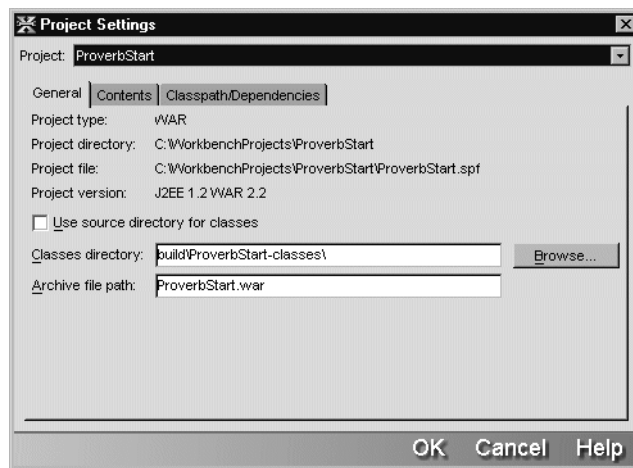


Managing general project settings

The General tab on the Project Settings dialog lets you view information about the open project and change the location of the source directory that stores the project class files.

➤ To view or modify project settings:

1. Open the project.
2. Choose **Project>Project Settings**.
3. Select the **General** tab and view or modify any of the options as follows:

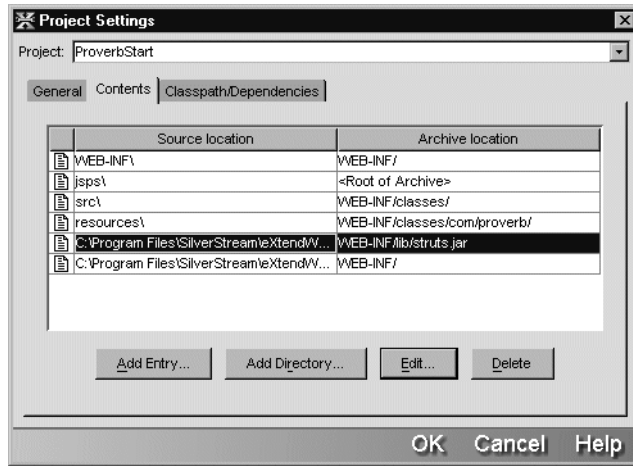


Setting	Description or action
Project	Lists the project currently open.
Project type	Lists the type of project you created.
Project directory	Lists the open project's root directory.
Project file	Lists the file name and location of the open project.
Project version	Lists the J2EE version

Setting	Description or action
Use source directory for classes	<p>Specifies whether you want to compile Java files into the same directory as their corresponding source files.</p> <p>By default, the check box is not selected and classes are compiled into the build directory beneath the project's root directory. You can change the build directory by changing the Classes directory setting (below).</p> <p>If you select the check box, all project classes are generated into the source directory along with with their source files.</p>
Classes directory	<p>Lists the root of the build directory where the project's compiled class files will be located. Workbench writes the generated classes to this directory when it builds the archive.</p> <p>The default is build/project_name-classes beneath the project's root directory. You can change the directory by typing or browsing to a different directory.</p>
Archive file path	<p>Lists the name and location of the archive. Workbench writes the generated archive to this location, which is relative to the project root.</p> <p>You defined this location when you created the project.</p>

Managing project content settings

You specify how files and directories are organized in the project's Source and Archive Layouts using the Contents tab of the Project Settings dialog.



This dialog lets you define files and directories in terms of project entries in a table. Each entry defines the location of a source file or directory in the file system and how it is added to the project archive.

When specifying file and directory locations, you can use environment variables and absolute and relative pathnames.

Using environment variables

Windows environment variables are useful when a development team shares files (such as a single project file or JARs) that are located outside the project's directory structure. A shared project file must be able to refer to files or directories that exist in different locations on different team members' machines. You typically use environment variables in Workbench for locating files that are not under the project's root directory.

You set the environment variables by using the Environment tab in the Windows System control panel. You reference the variables in Workbench using the following syntax: **%varname%** or **\${varname}**. You can use Workbench variables:

- When editing or adding to a project using the Add to Project dialog

For example, change `d:\utilproj\util.spf` to `%UTIL_PROJECT_DIR%\util.spf` or `${UTIL_PROJECT_DIR}\util.spf`

- When editing your project's classpath using the Classpath/Dependencies tab of Project Settings dialog

For example, add `%UTIL_PROJECT_DIR%\util.jar` or `${3RDPARTYJARS}\helpers.jar` to the classpath to include a subproject.

NOTE You need to restart Workbench before the value of an environment variable (set in the Windows Control Panel) takes effect.


It may often be easier to use a relative path (instead of an environment variable) to locate any shared project files that are in the project's directory tree. For example, you could specify a **src** directory to refer to the directory named **src** under your project's directory.

Using relative paths

The **project root** is the directory on your hard disk that contains the project file—for example: **C:\MyProj\Proverbs**. You can use relative paths when referring to files within the project's directory—for example, to specify up two directory levels: **..\mydir\file.jar**.

By default, any paths you specify for files or directories are set relative to the project's root directory, provided the source directories are nested beneath the root directory. Otherwise, you must specify a hardcoded path. Locations you set in Workbench are stored in the project file.

Because location settings will be shared among subprojects and possibly other developers, you should try to avoid absolute paths. If you need to share a project file and other source files that are not under your project's directory, set environment variables in Workbench.

 For more information on team design considerations, see the *Development Guide*.

Modifying project entries

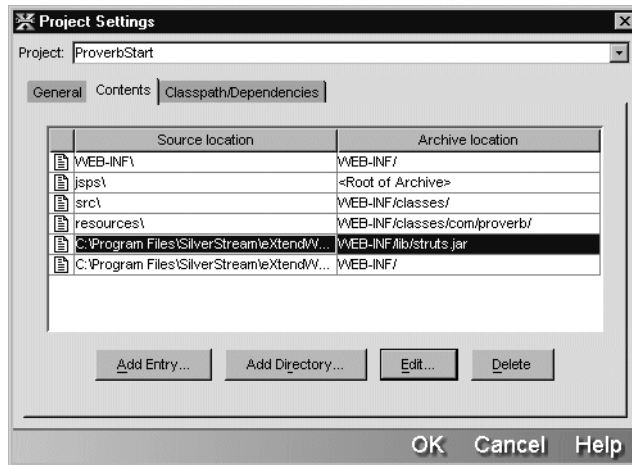
➤ To modify project entries:


1. Choose **Project>Project Settings**.
2. Open the project to modify.
You can choose between the current project and any associated subprojects.
3. On the **General** tab, view (and if necessary modify) the classes directory and the archive directory.

NOTE You cannot entirely modify the project type, directory, and file name within Workbench. See “Renaming a project” on page 37.


4. Select the **Contents** tab.

A project entry can be a file or a directory. As shown below, each project entry is defined by its source location and associated archive location.



Setting	Description or action
Project	The project to modify.
Source location	<p>The source location of the selected entry. A full path is listed whenever the source of the entry is not relative to the project root directory.</p> <p>Any files you later add to the source directory will also get included in the project archive.</p> <p>You can also use relative paths or environment variables when locating shared project files or referring to files located outside the project's directory structure.</p> <p> For more information, see “Using environment variables” on page 30 and “Using relative paths” on page 31.</p>

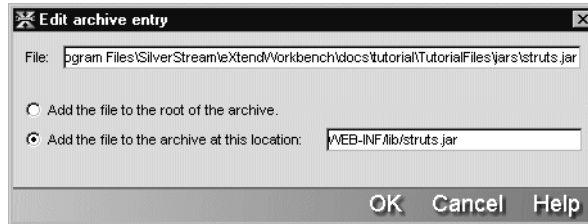
Setting	Description or action
Archive location	<p>The archive location of the contents of the selected entry. The archive location can be the same as or different from the source location.</p> <p>All archive locations are relative to the archive root directory.</p> <p>Any path you specify identifies the directory structure in the archive. For example, specifying src\com\proverb would include those files and directories in the archive with src\com\proverb as the directory structure in the archive.</p> <p>An asterisk (*) indicates that you want to include all files in the specified directory, but not any nested subdirectories.</p>
Add Entry	Lets you add a file to the project.
Add Directory	Lets you add a directory (and optionally subdirectories) to the project.
Edit	Lets you edit the selected entry name or location.
Delete	Lets you remove the selected project entry.

 For information on adding an entry or directory, see “Adding source files to a project” on page 18. For information on removing entries, see “Removing files, directories, and subprojects from projects” on page 35.

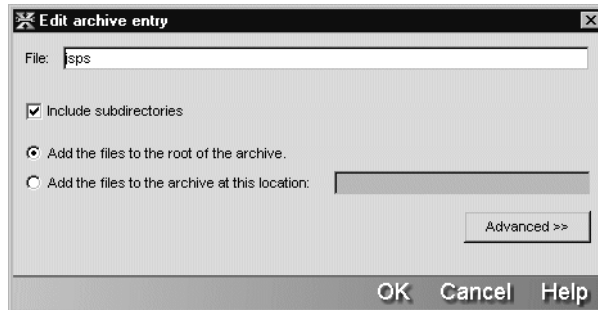
5. Select the project entry you want to modify by either double-clicking the entry or selecting the entry and clicking **Edit**.

The **Edit archive entry** dialog that appears depends on whether you are modifying a file, directory, or subproject entry.

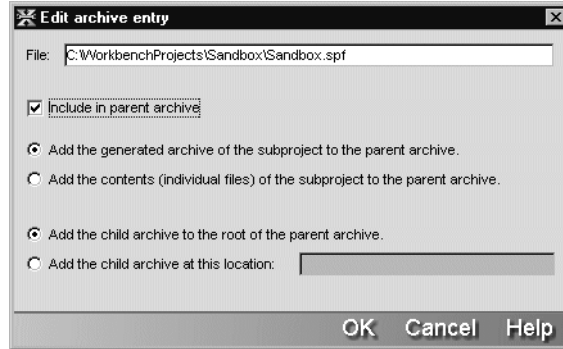
- The following dialog appears if you selected a **file**. The settings on this dialog are the same as on the Add to Project dialog. For more information, see “Adding to projects” on page 18.



- The following dialog appears if you selected a **directory**. The settings on this dialog are the same as on the Add to Project dialog. For more information, see “Adding to projects” on page 18.



- The following dialog appears if you selected a **subproject**. The settings on this dialog are the same as on the Add to Project dialog. For more information, see “Adding subprojects to a project” on page 21.



6. Click **OK** after you have modified the entry.

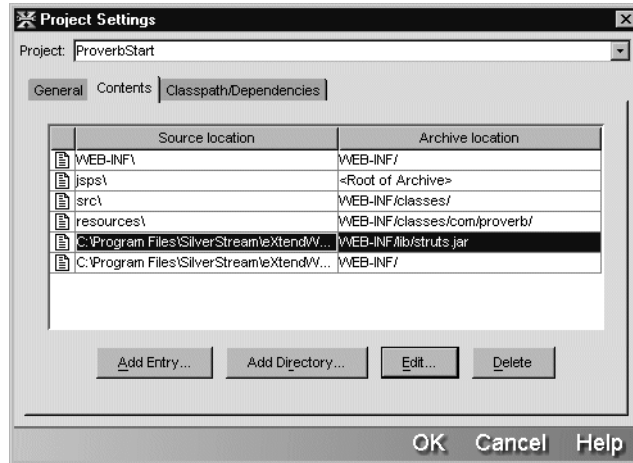
Removing files, directories, and subprojects from projects

There are two ways to remove items from a project: using the **Project Settings** dialog or using the **Remove From Project** popup menu in the Project tab. Removing a project’s source files or directories from within Workbench does not delete them from your hard disk. Workbench just removes the entry (or rule) that refers to the files or directories.

➤ To remove a file using the Project Settings dialog:

1. Choose **Project>Project Settings**.
2. Select the **Contents** tab.

3. Select the entry or entries you want to remove from your project. Press **Shift+Click** to select contiguous entries. Press **Ctrl+Click** to select noncontiguous entries.



4. Click **Delete**.
5. Click **OK** to perform the deletion. Click **Cancel** to close the dialog without performing the deletion.

If you clicked **OK**, Workbench removes the entry or entries so they are no longer referred to in the project.

Using the Remove From Project popup menu

You can also right-click the file or directory you want to remove in the Project tab of the Navigation Pane. Choosing **Remove From Project** removes the project entry from the Project Settings definition (shown above and also reflected in the SPF file) as follows:

- When you remove an explicit file (one that does not refer to any other files contained in any nested directories), Workbench simply removes the entry so that it is no longer referred to in the project.
- When you remove an individual file or a directory that was added to a project as part of nested subdirectories, Workbench prompts you to confirm that you want to remove the whole tree from the project. To exclude a file from a nested project directory, for example, you should either remove the directory from the project (and add it again later without the file) or delete the file from your hard drive.
- You can remove a subproject by selecting it in the top part of the Navigation Pane (one subproject at a time)

What Workbench lists for you is the directory trees that will be removed. For example, if you select to remove `src\abc.java`, Workbench will advise that this will cause the `src` directory tree entry to be removed. If you confirm that this is OK, Workbench removes the entire tree from the project.

Renaming a project

In rare cases, you may need to rename a project (the name preceding `.SPF`). Although you typically never directly edit a project file, you must do so in this situation.

➤ To rename a project:

1. Using your operating system tool, rename the project file.
2. (Optional) On the Contents tab of the Project Settings dialog, change the classes directory to match the revised project name. This ensures that the new project name will appear as a subdirectory of the build directory.
3. (Optional) Update the project name in the deployed object and the URL element in the deployment plan.

Steps 2 and 3 are necessary only if you want to keep all project names consistent. The project will build without them.

Compiling, building, and archiving

Workbench provides the tools you need to **compile** individual Java source files, **build** a complete project, and package the components in a J2EE-compatible **archive** for deployment to a J2EE server. This section describes the procedures for:

- Setting up your Workbench environment
- Using the commands

Setting up your Workbench environment

Setting up your Workbench environment for compiling and building includes:

- Defining the Java compiler
- Defining the project classpath

Defining the Java compiler

By default, Workbench uses the Javac 1.3 compiler. You can use the Build tab of the Preferences dialog to specify a different compiler. You can also specify options that you want sent to the compiler each time a Java file is compiled.



For more information, see “Build preferences” on page 15.

Defining the project classpath

The project classpath defines where Workbench can find the components that your source code references.

You can use environment variables when editing a project classpath.



For more information, see “Using environment variables” on page 30.

Workbench constructs the project classpath using these values:

Item	Description
Workbench defaults	<p>By default, Workbench uses:</p> <ul style="list-style-type: none">• The standard JDK default classpath (for all projects)• The SilverStream eXtend Workbench JAR file that provides the J2EE API packages that are needed for compiling J2EE projects. For J2EE 1.2 projects, the file is <code>j2ee_api_1_2.jar</code>; for J2EE 1.3 projects, the file is <code>j2ee_api_1_3.jar</code>. <p>If the J2EE API JAR file is accidentally removed from the classpath, you can find it in the compilelib directory of the Workbench install directory.</p>

Item	Description
Project contents	The Contents tab of the Project Settings dialog lists the components that you've added to a project. Workbench adds these items to the project's classpath in the order in which you added them to the project.
Classpath	<p>If your project has build dependencies on classes (for example, a WAR that contains a servlet that references an EJB), JARs (such as a Struts JAR), or related project files (like an EJB JAR and an EJB-client JAR), you can list these build dependencies using the Classpath/Dependencies tab of the Project Settings dialog.</p> <p>You can resolve the build dependency by adding either the related project's SPF file or its archive to the classpath. It is recommended that you put the project's SPF file on the classpath because:</p> <ul style="list-style-type: none"> • If you put the project file on the classpath, Workbench can determine when the related project has changed and if the related project needs to be rebuilt. This ensures that you always have the most recent archive. • If you put the archive on the classpath, Workbench cannot determine if the project has changed (which might result in the use of outdated files).

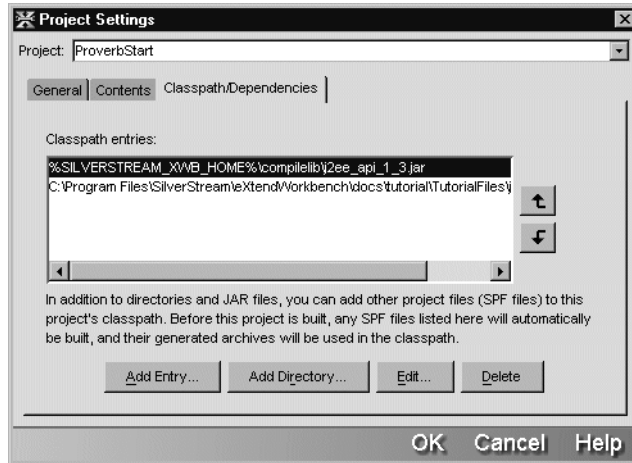
Parent project classpaths If you have a project that contains subprojects, Workbench builds the components and constructs the parent project's classpath as follows:

1. Builds any referenced projects **before** it builds the parent project.
The referenced projects are specified in the Contents tab or Classpath/Dependencies tab of the Project Settings dialog.
2. If the referenced projects build successfully, Workbench builds the parent project using the following:
 1. The parent project's contents
 2. The parent project's classpath
 3. The referenced project's classpaths (which are constructed following the same rules—the contents, the classpath, and any referenced projects)

Suppose you have an EAR project, and the EAR contains a WAR (a subproject), and the WAR contains a utility JAR. Workbench constructs the JAR's classpath first, then the WAR's classpath.

➤ **To add to a project classpath:**

1. With the project open, choose **Project>Project Settings**.
2. Select the **Classpath/Dependencies** tab and select a classpath entry.



3. Click **Add Entry** or **Add Directory**.
A selection dialog displays.
4. If adding files, click **Browse** and navigate to the appropriate directory and select one or more files (archives or project files) and click **Open**. You can press **Ctrl+Click** to add multiple noncontiguous files and **Shift+Click** to add multiple contiguous files.
Instead of browsing to files in the dialog, you can also directly type one or more files to add to the project's classpath. Enclose each entry in quotes and separate the entries with spaces. When typing, you can specify environment variables (see "Using environment variables" on page 30).
If adding a directory, type the directory (specifying environment variables if desired) or click **Browse** and select the directory.
5. Click **OK**.
6. Repeat Steps 3 and 4 for any other required items.
7. (Optional) To edit a single classpath entry:
 1. Select the entry.
 2. Click **Edit** and modify the entry in the dialog.
 3. Click **OK**.

8. When you have added, positioned, and edited all required classpath entries, click **OK** to close the Project Settings dialog.

You can now build the open project.

Using the commands

You can use the items on the Workbench **Project** menu to **compile** individual Java files, **build** an entire project, or create a project **archive**—or you can right-click a file, project, or archive, in the Navigation Pane to run the popup menu items. The menu items are:

Project menu item	What it does
Compile	<p>Compiles the currently open Java file.</p> <p>(Does not perform checking for interdependencies between the currently open file and other files in the project and its subprojects.)</p> <p>NOTE Compile is not available in the popup menu that appears when you right-click a project file in the Navigation Pane.</p>
Build	<ol style="list-style-type: none"> 1. Compiles all files in the currently open top-level project and any subprojects. Performs dependency checking on modified files to avoid unnecessary recompilations. 2. Saves the project's modified files if the Always save modified files before compiling preference setting is enabled. For more information, see "Setting preferences" on page 13. 3. Writes the generated class files to the locations specified in the Project Settings dialog.

Project menu item	What it does
Rebuild All	<ol style="list-style-type: none">1. Compiles all files in the currently open top-level project and any subprojects regardless of what has been modified.2. Saves the project's modified files if the Always save modified files before compiling preference setting is enabled. For more information, see "Setting preferences" on page 13.3. Writes the generated class files to the locations specified in the Project Settings dialog.
Build and Archive	<ol style="list-style-type: none">1. Executes the functionality described under the Build command (recompiles files subject to dependency checking).2. Creates the archives defined by the top-level project and its subprojects.
Rebuild All and Archive	<ol style="list-style-type: none">1. Executes the functionality described under the Rebuild All command (recompiles all files in the project).2. Creates the archives defined by the top-level project and its subprojects.

➤ **To compile a Java file:**

- With a Java file open, select **Projects>Compile**.
Workbench compiles the Java file and writes the compile messages to the Build tab of the Output Pane.

➤ **To build a project:**

- With a Workbench project open, select **Projects>Build**.
Workbench writes any build messages to the Build tab of the Output Pane.

➤ **To create an archive:**

- With a Workbench project open, select **Projects>Build and Archive**.
Workbench writes any build or archive messages to the Build tab of the Output Pane.

Building from the command line

Workbench provides a command-line tool (**xwbbuild**) that allows you to build projects outside of the Workbench IDE.

➤ To build a project from the command line:

1. Open a command window.
2. Make current the Workbench bin directory (it contains xwbbuild).
3. Issue the following command:

```
xwbbuild projectFile operation
```

where:

Argument	Description
projectFile	Path to the SilverStream project (.SPF) file for the project you want to build
operation	<p>One of the following:</p> <ul style="list-style-type: none"> • build—Builds and creates the archive(s) for the specified project (equivalent to selecting Project>Build and Archive) • rebuild—Rebuilds and creates the archive(s) for the specified project (equivalent to selecting Project>Rebuild All and Archive) • clean—Removes all files from the project’s build directory and deletes the archive(s) (no equivalent in the Workbench IDE)

NOTE xwbbuild displays messages while it processes the project.

For example, the following command builds and creates the archive for the myApp Workbench project (if changes had been made since the last time the project was built and archived):

```
xwbbuild c:\WorkbenchProjects\myApp\myApp.spf build
```

NOTE The Workbench IDE and xwbbuild use Apache Ant to do the build processing. For more information on using Ant, including additional command-line options you can provide with xwbbuild and how to use Ant to do your own customized processing, see “Using Ant” on page 44.

Validating archives

You should validate your archive's deployment descriptor before attempting to deploy the archive. Selecting **Project>Validate Archive** runs Sun's Verifier class.

Validation process When validating, Workbench:

1. Builds and archives the project.
2. Validates the deployment descriptor of the project archive against both the deployment descriptor DTD specified by the J2EE specification and the contents of the archive.
3. Validates the deployment descriptors of any subproject or prebuilt archives specified in the top-level deployment descriptor.

NOTE Any subproject that is **not** listed in the parent project's deployment descriptor will not be verified.

4. Writes any messages to the Validate tab of the Output Pane.

Validation output Validate Archive writes information to:

- The Validate tab of the Output Pane
- The results.txt file (located in your system TEMP directory)

In the Validate tab of the Output Pane, you can double-click the line containing the string **result.txt** to open the file in the Text Editor. The result.txt file displays:

- The archive that was tested
- The type of errors or warnings (if any) that were found

➤ To validate a project archive:

1. With the project open, select **Project>Validate Archive**.
Selecting this menu item builds the archive and (if successful) validates it.
2. After the process runs, check the **Validation** tab of the Output Pane.
3. If there are validation errors, double-click the following text in the Validate Pane:

```
Look in file "C:\TEMP\Results.txt" for detailed results on test
assertions.
```

The results.txt file opens.

When you are through noting and fixing the errors, you can try validating the archive again.

3 Archive Deployment

To make your J2EE application available to users, you deploy the archive on a J2EE server. This chapter describes how to deploy J2EE archives using Workbench and includes the following topics:


- Workbench-supported J2EE servers
- Workbench deployment types
- Using Workbench to deploy J2EE archives
- What Workbench does when you deploy a project
- Deploying Web Services
- Undeploying archives

Workbench-supported J2EE servers

Workbench provides built-in support for deploying archives to the following J2EE servers:

Server	Server archive support
SilverStream eXtend Application Server	Allows you to directly deploy application clients, EARs, EJB JARs, RARs, and WARs.
BEA WebLogic	Allows you to directly deploy application clients, EARs, EJB JARs, RARs, and WARs.
IBM WebSphere	Allows you to directly deploy EARs. Workbench allows you to develop any archive type. At deployment, Workbench repackages your archive as an EAR. Workbench supports local deployment to a Standard server only.
Oracle9iAS	Allows you to directly deploy EARs. Workbench allows you to develop any archive type. At deployment, Workbench repackages your archive as an EAR. You must wrap application clients in an EAR manually.

Server	Server archive support
SUN Reference Implementation (RI)	Allows you to directly deploy EARs. You must repackage other archive types in an EAR before deploying them.
Jakarta Tomcat	Allows you to directly deploy WARs.

 See the *Release Notes* for the latest information on the supported server versions.

Workbench deployment types


You can deploy the Workbench-produced archives using:

- Workbench rapid deployment
- Workbench production deployment
- Non-Workbench tools

Workbench rapid deployment

When developing, testing, and refining your application, you want fast turnaround—you want to make a change to your application and immediately see the result without having to redeploy the application. Workbench lets you do this using **rapid deployment**. You specify rapid deployment by simply checking a checkbox in Workbench's Deployment Settings dialog (described in “Creating deployment settings” on page 8). When you deploy the application, Workbench uses the target server's native file system deployment facilities.

Rapid deployment is most useful for changes to Web applications that involve JSP pages, HTML pages, images, JARs in the WEB-INF\lib, or classes in the WEB-INF\classes directories. If you make changes to other application components (like a WAR tag library or a deployment descriptor) Workbench automatically performs a full deployment.

 For more information about setting up a rapid deployment environment, see “Creating deployment settings” on page 8. For more information about the target server's native file system deployment, see “What Workbench does when you deploy a project” on page 14.

The following table lists the J2EE servers that allow you to use the Workbench rapid deploy feature and the kind of archives that you can rapid deploy with each:

Server	EAR	WAR	EJB	CAR	RAR
SilverStream eXtend Application Server (3.7.2)	No	No	Yes	No	No
SilverStream eXtend Application Server (3.7.x, starting with 3.7.3)	Yes	Yes	Yes	No	No
SilverStream eXtend Application Server (4.0 and higher)	Yes	Yes	No	No	No
BEA WebLogic	Yes	Yes	Yes	No	No
IBM WebSphere	Yes	Yes	Yes	No	No
Jakarta Tomcat	No	Yes	No	No	No
Oracle9iAS	Yes	Yes	Yes	No	No
Sun Reference Implementation	No	Yes	No	No	No

Workbench production deployment

When you've completely tested your application and are ready to put it into production, you can deploy the application to the server by **un**checking the rapid deploy checkbox in the deployment settings for the target server. Workbench uses the target server's native deployment tools to deploy the application in the appropriate deployment directory.

Non-Workbench tools

Alternatively, you can take your Workbench-generated archives and deploy them outside Workbench with the deployment facilities provided by your J2EE server. Because the archives Workbench generates are standard, you can deploy them to any standard J2EE server.

Using Workbench to deploy J2EE archives

To deploy a J2EE archive using Workbench, the **archive** must:

- Be properly structured according to the J2EE specification (see “Archive contents” on page 5)
- Reside in a SilverStream eXtend Workbench project (see Chapter 2, “Projects and Archives”)

Before Workbench can perform the deployment, **you** must supply:




- A server profile (see “Server profile” on page 24)
- Server-specific deployment information (see “Server deployment information” on page 7)
- Deployment settings (see “Creating deployment settings” on page 8)


Before Workbench can perform the deployment, Workbench must have:

- Access to the target server
- Permission to write to the server’s deployment area
- Permission to write temporary files when deploying to a SilverStream server: When deploying to a SilverStream server, Workbench invokes SilverCmd, which generates temporary files on disk. These files are created in the server’s installation directory, unless you have defined a HOME environment variable. If you have a HOME variable, the temporary files are created in %HOME%\silverstream. So if you have a HOME environment variable defined, it must point to a reachable and writeable location.

Archive contents





Sun's J2EE specifications define how different J2EE archives must be packaged for deployment. Before you try to deploy, make sure that your archive meets these requirements. The following table briefly lists the requirements. For more detailed information, see the *J2EE Blueprints* at: <http://java.sun.com/j2ee/docs.html>.



J2EE module	Standard archive requirements
Application client	<p>A JAR file containing:</p> <ul style="list-style-type: none"> • The Java classes that implement the application client • A deployment descriptor called application-client.xml located in the JAR's /META-INF directory • A manifest file with a Main-Class entry <p> For more information, see J2EE Deployment Descriptors DTDs in the online <i>Reference</i>.</p>
EAR	<p>An EAR file containing:</p> <ul style="list-style-type: none"> • The component archive files (such as EJB JAR files, WAR files, and application client JAR files); each of these components must include its own deployment descriptor • A deployment descriptor for the EAR called application.xml located in the EAR's /META-INF directory <p> For more information, see J2EE Deployment Descriptor DTDs in the online <i>Reference</i>.</p>
EJB JAR	<p>A JAR file containing:</p> <ul style="list-style-type: none"> • The bean implementation class, the remote and home interfaces, the primary key classes (if necessary), and any other utility classes • A deployment descriptor called ejb-jar.xml located in the JAR's /META-INF directory <p> For more information, see J2EE Deployment Descriptor DTDs in the online <i>Reference</i>.</p>

J2EE module	Standard archive requirements
RAR	A RAR file containing: <ul style="list-style-type: none">• The classes needed to implement the resource adapter• A deployment descriptor called ra.xml located in the JAR's /META-INF directory
WAR	A WAR file containing: <ul style="list-style-type: none">• JSP source files, Web Services, servlet classes, other supporting Java components, HTML documents, images, and other files required by the application• A deployment descriptor called web.xml located in the WAR's /WEB-INF directory• Helper classes in the WAR's /WEB-INF/classes directory• Helper libraries in the WAR's /WEB-INF/lib directory <p> For more information, see J2EE Deployment Descriptor DTDs in the online <i>Reference</i>.</p>

Server deployment information

Each J2EE server needs runtime information, and each has its own format for this information. The following table lists the deployment documents needed by each Workbench-supported server:

J2EE server	Archive	Server deployment information
SilverStream eXtend Application Server	Application client (CAR)	<p>Each type of archive uses an XML-based document called a deployment plan. The deployment plan can have any file name and can reside in any location outside the archive file. SilverStream defines a DTD for each archive type.</p> <p> For more information, see SilverStream Deployment Plan DTDs in the online <i>Reference</i>.</p> <p>You use Workbench's Deployment Plan Editor to create and populate the deployment plan.</p> <p> For more information, see "Deployment Plan Editor" on page 301.</p>
	EAR	
	EJB	
	RAR	
	WAR	
BEA WebLogic	Application client	<p>Each type of archive (except EAR) requires a special XML-based document. The EAR does not require a specific deployment document, but each individual module included in the EAR must have the appropriate WebLogic deployment document.</p> <p> For more information, see your WebLogic documentation.</p>
	EAR	
	EJB	
	RAR	
	WAR	
IBM WebSphere	Application client	<p>Each type of archive uses one or more deployment documents.</p> <p> For more information, see your WebSphere documentation.</p>
	EAR	
	EJB JAR	
	WAR	

J2EE server	Archive	Server deployment information
Oracle9iAS	Application client	Each type of archive requires a special XML-based document.  For more information, see your Oracle9iAS documentation.
	EAR	
	EJB JAR	
	WAR	
SUN Reference Implementation	EAR	META-INF/sun-j2ee-ri.xml  For more information, see J2EE RI Runtime Deployment Descriptor DTD in the online <i>Reference</i> .
Jakarta Tomcat	WAR	No specific file is needed.

Creating deployment settings


Before you can deploy a Workbench project, you need to define the project's deployment settings. They provide information about the server on which you plan to deploy the project.

➤ To create deployment settings:

1. Choose **Project>Deployment Settings**.


NOTE If you are deploying to a SilverStream server and the project's current deployment plan is not associated with a server profile, you will be told that you need to specify a server profile before the Deployment Settings dialog can be displayed. Do so in the Edit Server Profiles dialog, then continue specifying the deployment settings.

2. In the **Server Profiles** tab, specify the following information:

Option	What to do
Profile name	Select a server profile from the list or click Add to create a new profile.  For more information on server profiles, see “Server profile” on page 24.
Save this profile as default	Select this option to make the current server profile the default profile in new projects.
User name and Password	If you have a secure server, fill in the User name and Password text boxes with an authorized user name and password for the server.


3. Select the **Deployment Info** tab.

4. Specify the following options for servers that support rapid deployment:

Option	What to do
Enable Rapid Deployment	<p data-bbox="535 343 1268 434">Check this box when you want to deploy the archive using the rapid deployment feature. Uncheck it when you want to do a production deploy.</p> <p data-bbox="535 451 1268 512">What happens When this checkbox is checked, Workbench writes files to the rapid deployment directory specified in the server profile.</p> <p data-bbox="535 529 1268 720">NOTE If you have not set a rapid deployment directory in the server profile, you are prompted for one. This directory is not the same as the location for the server’s deployment tools. It is a location on disk where you want the server to write the deployment files. Many servers require a specific directory; see “Server profile” on page 24 for the list.</p> <p data-bbox="535 737 1268 859">Further action Workbench manages updates to the deployment area on subsequent rapid deploys. You do not have to do any manual procedure that you have to when directly using the server’s rapid deployment.</p> <p data-bbox="535 876 1268 998">When to use Use rapid deployment during the development/test/refinement stage of your application development cycle. Do not use it when you deploy your application to a production environment.</p> <p data-bbox="535 1015 1268 1119"> For more information on rapid deployment and how each server supports this feature and any special requirements, see “What Workbench does when you deploy a project” on page 14.</p>

5. Specify server-specific information:

For **SilverStream eXtend Application Servers**, specify the following:

Option	What to do
SilverStream Deployment Plan	Specify the file name and disk location of the SilverStream deployment plan
Overwrite existing deployment	<p>Check this box when you want the current deployment to overwrite any previously deployed objects of the same type and name</p> <p>If you deselect this box and objects of the same name and type already exist on the server, the deployment will fail</p>
Verbosity	<p>Specify the level of informational messages to display</p> <p>Values range from 0 (for no messages) to 5 (for the most messages)</p>
Ignore compile errors	<p>Applies only to WARs and to EARs containing WARs</p> <p>Check this box when you want the deployment to ignore any errors when compiling and to deploy only those items that build successfully</p> <p>If this box is not checked and a compile error occurs, deployment fails</p>
SilverCmd Flags	<p>(Optional) Specify command-line arguments for the deployment command</p> <p> For more information on the deployment commands that are executed, see the “What Workbench does when you deploy a project” on page 14</p> <p>If you specify multiple arguments, use spaces as the delimiters. If you want to pass VM arguments, then you must precede them with +. For example:</p> <pre data-bbox="564 1333 649 1354">+Xmx256</pre> <p>All of the values entered here are appended to the end of the deployment command that Workbench constructs</p>

For **BEA WebLogic servers**, specify:

Option	What to do
WebLogic Application Name	Specify the deployment name for your application. If this is a rapid deploy, this is the directory name under the deployment directory. Workbench defaults to the project name.
Generate Targets	Choose this button to automatically create a list of components to deploy to the target servers specified in the server profile. The list is displayed in the Components and Targets text box.
Components and Targets	Do one of the following: <ul style="list-style-type: none">• Accept the values created when the Generate Targets button is selected.• Edit the values created when the Generate Targets button is chosen.• Manually type the name of the components and their target servers. Use a colon after the component name, a comma between targets, and a semicolon between component:target pairs—for example: <code>componenta:target, target; componentx:target, target</code>

Option	What to do
Deployment options	<p>Choose one of these options:</p> <ul style="list-style-type: none"> • deploy—deploys the application. Use this option when deploying the application for the first time. If rapid deploy is checked, this option performs a rapid deploy; otherwise, it performs a production deploy. • update—updates a deployed application. Use this option for all redeployments, updates to an already deployed archive, or to enable a disabled application. • undeploy—disables the application. • list—provides a list of all deployed applications on the server specified by the current project's server profile.
debug	Check this option when you want to see the debug information produced by the WebLogic deploy tool.

For **IBM WebSphere servers**, specify:

Option	What to do
Node Name	Specify the name of the Standard server node to install to
Precompile JSP	Click the checkbox (true) if you want to precompile any JSP pages before deploying to the server



For **Oracle9iAS servers**, specify:

Option	What to do
Deployment Name	Specify the application deployment name

Target Path	Specify the path on the server to deploy to
Website Name	Specify the name of the OC4J web-site.xml file containing the name of the Web site to bind this application to

6. Select **OK** to store the deployment settings with the project file.
OR
Select **Deploy** to save the deployment settings with the project file and deploy the archive.
If you select **Deploy** and you specified a user name, you are prompted for the password before deployment can continue.

➤ To deploy a Workbench project:

1. Open the Workbench project.
Any archive that you want to deploy must be defined in a Workbench project. If you created the archive using another IDE, you must create a Workbench project for it before you can deploy it.
2. Make sure you have the server-specific deployment information in the appropriate format and location for your target server.
 For more information, see “Server deployment information” on page 7.
3. Define the deployment settings for the project.
 For more information, see “Creating deployment settings” on page 8.
4. Select **Project>Deploy Archive**.
The first time you select **Project>Deploy Archive**, you must choose from a list of server profiles. Use **New** on the Server Profile dialog to create a server profile if you don't already have one.

What Workbench does when you deploy a project

When you deploy a project, Workbench uses the deployment settings to determine the J2EE server. Then:

1. Workbench compiles the Java files and creates an archive. (JSP files are compiled during deployment or when their URLs are invoked from a browser.)

2. When the compilation is successful, Workbench calls the appropriate deploy command for the target server.

The following table lists the deploy command that is called for each server:

Server	Archive	Deploy command description
SilverStream eXtend Application Server	CAR	Standard/Production deploy: SilverCmd DeployCAR Rapid deploy: Not supported for CARs
	EAR	Standard/Production deploy: SilverCmd DeployEAR Rapid deploy: Rapid deployment is supported for EARs for SilverStream eXtend Application Server Version 3.7.3 and later. It supports the rapid deployment of WARs in the EAR. It works like this: <ul style="list-style-type: none"> • When a JSP page, an HTML page, a CLASS file, or a JAR file in a WAR within the EAR changes, Workbench invokes the server's JSP/FS deployment. • When other files in the EAR are changed (such as the EAR deployment plan, EAR deployment descriptor, EJB archive, client archive, WAR deployment plan, or WAR tag library), Workbench invokes the standard/production deployment. • Workbench manages updates to the deployment area on subsequent rapid deploys (so you do not need to do any manual procedure that you might have to when directly using the server's rapid deployment).
	EJB JAR	Standard/Production deploy: SilverCmd DeployEJB Rapid deploy: SilverCmd QuickDeployEJB (3.7.x only) <ul style="list-style-type: none"> • You must use the standard deploy process the first time you deploy. You can use the rapid deploy feature on subsequent deployments.

Server	Archive	Deploy command description
	WAR	<p>Standard/Production deploy: SilverCmd DeployWAR</p> <p>Rapid deploy for Version 3.7.3 (and later): JSP/FS</p> <p>During the JSP/FS process:</p> <ul style="list-style-type: none">• Workbench expands the WAR file in the directory <i>SilverStreamInstallDir/webapps/DBname/URL</i> where <i>SilverStreamInstallDir</i> is the directory containing the SilverStream eXtend Application Server installation, <i>DBname</i> is the name of the database containing the application deployed to the file system, and <i>URL</i> is the URL specified in the deployment plan for the application (if you have specified more than one, the first one is used).• Workbench manages updates to the deployment area on subsequent rapid deploys (so you do not need to do anything manually that you might have to when directly using the server's rapid deployment such as creating the RELOAD file).• Workbench updates the <deployToFileSystem> attribute automatically when you specify a rapid deploy.

Server	Archive	Deploy command description
BEA WebLogic	All supported archives	<p data-bbox="672 282 1186 314">Standard/Production deploy: weblogic.deploy</p> <p data-bbox="672 331 1268 552">Rapid deploy: Workbench uses WebLogic's Dynamic Deployment feature to provide rapid deployment of EARs, EJBs, and WARs. You'll need to enable WebLogic Auto-Deployment through the WebLogic Management console before Workbench will be able to perform a rapid deploy. For more information on setting Auto-Deployment, see your WebLogic documentation.</p> <ul data-bbox="672 574 1268 838" style="list-style-type: none"><li data-bbox="672 574 1208 664">• During a rapid deploy, Workbench copies the modified files to the user-specified deployment directory and touches the REDEPLOY file.<li data-bbox="672 687 1268 838">• If you want to do a standard deployment after a rapid deployment, delete the rapid deployment directory and then do the standard deployment. If you do not delete the rapid deployment directory, your changes will not be reflected.

Server	Archive	Deploy command description
IBM WebSphere	All supported archives	<p>Standard/Production deploy: seappinstall</p> <p>Rapid deploy: Workbench copies the modified files to the user-specified deployment directory.</p> <ul style="list-style-type: none"> • You must use the standard deploy process the first time you deploy. You can use the rapid deploy feature on subsequent deployments. • If the changes to an EAR include changes to the WAR deployment descriptor, TLD files, or files located in WEB-INF\lib or WEB-INF\classes in the WAR, you'll need to restart the server to see the changes. • If you remove a file from any of the archives within the EAR, you'll need to: <ol style="list-style-type: none"> 1. Stop the server 2. Remove the class from the following: 3. <i>WebSphereinstalldir\AppServer\temp\machine_name\Default_Server\applicationname</i> 4. Restart the server
Oracle9iAS	All supported archives	<p>Standard/Production deploy: admin.jar</p> <p>Rapid deploy: Workbench copies the modified files to the user-specified deployment directory.</p> <ul style="list-style-type: none"> • You must use the standard deploy process the first time you deploy. You can use the rapid deploy feature on subsequent deployments. • For rapid deployment of EARs, the deployment directory specified must be the server's \applications directory. • If you are updating an EAR and the updates include changes to the WAR deployment descriptor, TLD files, or files located in WEB-INF\lib or WEB-INF\classes in the WAR, you'll need to restart the server to see the changes.

Server	Archive	Deploy command description
SUN RI	EAR	<p>Standard/Production deploy: deploytool</p> <ul style="list-style-type: none"> You must restart the server after a standard deploy <p>Rapid deploy: copy</p> <ul style="list-style-type: none"> Explodes the archive then copies the contents to a deployment directory specified by the user If the rapid deploy directory does not exist, Workbench creates it On subsequent rapid deploys, only the changed files are copied to the deployment directory You must restart the server after a rapid deploy
Jakarta Tomcat	WAR	<p>Standard/Production deploy: copy</p> <ul style="list-style-type: none"> Copies the archive to the server's \webapps directory You must restart the server after a standard deploy <p>Rapid deploy: copy</p> <ul style="list-style-type: none"> Explodes the archive then copies the contents to a deployment directory specified by the user On subsequent rapid deploys, only the changed files are copied to the deployment directory You do not need to restart the server after a rapid deploy

- The target server's deployment command creates the appropriate deployment objects on the target server.
- Workbench displays a message stating the status (success or failure) or any warning or error messages in the Deploy tab of the Output Pane.

Deploying Web Services

When you create a Web Service in Workbench by using the Web Service Wizard or by using jBroker Web directly, a servlet is generated to handle access to that Web Service (from HTTP SOAP requests). As a result, a WAR is required to package your Web Services (one or more per WAR) for deployment to a J2EE server where they will run.

You deploy that WAR in the usual way (as described earlier in this chapter). In addition, you must make sure it has runtime access to the following archives required by jBroker Web:

- **jbroker-web.jar**, which contains the jBroker Web API classes
- **jaxrpc-api.jar** and **saaj-api.jar**, which contain the Java API classes for XML-based RPC and SOAP processing
- **xerces.jar** or another XML parser
- If the WAR uses **SOAP message handlers** (an advanced JAX-RPC feature), it will also require the following archives: `activation.jar`, `commons-logging.jar`, `dom4j.jar`, `jaxp-api.jar`, and `saaj-ri.jar`

How you set up this access depends on the type of J2EE server you use:

If you deploy to one of the following servers, you must add the required JARs to the server's classpath. (Consult your server documentation to learn about adding to the classpath.)

- BEA WebLogic
- IBM WebSphere
- Jakarta Tomcat
- Oracle9i

If you deploy to the SilverStream eXtend Application Server, there's no need to add the required JARs to the server's classpath as long as you include them in the `WEB-INF/lib` directory of your WAR. If you don't include the required JARs in the WAR, you must add them to the server's `AGCLASSPATH` environment variable or specify them with the `classpathJars` deployment plan element. (For more information about `AGCLASSPATH` and `classpathJars`, see the SilverStream eXtend Application Server Core Help.)

You can obtain the required JARs by copying them from the Workbench **compilelib** directory.

Undeploying archives

Depending on the deployment server, you can disable or delete deployed archives from the server from within Workbench.

Workbench doesn't directly perform the undeployment; it calls server facilities to do the work. So, for example, if a server supports deletion but not disabling of archives, then you can delete but not disable archives from Workbench.

Typically, disabling leaves the files on the server but makes them unavailable, and deleting physically removes the files from the server. However, since Workbench simply executes the server's undeployment facility, exactly what happens depends on the server. For example, undeploying an application that had been deployed with Rapid deployment does not necessarily delete or rename the deployment directory; the server might just delete the references to that application from its metadata. See your server documentation for information about exactly what happens when you undeploy an archive.

Here is the undeployment support for the servers supported by Workbench:

Server	Disable?	Delete?	Notes
SilverStream 3.x	No	Yes	
SilverStream 4.x	No	Yes	
BEA WebLogic	Yes	Yes	
IBM WebSphere	Yes	Yes	
Oracle9iAS	No	No	Oracle9iAS Version 1 does not have an undeploy feature. To remove an application, you must manually delete the directories and archives and remove the references from the configuration files.
Jakarta Tomcat	No	No	Tomcat does not have an undeploy feature.
Sun Reference Implementation	No	Yes	

➤ To undeploy an archive:

1. With the project open, select **Project>Undeploy Archive**.

NOTE The menu item is disabled if the deployment server does not provide an undeploy feature.

The dialog that displays depends on the type of server specified in your project's Deployment settings.

- If your deployment server supports both disabling and deleting archives, you are asked which action you want to perform
 - If your deployment server supports only disabling archives, you are asked to confirm the disabling action
 - If your deployment server supports only deleting archives, you are asked to confirm the deletion
2. Respond to the dialog.

The archive is either disabled or deleted. You can see the commands that Workbench issues in the Deploy tab of the Output Pane.

4

Component Wizards

To speed project development, use Workbench wizards when creating Java components:

- EJB Wizard
- JSP Wizard
- Servlet Wizard
- Java Class Wizard
- JavaBean Wizard
- Tag Handler Wizard

You access the wizards by choosing **File>New** from the menu.

EJB Wizard

Use the EJB Wizard to create EJB1.1 entity and session beans or EJB2.0 entity, session, and message beans. The following sections describe:

- About the EJB Wizard
- Starting the EJB Wizard
- Panel sequence
- Panel reference

About the EJB Wizard

The EJB Wizard can speed your EJB development effort by providing:

- A skeleton of the bean implementation class
- The home, remote, local, and localhome interfaces (as needed)
- A primary key class (as needed)

Once you have created the EJB using the EJB Wizard, you can modify it in the Java Editor by opening its Java source files in the Project tab of Workbench.

Starting the EJB Wizard

➤ **To start the EJB Wizard:**

1. Click **File>New**.
2. On the J2EE tab, choose **EJB** and click **OK**. (Alternatively, you can double-click **EJB**.)
3. The steps that you follow depend on the type of bean you want to generate, see “Panel sequence” on page 2 for more information.

Panel sequence

This section lists the panels you need to complete in the EJB Wizard, depending on the type of bean you want to create. You can click the link to get more information about how to complete the panel.

If you want to create	You step through these panels
A stateful or stateless session bean	<ol style="list-style-type: none">1. Specifying the EJB type2. Specifying the EJB JAR configuration3. Specifying the project, package, and directory4. Specifying the EJB source<ul style="list-style-type: none">• Specifying the source class or interface5. Specifying the EJB class and interface names6. Specifying methods7. Specifying additional classes or packages to import8. Completing the EJB

If you want to create	You step through these panels
A message-driven bean	<ol style="list-style-type: none">1. Specifying the EJB type2. Specifying the EJB JAR configuration3. Specifying the project, package, and directory4. Specifying the EJB source<ul style="list-style-type: none">• Specifying the source class or interface5. Specifying the EJB class and interface names6. Specifying methods7. Specifying additional classes or packages to import8. Completing the EJB

If you want to create	You step through these panels
A BMP entity bean	<ol style="list-style-type: none">1. Specifying the EJB type2. Specifying the EJB JAR configuration3. Specifying the project, package, and directory4. Specifying the EJB source<ul style="list-style-type: none">• Specifying the source class or interfaceor<ul style="list-style-type: none">• Specifying the source database and Selecting a database table5. Specifying the EJB class and interface names6. Specifying persistent (data) fields7. Specifying primary key fields8. Specifying fields that require get/set methods9. Specifying create() methods10. Specifying find() methods11. Specifying additional classes or packages to import12. Specifying resource references13. Completing the EJB

If you want to create	You step through these panels
A 1.x CMP entity bean	<ol style="list-style-type: none">1. Specifying the EJB type2. Specifying the EJB JAR configuration3. Specifying the project, package, and directory4. Specifying the EJB source<ul style="list-style-type: none">• Specifying the source class or interfaceor<ul style="list-style-type: none">• Specifying the source database and Selecting a database table5. Specifying the EJB class and interface names6. Specifying persistent (data) fields7. Specifying primary key fields8. Specifying fields that require get/set methods9. Specifying create() methods10. Specifying find() methods11. Specifying additional classes or packages to import12. Completing the EJB

If you want to create	You step through these panels
A 2.x CMP entity bean	<ol style="list-style-type: none">1. Specifying the EJB type2. Specifying the EJB JAR configuration3. Specifying the project, package, and directory4. Specifying the EJB source<ul style="list-style-type: none">• Specifying the source class or interfaceor<ul style="list-style-type: none">• Specifying the source database and Selecting a database table5. Specifying the EJB class and interface names6. Specifying persistent (data) fields7. Specifying primary key fields8. Specifying fields that require get/set methods9. Specifying relationships10. Specifying create() methods11. Specifying find() methods12. Specifying additional classes or packages to import13. Completing the EJB

Panel reference

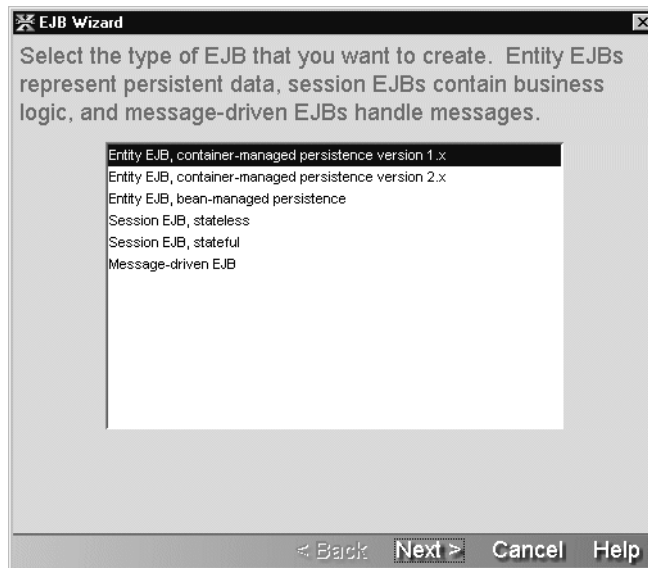
This section describes the options on each panel of the EJB Wizard. The panels are:

- Specifying the EJB type
- Specifying the EJB JAR configuration
- Specifying the project, package, and directory
- Specifying the EJB source
- Specifying the source database

- Selecting a database table
- Specifying the source class or interface
- Specifying the EJB class and interface names
- Specifying methods
- Specifying persistent (data) fields
- Specifying primary key fields
- Specifying fields that require get/set methods
- Specifying create() methods
- Specifying relationships
- Specifying find() methods
- Specifying additional classes or packages to import
- Specifying resource references
- Completing the EJB

Specifying the EJB type

This panel lets you specify the type of EJB you want to create.



➤ **To complete this panel:**

1. Specify the EJB type:

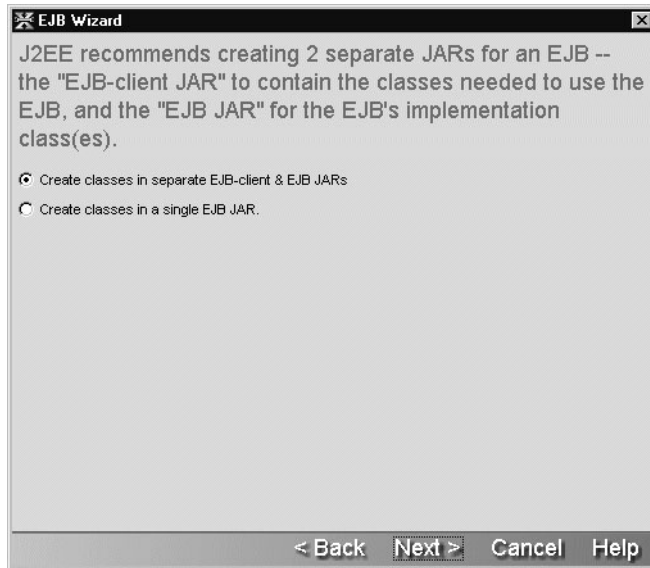
Option	What to do
Entity EJB, container-managed persistence version 1.x	Select this option when you want the EJB Wizard to create an entity bean that uses container-managed persistence (CMP) defined by the EJB1.1 specification
Entity EJB, container-managed persistence version 2.x	Select this option when you want the EJB Wizard to create an entity bean that uses container-managed persistence (CMP) defined by the EJB2.0 specification
Entity EJB, bean-managed persistence	Select this option when you want the EJB Wizard to create an entity bean that uses bean-managed persistence (BMP)
Session EJB, stateless	Select this option when you want the EJB Wizard to create a stateless session bean A stateless session bean is released to the instance pool after each method call completes, so it is not guaranteed that a client will have the same instance on subsequent method calls
Session EJB, stateful	Select this option when you want the EJB Wizard to create a stateful session bean A stateful session bean is bound to the client session that creates it, so it can be used to maintain values associated with that client session
Message-driven EJB	Select this option when you want the EJB Wizard to create a message-driven bean

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying the EJB JAR configuration

This panel lets you specify whether the wizard should create one EJB JAR or an EJB JAR and an EJB-client JAR.



➤ **To complete this panel:**

1. Specify the EJB JAR configuration:

Option	What to do
Create separate EJB-client & EJB JARs	Select this option if you want the wizard to use these two JARs: <ul style="list-style-type: none"> • EJB JAR—Contains the bean implementation classes, any utility classes that are private to the implementation, and a deployment descriptor in the META-INF directory. • EJB-client JAR—Contains the EJB home and remote interfaces, a primary key class, and any utility classes that a client might require to use the EJB. The EJB-client JAR is a plain archive file; it does not contain a deployment descriptor. If you have EJBs that are used by other EJBs in the EJB JAR (like helper EJBs) but are not used by clients, then do not put the home and remote interfaces of the helper EJBs in the EJB-client JAR.
Create a single JAR for all EJB classes	Select this option if you want the wizard to use a single EJB JAR that will contain all of the EJB classes and interfaces.

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

How EJB JARs and EJB-client JARs are related in a project An EJB-client JAR project is a peer to its EJB JAR project—it is not a subproject of the EJB JAR project. The EJB JAR and EJB-client JAR are linked in the following ways:

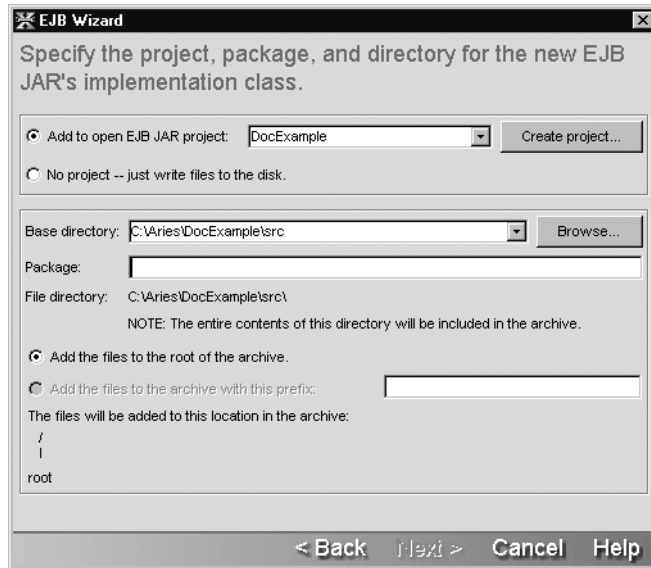
- EJB JAR project classpath includes EJB-client JAR project
- EJB JAR has a manifest file that includes a Class-Path entry for the EJB-client JAR archive
- The EJB JAR’s deployment descriptor contains an <ejb-client-jar> element containing the name of the EJB-client JAR archive

If you create an EJB JAR as a subproject, its EJB-client JAR will also be made a subproject of the same parent project. The EJB-client JAR will have the same project location, same archive location, same subproject status, and same inclusion in its parent archive as the EJB JAR.

Specifying the project, package, and directory


This panel is used to specify details about the project location (project, directory, package) where the wizard is to store the EJB files it generates.

If you chose to use **both an EJB JAR and an EJB-client JAR**, you are prompted to provide the project/package/directory information for the EJB-client JAR on a panel similar to this one.



➤ **To complete this panel:**

1. On the top portion of this panel of the EJB Wizard, specify one of the following three project association options:

Option	What to do
Add to open EJB JAR project	<p>If you currently have one or more EJB projects open in Workbench, you can add the EJB to one of those projects by selecting it from the dropdown list. If the project that you want to associate the EJB with is not currently open, you must open the target project before starting up the EJB wizard.</p> <p>If the EJB project is defined as an EJB1.1 project, then you cannot add EJBs that use EJB2.0 features—and the wizard prevents you from doing so.</p>
Create project	<p>Click Create project to start the New Project Wizard.</p> <p>When you create a new EJB project, you are prompted to specify whether it is an EJB1.1 or EJB2.0 project. You can add EJB1.1 beans to an EJB2.0 project, but not vice versa.</p> <p> For details, see “Creating projects and subprojects” on page 56.</p>
No project -- just write files to the disk	<p>If you do not want to associate the EJB with any Workbench project, you can still use the wizard to create the class in a nonproject directory on the file system.</p>

2. On the bottom portion of this panel of the EJB Wizard, specify the following:

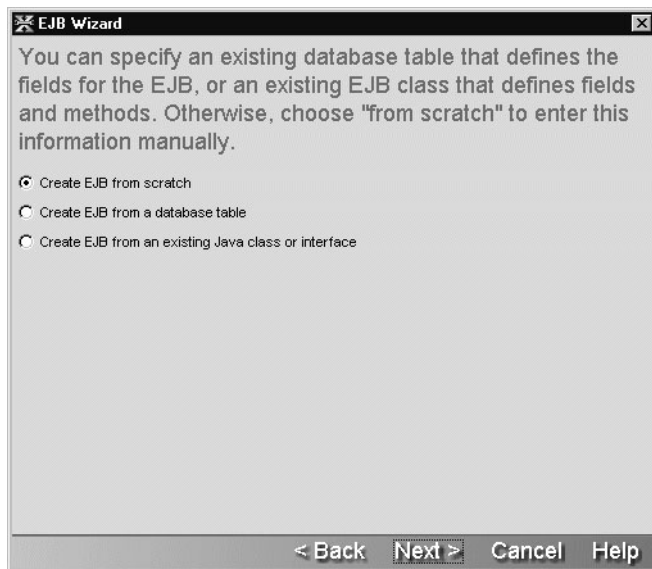
Option	What to do
Base directory	If you specified an EJB project, the default base directory is the project directory. Otherwise, this field is empty. (Click Browse to specify a file system location.)
Package	Specify the EJB's package name. This is required in the Workbench environment.
File directory	The contents of Base directory and Package are combined to specify the location of the EJB source file, which is displayed under the File directory . This is the file system location where the wizard creates the bean source file and home and remote interfaces.

3. Click **Next** to continue.

Return to "Panel sequence" on page 2.

Specifying the EJB source

This panel is used to identify whether the EJB source that the wizard generates should be completely new, based on an existing source file, or (for entity beans) a database table.



➤ **To complete this panel:**

1. Choose one of the following options:

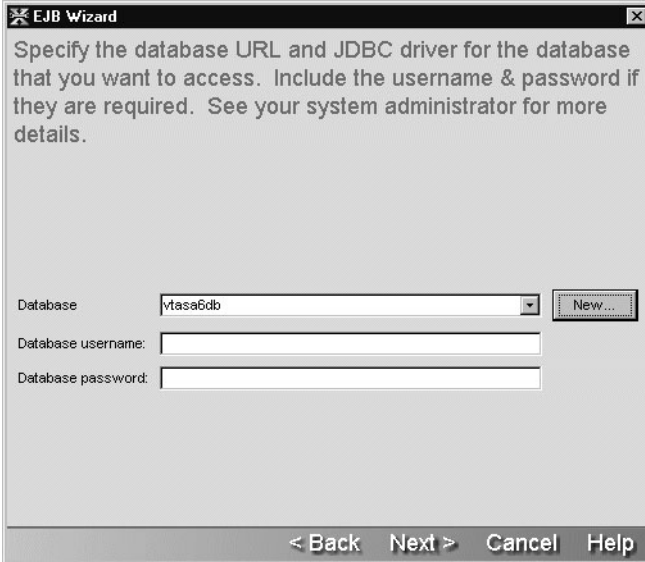
Option	What to do
Create EJB from scratch	Select this option if you want to create a new EJB
Create EJB from database	Select this option to create an entity bean whose fields are based on the fields in a specific database table
Create EJB from an existing Java class or interface	Select this option to use the properties of an existing EJB class or interface as the starting point for your EJB

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying the source database

This panel is used to specify the information that the wizard needs to connect to a database. Once connected to the database, it is able to get the list of database tables so that you can pick the database table on which the entity bean should be based.



EJB Wizard

Specify the database URL and JDBC driver for the database that you want to access. Include the username & password if they are required. See your system administrator for more details.

Database:


Database username:

Database password:

< Back Next > Cancel Help

➤ **To complete this panel:**

1. Specify:

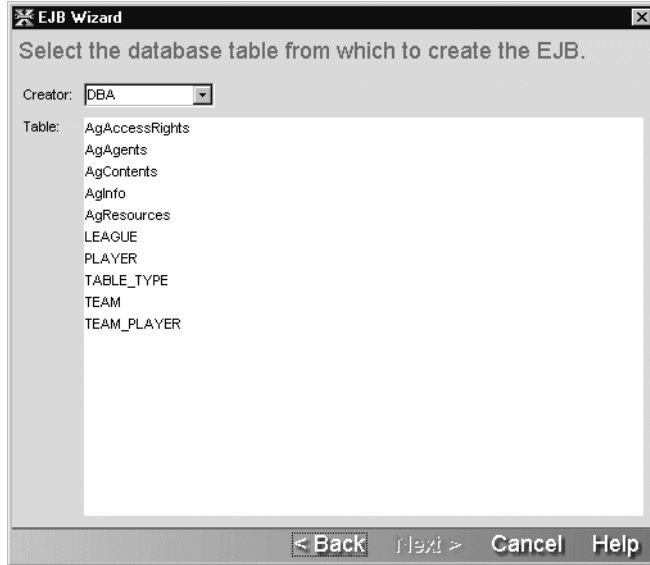
Option	What to do
Database	<p>Select a database profile from the dropdown list box. If the dropdown is not populated or if the existing profiles are unsuitable, you must create a database profile by clicking New or leaving the wizard and choosing Edit>Profiles and selecting the Databases tab.</p> <p>When you complete this panel (by clicking Next), Workbench creates a client connection to the database. This means that the database driver (specified in the database profile) must be available to Workbench.</p> <p>You can make the driver available in one of two ways:</p> <ul style="list-style-type: none"> • Putting the database driver in the Workbench lib/ext directory, where it will be picked up automatically by Workbench. <p>OR</p> <ul style="list-style-type: none"> • Putting the location of the driver on Workbench’s classpath (not the project classpath—but the classpath used when you start Workbench.) <p> For more information on database profiles, see “Database profile” on page 27.</p>
Database username and Database password	<p>Type a user name and password that you can use to connect directly to the specified database. This user name and password combination must allow access to the database’s system tables so that Workbench can access the database’s metadata.</p>

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Selecting a database table

This panel presents a list of database tables. You can select the database table that you want to use as the basis for your entity bean.



➤ **To complete this panel:**

1. Specify the following two options:

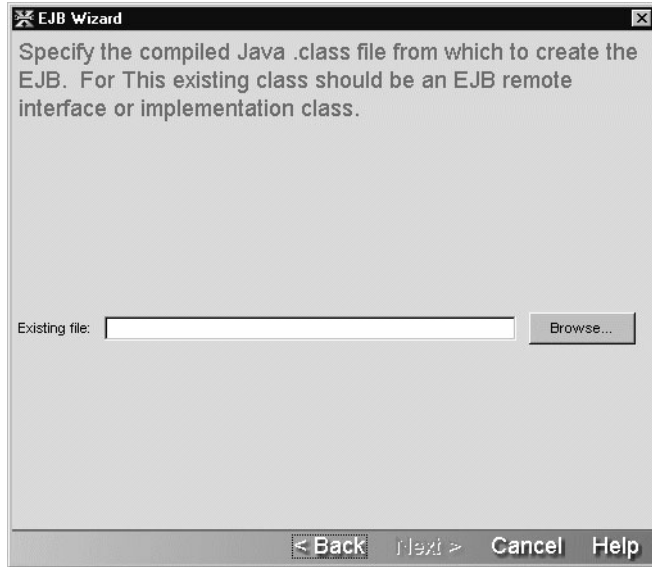
Option	What to do
Catalog/Creator/Schema	Select the Catalog/Creator/Schema containing the database table you want to use for the entity bean
Table	Select the database table that contains the fields you want to include in the entity bean

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying the source class or interface

This panel lets you choose an existing Java class or interface that the wizard should use as the basis for your EJB.



➤ **To complete this panel:**

1. Specify:

Option	What to do
Existing file	Click Browse to locate the remote interface or EJB class that you want to use as the starting point for your EJB The file you specify can only be a class file for a bean implementation or remote interface

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying the EJB class and interface names

This panel lets you specify a name for the EJB classes and interfaces that the wizard generates.

How the wizard names EJBs The EJB Wizard generates names for the EJB's implementation classes and interfaces based on a **Base** name that you supply in this wizard panel. It follows these rules when naming the EJB components:

EJB component	Naming conventions
Bean class	Prepends EB, SB, or MB and appends Bean to the base name For example, SBCalculatorBean
Remote interface	Prepends EB or SB to the base name For example, SBCalculator
Home interface	Prepends EB or SB and appends Home to the base name For example, SBCalculatorHome
Local interface (EJB2.x only)	Prepends EB or SB and appends Local to the base name For example, SBCalculatorLocal
Local home interface (EJB2.x only)	Prepends EB or SB and appends LocalHome to the base name For example, SBCalculatorLocalHome
Primary key classes (entity beans only)	Prepends EB and appends PK to the base name For example, EBCustomerPK

➤ **To complete this panel:**

1. On the top portion of this panel of the EJB Wizard, specify values for the following components:

Option	What to do
Base name	<p>Specify a legal name for the EJB class and interfaces. This name is used to construct the names for the other EJB components</p> <p>If you are creating an entity bean based on a database table, the wizard defaults these names to the database table name as the Base name and then uses the rules defined in “How the wizard names EJBs” on page 19 just above</p>
Logical EJB name	<p>Accept the default or provide a legal name</p> <p>This name is used:</p> <ul style="list-style-type: none"> • For comments in the wizard-generated code • As the <ejb-name> element in the deployment descriptor (when used within the scope of an open project)

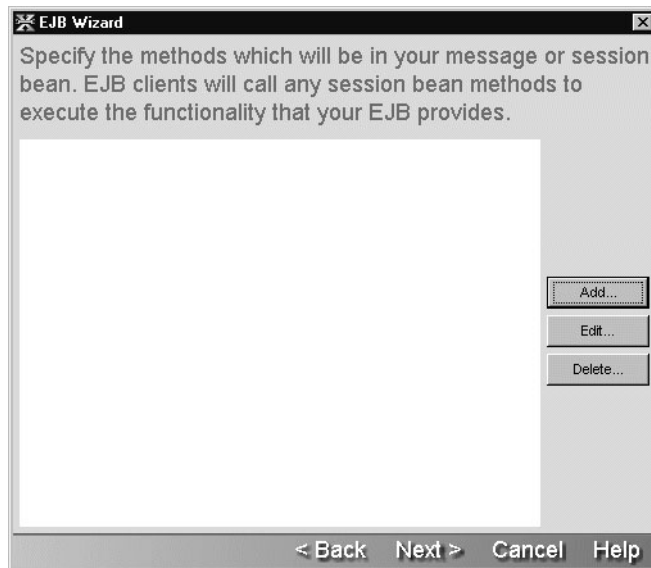
Option	What to do
Implementation class	Accept the default or specify a legal Java class name
Create primary key class	Check this when you want the EJB Wizard to create a separate primary key class
Primary key class	Accept the default or specify a legal Java class name

2. If you are creating an entity or session bean that uses CMP 2.x, you are prompted to select the radio button (on the bottom portion of this panel) that represents the set of interfaces that you want the wizard to generate.
3. Accept the default names for the set of interfaces, or specify legal Java names.
4. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying methods

This panel lets you specify the methods that the wizard will add to the bean implementation class and the remote and/or local interface. Methods on the remote and/or local interface can be called by the EJB’s client.



➤ **To complete this panel:**

1. On this panel of the EJB Wizard, click **Add** and specify the details of one method at a time:

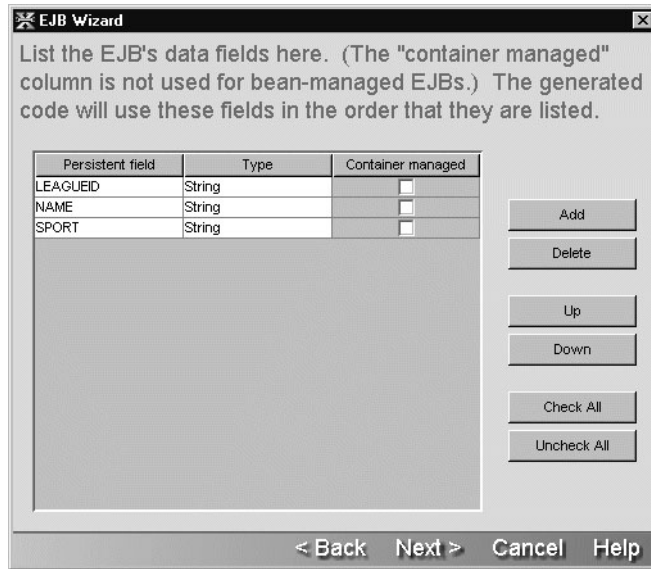
Option	What to do
Method name	Specify a legal method name
Scope	This value must be public so that the method is available to external clients; use the Java Editor to specify any nonpublic methods
Return type	Select the method's return type
Parameters	Click Add to specify the following values for the parameter: <ul style="list-style-type: none">• Type—Specify the parameter's data type• Name—Specify a legal name for the parameter
Exceptions	Click Add to specify the Exceptions that are thrown by this method You do not need to add the java.rmi.RemoteException; it is added to the remote interface by default

2. Click **OK** to create the methods.
3. Repeat these steps to create other methods or click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying persistent (data) fields

This panel lets you specify the CMP entity bean's persistent fields or the BMP entity bean's data fields. This panel is already populated if you base the bean on a database table. Otherwise, you'll have to use the Add button to add the fields you want.



➤ **To complete this panel:**

1. Specify:

Option	What to do
Persistent field	<p>If the fields that should be managed by the container are listed, make sure the check box in the container-managed column is checked</p> <p>Use the Check All/ Uncheck All and Add/Delete buttons to manage the list of container-managed data fields</p> <p>Use the Up/Down buttons to move the fields to the appropriate position if the entity bean should have a composite key</p> <p>Container-managed fields are listed in the deployment descriptor and can be mapped to a database field at deployment time</p>
Type	<p>When adding a new field, provide the Java data type. The data type must be the Java type that corresponds to the field's JDBC type. For a list of the data types, see the javadoc for <code>java.sql.Types</code>.</p>
Container managed	Check or uncheck the fields as needed

2. Click **Next** to continue.

Return to "Panel sequence" on page 2.

Specifying primary key fields

This panel lets you specify the fields that make up the entity bean's primary key.

EJB Wizard

Indicate which fields are part of the EJB's primary key. If you don't specify any primary key fields, you will have to specify the primary key class at deployment time.

Field	Primary key
LEAGUEID	<input checked="" type="checkbox"/>
NAME	<input type="checkbox"/>
SPORT	<input type="checkbox"/>

Use this single field directly as the primary key.

< Back Next > Cancel Help

➤ **To complete this panel:**

1. Specify the following information for each field that is part of a primary key field:

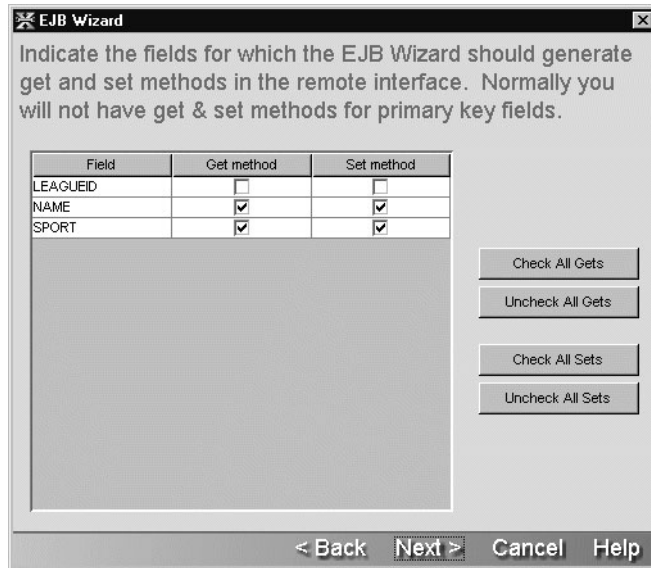
Option	What to do
Field	Move the cursor to the field
Primary Key	<p>Depends on how you responded to the Create primary key checkbox on the wizard panel described in “Specifying the EJB class and interface names” on page 18. If you:</p> <ul style="list-style-type: none"> • Checked the Create primary key class checkbox, you can check one or more fields to be included in the primary key class that the wizard will generate. <ul style="list-style-type: none"> • If you select a single field for the primary key, you must also select the Use this single field... checkbox at the bottom of the wizard panel described below. • Did not check the Create primary key class checkbox, you can do either of the following: <ul style="list-style-type: none"> • Select a single field for the primary key. You must also select the Use this single field... checkbox at the bottom of the wizard panel described below. • Unselect any primary key fields. The wizard will generate code that uses a primary key class of type <code>java.lang.Object</code>. You will have to specify the primary key class type at deployment.
Use this single field directly as the primary key	<p>Check this if you’ve selected a single field that the wizard should use as the primary key.</p> <p>The field must be a String or a wrapper class for a primitive type (such as <code>java.lang.Integer</code>). The wizard generates the code so that the wrapper-class type is in the deployment descriptor’s <code><primkey-class></code> element and the primary key field name is in the <code><primkey-field></code> element.</p>

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying fields that require get/set methods

This panel lets you specify the fields for which the wizard should generate accessor (get/set) methods.



➤ To complete this panel:

1. Specify the following information for each field that requires a get or set method:

Option	What to do
Field	Move the cursor to the field

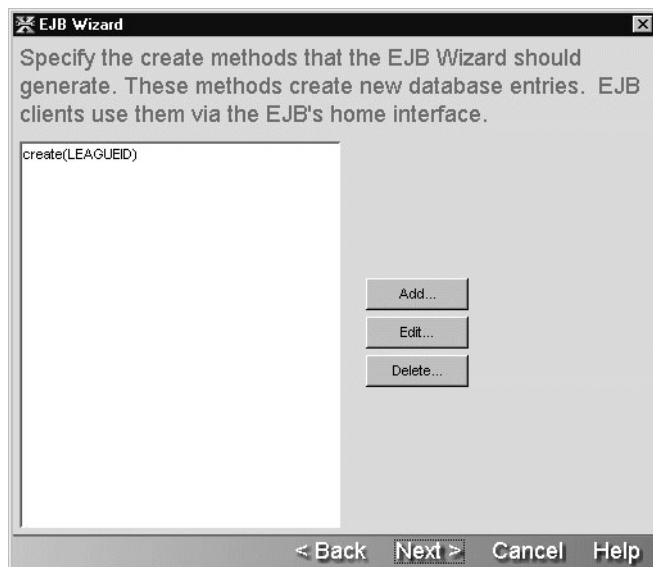
Option	What to do
Get method	Specify whether the wizard should generate (checked) a get method for this data field If your entity bean will be doing any kind of read-only or read-write data access on this data field, you should have the wizard generate a get method
Set method	Specify whether the wizard should generate (checked) a set method for this data field If your entity bean will be doing updates on this data field, you should have the wizard generate a set method

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying create() methods

This panel lets you specify the create() methods that the wizard should generate.



➤ **To complete this panel:**

1. Click **Add** to define a new create() method.

OR

Highlight an existing create() method and click **Edit**.

The Create Method Detail panel appears.

2. On the Create Method Detail panel, specify:

Option	What to do
Field	Move the cursor to the field and check or uncheck the fields that should be included in the create() method generated by the EJB Wizard
Do not delegate—generate code for this create method	For bean-managed entity beans only Click this radio button if you want the EJB Wizard to generate skeleton code for this create() method using the checked fields
Delegate to another create method	For bean-managed entity beans only Click this radio button if you do not want the EJB wizard to generate skeleton code for this create() method, then select the method to call instead from the dropdown

3. Click **OK** to return to the create() methods panel.
4. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying relationships

This panel lets you specify values for the <relationships> node in the EJB deployment descriptor. Relationships exist between two entity beans with container-managed persistence. However, when you use the EJB Wizard, you are creating a single bean at a time so you can only define the <relationship> node entries for the bean you are currently creating. You can define a relationship from the current bean to a preexisting bean or a not-yet-defined bean. For the related bean, you can use the name you know you will be giving it later, or you can use the default name **EBUnspecified**. When you use EBUnspecified, the wizard generates an incomplete relationship node in the deployment descriptor.

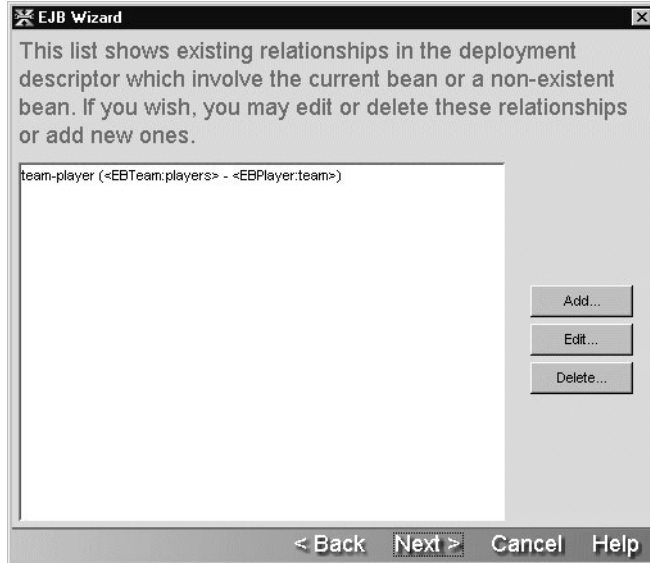
Relationships can be defined as bidirectional or unidirectional.

How to define bidirectional and unidirectional relationships In a bidirectional relationship, each bean knows about the other bean in the relationship. Each bean has methods for accessing the relationship field of the other bean. The wizard can generate these accessor methods when you define a relationship field for both sides of the relationship. The relationship field is represented in the wizard as the CMR field name.

In a unidirectional relationship, only one bean in the relationship knows about the other bean. An example of a unidirectional relationship is between lineitem and a product. The lineitem needs to know about the product, but the product does not know about the lineitem. In a unidirectional relationship, you would define a relationship field (a CMR field name) for the lineitem bean, but not for the product.

Editing bean relationships The wizard allows you to edit only the relationships listed in the deployment descriptor that are considered incomplete and that:

- Have the same bean name as the bean you are creating
OR
- Use a bean name that does not already exist



➤ **To complete this panel:**

1. To add a relationship, choose **Add**.
2. To edit or delete a relationship, highlight the relationship and choose the button appropriate to the action you want.

The Relationship Detail panel The wizard requires the following elements to generate accessor methods if this is part of a bidirectional relationship, or if it is unidirectional and is the bean that knows about the other bean:

- The CMR field name for the bean you are creating
- Whether you require a get and/or a set method
- The get/set methods return/param type

You can fill all of the other information in later using the Deployment Descriptor Editor.

Relationship Detail

Specify the details of this relationship.

Relationship name: league-unspecified

Relationship role 1:

Multiplicity: One Many

EJB name: EBLEAGUE

Cascade delete:

CMR field name:

Access methods: Get method Set method

Return/param type:

Relationship role 2:

Multiplicity: One Many

EJB name: EBUnspecified

Cascade delete:

CMR field name:

OK Cancel Help

➤ **To complete this panel:**

1. Specify:

Option	What to do
Relationship name	<p>Enter a unique name that identifies the relationship you are constructing.</p> <p>This corresponds to the <ejb-relation-name> element in the Deployment Descriptor. This element is not required by the Deployment Descriptor or the wizard.</p> <p>For each relationship, you must specify two beans.</p>
Relationship role 1	
Multiplicity	<p>Enter the cardinality of the relationship from the current bean (the one you are creating) to the related bean; it can be One or Many.</p> <p>This corresponds to the <multiplicity> element in the deployment descriptor.</p>
EJB name	<p>Enter the bean name. The bean name entered here must always match an <ejb-name> element in the enterprise-beans section of the deployment descriptor.</p> <p>The wizard adds this entry to the to the <ejb-name> element of the <relationship-role-source> element in the deployment descriptor.</p>
Cascade delete	<p>Check this if you want the current bean to be deleted when the related bean is deleted.</p> <p>This is only available when the related bean's multiplicity is One.</p> <p>This corresponds to the <cascade-delete> element of the deployment descriptor.</p>

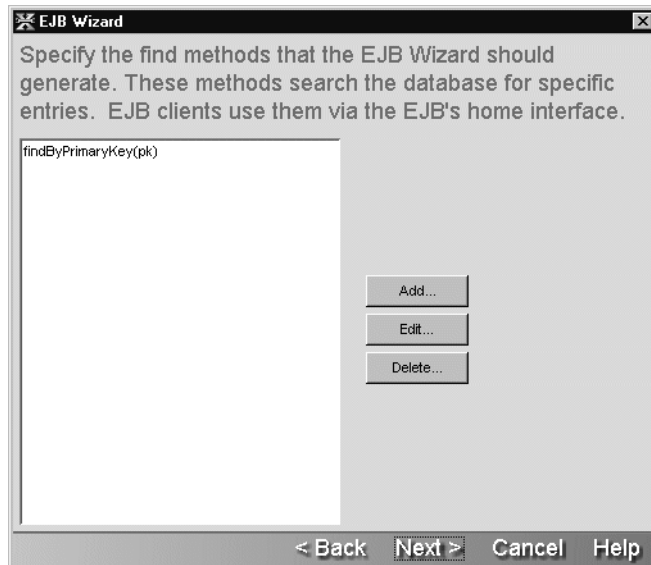
Option	What to do
CMR field name	<p>If this bean is in a bidirectional relationship or is in a unidirectional relationship and knows about the related bean, then enter a name that begins with a lowercase letter.</p> <p>The wizard uses the name to generate methods for accessing the related bean. This corresponds to the <cmr-field-name> element of the cmr-field node in the deployment descriptor.</p>
Access methods	If a cmr-field is specified, you must create set and/or get methods. Otherwise, no accessor methods are needed.
Return/param type	The return type must be the local interface of the related bean or a java.util.Collection type (depending on the related bean's multiplicity).
Relationship role 2	
Multiplicity	Enter the cardinality of the relationship of the related bean to the bean you are creating. It can be One or Many.
EJB name	<p>Enter the bean name.</p> <p>This should match an <ejb-name> element in the enterprise-beans section of the deployment descriptor (although it might not exist yet).</p>
Cascade delete	Check this check box if you want this bean removed when the current bean is removed.
CMR field name	<p>If this bean is in a bidirectional relationship or is in a unidirectional relationship and knows about the current bean, then enter a name that begins with a lowercase letter.</p> <p>The wizard uses the name to generate methods for accessing the related bean. This corresponds to the <cmr-field-name> element of the cmr-field node in the deployment descriptor.</p>

2. Click **OK** to return the relationship panel.
3. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying find() methods

This panel lets you define the bean’s finder methods.



➤ To complete this panel:

1. On this panel of the EJB Wizard, click **Add** to define a new find() method.
OR
Highlight an existing find() method and click **Edit**.
The Find Method Detail panel appears.

2. On the Find Method Detail panel, specify:

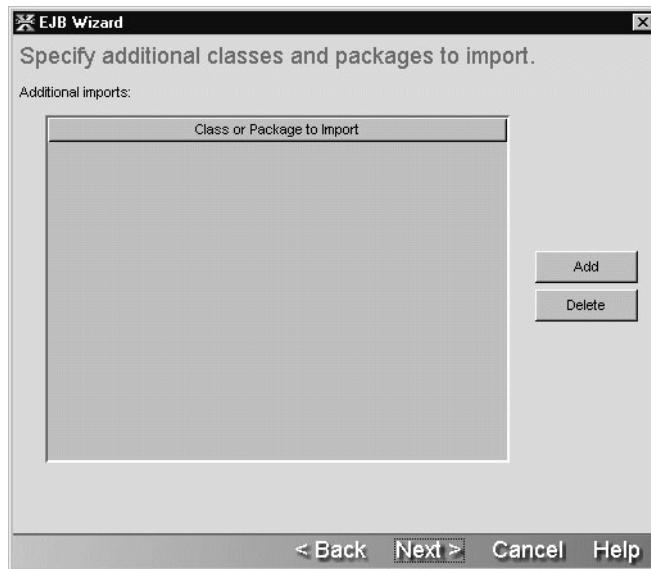
Option	What to do
Method name	Specify a legal Java method name
Returns	Click the radio button associated with the Java type that is returned
Method parameters type	Click Add to enable the parameter type and name text box, then specify the Java data type of the parameter
Method parameter name	Specify a legal Java parameter name

3. Click **OK** to return to the find() methods panel.
4. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying additional classes or packages to import

This panel is used to specify any other classes or packages you want the wizard to generate import statements for. The wizard does not import any packages by default.



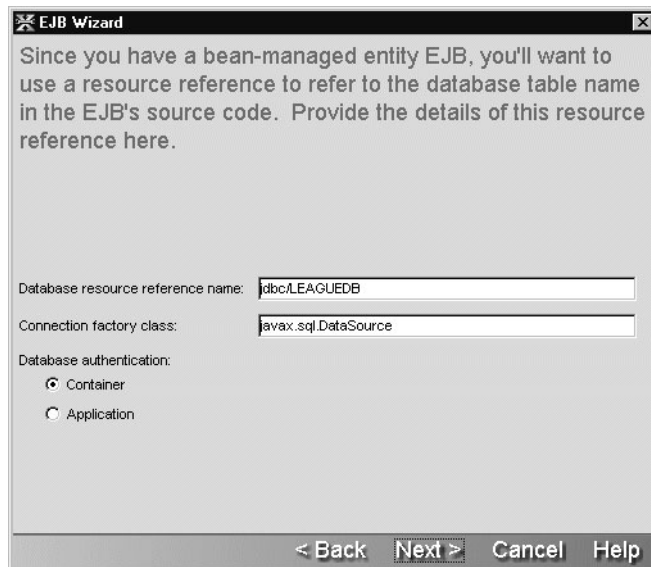
➤ **To complete this panel:**

1. Click **Add** to specify classes or packages to import.
2. Type the fully qualified path name of the Java class or the full package name in the text field.
3. When you are done adding or removing additional classes or packages, click **Next** to continue.

Return to “Panel sequence” on page 2.

Specifying resource references

This panel lets you specify the resource reference that you’ll use as a substitute for the database table name in the bean’s source code, the connection factory class, and the type of authentication.



The screenshot shows a dialog box titled "EJB Wizard" with a close button in the top right corner. The main text reads: "Since you have a bean-managed entity EJB, you'll want to use a resource reference to refer to the database table name in the EJB's source code. Provide the details of this resource reference here." Below this text are three input fields: "Database resource reference name:" with the value "jdbc:LEAGUEDB", "Connection factory class:" with the value "javax.sql.DataSource", and "Database authentication:" with two radio buttons, "Container" (selected) and "Application". At the bottom of the dialog are four buttons: "< Back", "Next >" (highlighted with a dotted border), "Cancel", and "Help".

➤ **To complete this panel:**

1. Specify:

Option	What to do
Database resource reference name	Specify the name that will be put in the JNDI environment and looked up by anyone trying to get the named resource The EJB specification recommends that this be prefaced with jdbc/ —for example: <code>jdbc/MyDataSource</code> The deployer will later map this reference to the appropriate database
Connection factory class	Specify the Java type of the factory (not of the resource)
Database authentication	Specify who performs the login to the resource: <ul style="list-style-type: none">• Specifying container means the container signs on to the resource manager in order to obtain the resource factory• Specifying application means the code in the EJB signs on to the resource manager programmatically

2. Click **Next** to continue.

Return to “Panel sequence” on page 2.

Completing the EJB

This panel shows you all of the classes and interfaces that the wizard will generate. Review it carefully to make sure that you have specified everything that you wanted to. If you find a mistake, you can click the Back button to return to a panel and make the appropriate change.



➤ To complete the EJB:

1. Check the values in the Summary panel to make sure you have specified everything correctly and then click **Finish**.
2. When the Summary panel reports the wizard is done creating the EJB, click **OK**.
Now the EJB implementation class and the interfaces are open for editing in the Java Editor.

Return to “Panel sequence” on page 2.

JSP Wizard

Use the JSP Wizard to create JSP pages. The following sections describe:

- About the JSP Wizard
- Starting the JSP Wizard
- Specifying the JSP page name and other options
- Specifying the project, directory, and package
- Specifying imports
- What happens

About the JSP Wizard

Use the JSP Wizard to quickly specify a variety of attributes for your JSP page and add your JSP page to an open project.

Starting the JSP Wizard

➤ **To start the JSP Wizard:**


1. Choose **File>New**.
2. On the J2EE tab, choose **JSP** and click **OK**. (Alternatively, you can double-click **JSP**.)
The first panel of the JSP Wizard displays.
3. Continue as described in “Specifying the JSP page name and other options” on page 40.

Specifying the JSP page name and other options

➤ **To specify the JSP page name and other options:**

1. On the first panel of the JSP Wizard, specify the following options:

Option	What to do
JSP name	Specify the name for the JSP page. You don't need to specify the .JSP extension.

Option	What to do
Page title	<p>Specify the text for the JSP page's title.</p> <p>Generated as <code><title>text</title></code>.</p>
Content type	<p>Specify the MIME-type of the response generated by the JSP page. Choose from the list provided.</p> <p>The default is HTML.</p> <p>Generated as the contentType attribute of the page directive.</p>
Template	<p>Specify a code-generation template, if any.</p> <p>Your implementation of Workbench may or may not contain additional extension templates of classes that are commonly used as starting points for applications in your environment.</p> <p> For more information on extensibility, see “Extending the Workbench toolset and services” on page 50.</p>
Use session	<p>Specify whether the JSP page participates in session management (in other words, is part of a session).</p> <p>Generated as the session attribute of the page directive.</p>
Thread safe	<p>Specify whether the JSP page, once compiled into a servlet, can respond to multiple simultaneous requests. If not, deselect the check box.</p> <p>Generated as the isThreadSafe attribute of the page directive.</p>
Form-based page	<p>Specify whether a simple HTML form is generated on the JSP page (enabled only if you are generating an HTML or XHTML page).</p>
Create error page	<p>Specify whether an error page is generated for this JSP page (enabled only if you are generating an HTML or XHTML page). The error page is displayed if an uncaught error occurs when the server is processing the JSP page.</p> <p>Generated as the errorPage attribute of the page directive.</p>


Option	What to do
Specify import values	Specify whether you want to specify Java classes and packages to import so that you can reference classes in the JSP page without having to explicitly specify package names. If you select this option, you will see an additional panel in the wizard where you can specify the classes and packages.

2. Click **Next** to proceed to the next wizard panel. See “Specifying the project, directory, and package” on page 42 for details.

Specifying the project, directory, and package

➤ To specify the project, directory, and package:

1. On this panel of the JSP Wizard, specify the following options:

Option	What to do
Add to open WAR project	If you currently have one or more Web archive (WAR) projects open in Workbench, you can add the JSP page to one of those projects by selecting it from the list.
Create project	If you do not have a WAR project open in Workbench but want to associate the JSP page with a WAR project, click Create project to start the New Project Wizard.  See “Creating projects and subprojects” on page 56 for details.
No project—just write files to the disk.	Choose this option if you do not want to associate the JSP page with a project; the wizard will create the JSP page in a nonproject directory on the file system.

Option	What to do
Base directory	<p>If you specified a WAR project, the default base directory is the src subdirectory of the project directory. Otherwise, this field is empty.</p> <p>Click Browse to specify a file system location.</p> <p>You can add one or more subdirectories to the default base directory.</p>
Package	<p>Specify a package hierarchy (with levels separated by periods) to place the JSP page in a subdirectory of the base directory.</p> <p>This affects only the directory where the JSP page is saved and the default URL for accessing the JSP page. The JSP page itself is unaffected.</p> <p>For example, if the base directory is <i>ProjectDir/jsps</i> and you specified com.myco as the package, the JSP page will be created in <i>ProjectDir/jsps/com/myco</i>.</p>
File directory	<p>The contents of Base directory and Package are combined to specify the location of the JSP page, which is displayed under File directory.</p> <p>This is the file system location where the wizard creates the JSP page.</p> <p>You cannot change the contents of this field directly; you must change Base directory and/or Package.</p>
Add the files to the root of the archive	<p>When generating the project archive, place the JSP page at the root of the archive (taking into account any package structure you specified).</p>
Add the files to the archive with this prefix	<p>When generating the project archive, place the JSP pages in a directory tree as specified in the prefix (taking into account any package structure you specified).</p>

Option	What to do
The files will be added to this location in the archive	<p>The location in the archive of the JSP page, as specified by the two preceding selections, are reflected in this field.</p> <p>You cannot change the contents of this field directly; you must change the preceding two selections.</p>

2. If on the first panel you specified that you want to specify import values, click **Next** to proceed to the next panel. See “Specifying imports” on page 44.

Otherwise, you are done. Click **Finish**. When the final wizard panel reports that it has finished creating the JSP page, click **OK**. See “What happens” on page 44.

Specifying imports

➤ To specify classes and packages to import:

1. On this panel of the JSP Wizard, specify which classes you want to reference in the JSP page without having to specify their package names.

Classes or packages you specify here are generated as the **import** attribute of the page directive. This directive corresponds to import statements in a Java source file.

NOTE As a convenience, every JSP page automatically imports all the classes from these packages: `java.lang`, `javax.servlet`, `javax.servlet.http`, and `javax.servlet.jsp`.

To add a class or package, click **Add** and specify the class or package. You can add as many classes or packages as you want.

2. Click **Finish**. When the final wizard panel reports it has finished creating the JSP page, click **OK**. See “What happens” on page 44.

What happens

The JSP page is generated and appears in the JSP Editor. If you specified that you wanted an error page associated with the JSP page, the error page is generated in the same directory with the name `JSPPageNameErrorPage.jsp` and is specified in the JSP page’s **errorPage** attribute of the page directive.

The wizard adds the JSP page (and error page if present) to the open project if you selected that option.

Servlet Wizard

Use the Servlet Wizard to create servlet Java class files. Topics discussed in this section include:


- About the Servlet Wizard
- Starting the Servlet Wizard
- Specifying the class name and other servlet options
- Specifying the project, directory, and package
- Specifying which HttpServlet methods to override
- Specifying which interfaces to implement
- Specifying which classes and packages to import

About the Servlet Wizard

The Servlet Wizard provides an automated mechanism for creating Java servlet source files. The wizard provides options to specify these attributes of a Java servlet class:

- The content type of the HTTP response document, such as HTML, XML, and so on
- Whether to implement the single- or multi-threaded model interface
- Whether to include the servlet in an existing Workbench project, in a new project, or not in any project
- Whether the servlet is to be a member of a package (that is, whether to specify a package definition)
- The directory structure for the Java source files and for the generated classes in the archive
- Whether to override specified HttpServlet methods
- Whether to implement any interfaces
- Whether to import any classes or packages

Once you have created a servlet using the Servlet Wizard, you can modify that servlet in the Java Editor.

 For details about editing servlets in Workbench, see Chapter 6, “Source Editors” in this book and the chapter on writing servlets in the *Development Guide*.

Starting the Servlet Wizard


➤ **To start the Servlet Wizard:**

1. Choose **File>New**.
2. On the J2EE tab, choose **Servlet** and click **OK**. (Alternatively, you can double-click **Servlet**.)
3. To continue, see “Specifying the class name and other servlet options” on page 46.

Specifying the class name and other servlet options

➤ **To specify the class name and other servlet options:**

1. On the first panel of the Servlet Wizard, specify the following options:

Option	What to do
Class name	Specify an appropriate name for the servlet class.
Content type	Specify the type of the document content of the HTTP response the servlet is to generate. The default is HTML.
Template	Specify a code-generation template, if any. Your implementation of Workbench may or may not contain additional extension templates of classes that are commonly used as starting points for applications in your environment.  For more information on extensibility, see “Extending the Workbench toolset and services” on page 50.


Option	What to do
Implement SingleThreadModel	<p>Specify whether the servlet class is to implement the SingleThreadModel interface.</p> <p>Implementing this interface guarantees that no more than one request thread accesses a single instance of your servlet. While this can guarantee that servlet fields are accessed by only one thread at a time, there can be significant performance costs if your servlet is accessed frequently.</p> <p>The default is to allow multithreaded access to the servlet.</p>

2. Click **Next** to go to the next wizard panel. See “Specifying the project, directory, and package” on page 47.

Specifying the project, directory, and package

➤ To specify the project, directory, and package:

1. On the second panel of the Servlet Wizard, specify the following options:

Option	What to do
Add to open WAR project	If you currently have one or more Web Archive (WAR) projects open in Workbench, you can add the servlet to one of those projects by selecting it from the list.
Create project	<p>If you do not have a WAR project open in Workbench but want to associate the servlet with a WAR project, you can click Create project to start the New Project Wizard.</p> <p> For details, see “Creating projects and subprojects” on page 56.</p>

Option	What to do
No project—just write files to the disk.	If you do not want to associate the servlet with any Workbench project, you can still use the wizard to create a servlet class anywhere on the file system.
Base directory	If you specified a WAR project, the default base directory is a src subdirectory located directly under the project directory. Otherwise, this field is empty. Click Browse to specify a file system location. You can add one or more subdirectories to the default base directory.
Package	If your servlet is to be a member of a package (for example, com.mwbi.welcome), specify the package name in this field.
File directory	The contents of Base directory and Package are combined to specify the location of the servlet source file, which is displayed under File directory . This is the file system location where the wizard creates the servlet source file. You cannot change the contents of this field directly; you must change Base directory and/or Package .
Add the files to the root of the archive	When generating the project archive, place the generated class files for the servlet at the root of the archive.

Option	What to do
Add the files to the archive with this prefix	<p>When generating the project archive, place the generated class files for the servlet in a directory tree as specified in the prefix.</p> <p>The default is to place the servlet classes in a WEB-INF/classes directory under the root of the archive.</p> <p>If you specified a package name, the directory structure associated with that package is added to the prefix to determine the final archive path for the generated classes.</p>
The files will be added to this location in the archive	<p>The location in the archive of the generated servlet class files, as specified by the two preceding selections, are reflected in this field.</p> <p>You cannot change the contents of this field directly; you must change the preceding two selections.</p>

2. Click **Next** to go to the next wizard panel. See “Specifying which HttpServlet methods to override” on page 49.

Specifying which HttpServlet methods to override

➤ To specify which HttpServlet methods you want to override:

1. On this panel of the Servlet Wizard, specify which methods in the HttpServlet class to override in the servlet.

Typically, you want to override the doGet and doPost methods. This panel enables you to override these HttpServlet methods:

- doGet
- doPost
- doPut
- doDelete
- init
- destroy

- `getServletInfo`

Choosing any of these methods causes the wizard to insert the basic structure for that method into the servlet code it generates so that you can easily add the appropriate processing logic later using the Java Editor.

2. Click **Next** to go to the next wizard panel. See “Specifying which interfaces to implement” on page 50.

Specifying which interfaces to implement

➤ To specify which interfaces to implement:

1. On this panel, specify any interfaces that the servlet will implement. Click **Add** to specify an interface. You must specify the fully qualified name of the interface. For each interface, you can specify whether you want the wizard to generate stub methods.
2. You can rearrange the list of interfaces by clicking **Up** or **Down**. You can specify that you want stub methods for all or none of the interfaces by clicking **Check All** or **Uncheck All**. The wizard will generate the following for each interface you specify:
 - An entry in the servlet’s **implements** statement
 - All necessary imports
 - Stub code for all interfaces where you checked **Generate Stub Code**
3. Click **Next** to go to the next wizard panel. See “Specifying which classes and packages to import” on page 50.

Specifying which classes and packages to import

➤ To specify which classes and packages to import:

1. On this panel, specify any additional classes or packages that the servlet should import. The wizard will generate an **import** statement for each entry you make here.
2. Once you have specified the imports, click **Finish**. The Servlet Wizard creates a Java servlet class based on what you specified.
3. When the wizard reports that it is done creating the servlet, click **OK**. The servlet code appears in the Java Editor. If you specified that the servlet is to be associated with a WAR project, the wizard adds the servlet to that project.

Java Class Wizard

Use the Java Class Wizard to create **general-purpose** Java class files. The following sections describe:

- About the Java Class Wizard
- Starting the Java Class Wizard
- Specifying the class name and other options
- Specifying which interfaces to implement
- Specifying which classes and packages to import
- Specifying the project, directory, and package

About the Java Class Wizard

With the Java Class Wizard you specify a variety of class attributes such as scope and whether to create a class or an interface. The wizard lets you add the new source file to an existing project, create a new project to add the new source file to, or simply write the new class file to disk.

Starting the Java Class Wizard


➤ **To start the Java Class Wizard:**

1. Choose **File>New**.
2. On the J2EE tab, choose **Java file** and click **OK**. (Alternatively, you can double-click **Java file**.)
3. Continue as described in “Specifying the class name and other options” on page 52.

Specifying the class name and other options

➤ **To specify the class name and other options:**

1. On the first panel of the Java Class Wizard, specify the following options:

Option	What to do
Class name	Specify an appropriate name for the Java class.
Base class	Specify the base class, if any. You can enter a simple or a fully qualified name.
Create class or interface?	Specify whether to create a class or an interface.
Template	Specify a code-generation template, if any. Your implementation of Workbench may or may not contain additional extension templates of classes that are commonly used as starting points for applications in your environment.  For more information on extensibility, see “Extending the Workbench toolset and services” on page 50.
Bottom group (check boxes)	Use any of the following check boxes to further specify class attributes: <ul style="list-style-type: none">• Public scope• Create a default constructor• Create main() method• Serializable

2. Click **Next** to go to the next wizard panel. See “Specifying which interfaces to implement” on page 53.

Specifying which interfaces to implement

➤ To specify which interfaces to implement:

1. On this panel, specify any interfaces that the class will implement. Click **Add** to specify an interface. You must specify the fully qualified name of the interface. For each interface, you can specify whether you want the wizard to generate stub methods.
2. You can rearrange the list of interfaces by clicking **Up** or **Down**. You can specify that you want stub methods for all or none of the interfaces by clicking **Check All** or **Uncheck All**. The wizard will generate the following for each interface you specify:
 - An entry in the class's **implements** statement
 - All necessary imports
 - Stub code for all interfaces where you checked **Generate Stub Code**
3. Click **Next** to go to the next wizard panel. See “Specifying which classes and packages to import” on page 53.

Specifying which classes and packages to import

➤ To specify which classes and packages to import:


1. On this panel, specify any additional classes or packages that the class should import. The wizard will generate an **import** statement for each entry you make here.
2. Click **Next** to go to the next wizard panel. See “Specifying the project, directory, and package” on page 53.

Specifying the project, directory, and package

➤ To specify the project, directory, and package:

1. On the top portion of this panel of the Java Class Wizard, specify one of the following three project association options:

Option	What to do
Add to open project	If you currently have one or more projects open in Workbench, you can add the class file to one of those projects by selecting it from the list.

Option	What to do
Create project	<p>If you do not have a project open in Workbench but want to associate the class file with a project, click Create project to start the New Project Wizard.</p> <p>When you are through, the new project is selected as the project to add the new class file to.</p> <p> For more information, see “Project design considerations” on page 53.</p>
No project—just write the files to the disk	Choose this option if you do not want to associate the class file with a project; the wizard will create the class file in a nonproject directory on the file system.

2. On the lower portion of this Java Class Wizard panel, specify the following options:

Option	What to do
Base directory	<p>If you specified a WAR project, the default base directory is a src subdirectory located directly under the project directory. Otherwise, this field is empty.</p> <p>Click Browse to specify a file system location.</p> <p>The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.</p>
Package	<p>Specify the fully-qualified Java package name for the new class. You can specify a package hierarchy with levels separated by periods.</p> <p>The Java class you are creating is saved in the Base directory combined with the Package directory.</p> <p>For example, if the base directory is <i>ProjectDir/classes</i> and you specified com.myco as the package, the class will be created in <i>ProjectDir/classes/com/myco</i>.</p>

Option	What to do
File directory	<p>The contents of Base directory and Package are combined to specify the location of the Java class source file, which is displayed under File directory.</p> <p>This is the file system location where the wizard creates the Java class source file.</p> <p>You cannot change the contents of this field directly; you must change Base directory and/or Package.</p>
Add the files to the root of the archive	<p>Adds the compiled Java class file to the archive root combined with the package path when generating the project archive.</p>
Add the files to the archive with this prefix	<p>Adds the compiled Java class file to the specified archive directory combined with the package path when generating the project archive.</p> <p>If you specified a package name, the directory structure associated with that package is added to the prefix to determine the final archive path for the generated class.</p>
The files will be added to this location in the archive	<p>The location in the archive of the generated Java class file, as specified by the two preceding selections, are reflected in this field.</p> <p>You cannot change the contents of this field directly; you must change the preceding two selections.</p>

3. Click **Finish**.
4. When the final wizard panel reports it's done creating the Java class, click **OK**.
The code appears in the Java Editor. (The Java class is added to the open project only if you selected that option.)

The wizard creates the Java class source file. After you write the methods that implement the specific functionality for this new class (as well as any import statements), you can add the new class file to a project.



For more information, see “Adding to projects” on page 68.

JavaBean Wizard

Use the JavaBean Wizard to create JavaBeans. The following sections describe:

- About the JavaBean Wizard
- Starting the JavaBean Wizard
- Specifying the class name and other options
- Specifying the data fields
- Specifying which interfaces to implement
- Specifying which classes and packages to import
- Specifying the project, directory, and package

About the JavaBean Wizard

Use the JavaBean Wizard to quickly create a skeleton for a JavaBean and add it to an open project.

Starting the JavaBean Wizard

➤ **To start the JavaBean Wizard:**

1. Choose **File>New**.
2. On the J2EE tab, choose **JavaBean** and click **OK**. (Alternatively, you can double-click **JavaBean**.)

The first panel of the JavaBean Wizard displays.


3. Continue as described in “Specifying the class name and other options” on page 56.

Specifying the class name and other options

➤ **To specify the class name and other options:**

1. On the first panel of the JavaBean Wizard, specify the following options:

Option	What to do
Class name	Specify the name for the JavaBean. You don't need to specify the .Java extension.

Option	What to do
Base class	<p>If the JavaBean is inherited from a base class, specify the name of the base class. You can specify a simple or fully qualified name.</p> <p>Generated as extends <i>class</i>.</p>
Template	<p>Specify a code-generation template, if any.</p> <p>Your implementation of Workbench may or may not contain additional extension templates of classes that are commonly used as starting points for applications in your environment.</p> <p> For more information on extensibility, see “Extending the Workbench toolset and services” on page 50.</p>

2. Click **Next** to proceed to the next wizard panel. See “Specifying the data fields” on page 57 for details.

Specifying the data fields

➤ To specify the data fields for the JavaBean:

1. Define each data field by clicking **Add** and specifying the name and data type.
The generated Java file will define the fields in the order in which they are listed here. You can reorder the list by selecting a field and clicking **Up** or **Down**.
2. Click **Next** to proceed to the next wizard panel. See “Specifying which interfaces to implement” on page 57 for details.

Specifying which interfaces to implement

➤ To specify which interfaces to implement:

1. On this panel, specify any interfaces that the bean will implement. Click **Add** to specify an interface. You must specify the fully qualified name of the interface. For each interface, you can specify whether you want the wizard to generate stub methods.
2. You can rearrange the list of interfaces by clicking **Up** or **Down**. You can specify that you want stub methods for all or none of the interfaces by clicking **Check All** or **Uncheck All**.

The wizard will generate the following for each interface you specify:

- An entry in the bean's **implements** statement
 - All necessary imports
 - Stub code for all interfaces where you checked **Generate Stub Code**
3. Click **Next** to go to the next wizard panel. See “Specifying which classes and packages to import” on page 58.

Specifying which classes and packages to import


➤ **To specify which classes and packages to import:**

1. On this panel, specify any additional classes or packages that the bean should import. The wizard will generate an **import** statement for each entry you make here.
2. Click **Next** to go to the next wizard panel. See “Specifying the project, directory, and package” on page 58.

Specifying the project, directory, and package

➤ **To specify the project, directory, and package:**

1. On the top portion of this panel of the JavaBean Wizard, specify one of the following three project association options:

Option	What to do
Add to open project	If you currently have one or more projects open in Workbench, you can add the bean to one of those projects by selecting it from the list.
Create project	If you do not have a project open in Workbench but want to associate the bean with a project, click Create project to start the New Project Wizard. When you are through, the new project is selected as the project to add the new bean to.  For more information, see “Project design considerations” on page 53.

Option	What to do
No project—just write the files to the disk	Choose this option if you do not want to associate the bean with a project; the wizard will create the bean in a nonproject directory on the file system.

2. On the lower portion of this JavaBean Wizard panel, specify the following options:

Option	What to do
Base directory	<p>If you specified a project, the default base directory is a src subdirectory located directly under the project directory. Otherwise, this field is empty.</p> <p>Click Browse to specify a file system location.</p> <p>The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.</p>
Package	<p>Specify the fully-qualified Java package name for the new bean class. You can specify a package hierarchy with levels separated by periods.</p> <p>The bean you are creating is saved in the Base directory combined with the Package directory.</p> <p>For example, if the base directory is <i>ProjectDir/classes</i> and you specified com.myco as the package, the bean will be created in <i>ProjectDir/classes/com/myco</i>.</p>
File directory	<p>The contents of Base directory and Package are combined to specify the location of the bean, which is displayed under File directory.</p> <p>This is the file system location where the wizard creates the bean source file.</p> <p>You cannot change the contents of this field directly; you must change Base directory and/or Package.</p>
Add the files to the root of the archive	Adds the compiled JavaBean to the archive root combined with the package path when generating the project archive.

Option	What to do
Add the files to the archive with this prefix	Adds the compiled JavaBean to the specified archive directory combined with the package path when generating the project archive. If you specified a package name, the directory structure associated with that package is added to the prefix to determine the final archive path for the generated bean.
The files will be added to this location in the archive	The location in the archive of the generated JavaBean, as specified by the two preceding selections, are reflected in this field. You cannot change the contents of this field directly; you must change the preceding two selections.

3. Click **Finish**.
4. When the final wizard panel reports it's done creating the JavaBean, click **OK**.
The code appears in the Java Editor. (The JavaBean is added to the open project only if you selected that option.)

The wizard creates the skeleton of the JavaBean source file. The skeleton includes an empty constructor, declarations for all the data fields (as `m_name`), and get and set methods for all the fields.

Tag Handler Wizard

Use the Tag Handler Wizard to create tag handler classes for custom JSP tags. The following sections describe:

- About the Tag Handler Wizard
- Starting the EJB Wizard
- Specifying the class name and other options
- Specifying the project, directory, and package
- Specifying the tag library descriptor file
- Specifying the body type
- Specifying tag handler attributes
- Specifying tag handler scripting variables

- Specifying TagExtraInfo class
- What happens

About the Tag Handler Wizard

The Tag Handler Wizard can speed your JSP development effort by:

- Creating a skeleton of the tag handler class
- Creating or modifying the associated tag library descriptor file (TLD)
- Updating the web.xml file to include the required information

You can edit the tag handler class using the Java Editor. You can modify the TLD or the web.xml files using the XML Editor. Both files are in the Project tab of Workbench.

Starting the Tag Handler Wizard


➤ To start the Tag Handler Wizard:

1. Click **File>New**.
2. On the J2EE tab, choose **Tag handler** and click **OK**. (Alternatively, you can double-click **Tag handler**.)
3. Continue as described in “Specifying the class name and other options” on page 62.

Specifying the class name and other options

➤ To specify the class name and other options:

1. On the first panel of the Tag Handler Wizard, specify the following options:


Option	What to do
Class name	Specify the name for the tag handler. The name must be a valid Java name. You do not need to specify the .java extension. This value is added to the TLD file in the <tagclass> element.
Tag name	Specify the name of the custom tag. This will appear in the <name> element of the tag definition in the tag library descriptor file (TLD).
Template	Specify a code-generation template, or accept the default. Your implementation of Workbench may or may not contain additional extension templates of classes that are commonly used as starting points for applications in your environment.  For more information on extensibility, see “Extending the Workbench toolset and services” on page 50.
Attributes	Select this check box if the custom tag should support tag element attributes. If you select this option, you will see an additional wizard panel where you can specify the details of the attribute(s).
Scripting Variables	Select this check box if the custom tag should support scripting variables. If you select this option, you will see an additional wizard panel where you can specify the details of the scripting variable(s).
Body Tag	Select this check box if the custom tag will use the content of the tag element’s body in a JSP page. If you select this option, you will see an additional wizard panel where you can specify the details of the body tag(s).

- Click **Next** to go to the next wizard panel. See “Specifying the project, directory, and package” on page 63.

Specifying the project, directory, and package

➤ To specify the project, directory, and package:

- On the top portion of this panel of the Tag Handler Wizard, specify one of the following three project association options:

Option	What to do
Add to open project	If you currently have one or more projects open in Workbench, you can add the class file to one of those projects by selecting it from the list.
Create project	<p>If you want to associate the class file with a new project, click Create project to start the New Project Wizard.</p> <p>When you are through, the new project is selected as the project to add the new class file to.</p> <p> For more information, see “Project design considerations” on page 53.</p>
No project—just write the files to the disk	<i>Disabled—you must associate the tag handler class with a project.</i>

- On the lower portion of this panel, specify the following options:

Option	What to do
Base directory	<p>The default base directory is the project’s src subdirectory located directly under the project directory.</p> <p>Click Browse to specify a file system location.</p> <p>The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.</p>

Option	What to do
Package	Specify the fully qualified Java package name for the new tag handler class. You can specify a package hierarchy (with levels separated by periods).
File directory	<p>This is the file system location where the wizard creates the tag handler source file and the TagExtraInfo class source file, if any.</p> <p>The tag handler class you are creating is saved in the Base directory combined with the Package directory.</p> <p>For example, if the base directory is <i>ProjectDir/classes</i> and you specified com.myco as the package, the class will be created in <i>ProjectDir/classes/com/myco</i>.</p> <p>You cannot change the contents of this field directly; you must change Base directory and/or Package.</p>
Add the files to the root of the archive	Adds the compiled tag handler class file to the archive root combined with the package path when generating the project archive.
Add the files to the archive with this prefix	<p>Adds the compiled tag handler class file to the specified archive directory combined with the package path when generating the project archive.</p> <p>If you specified a package name, the directory structure associated with that package is added to the prefix to determine the final archive path for the generated class.</p>
The files will be added to this location in the archive	<p>The location in the archive of the generated tag handler class file, as specified by the two preceding selections, are reflected in this field.</p> <p>You cannot change the contents of this field directly; you must change the preceding two selections.</p>

3. Click **Next** to proceed to the next wizard panel. See “Specifying the tag library descriptor file” on page 65.

Specifying the tag library descriptor file

➤ To specify the tag library descriptor file:

1. On the top portion of this panel of the Tag Handler Wizard, choose one of the following options:

Option	What to do
Use Existing TLD	Choose this option when you are adding new custom tags to an existing TLD, then specify the TLD name and disk location
Create New TLD	Choose this option when you are creating a new TLD and specify the remaining fields

2. If you chose **Create New TLD**, complete the following fields:

Option	What to do
Taglib Short Name	Specify the value to be used in the Tab Library Descriptor <short-name> element.
Taglib URI	Specify a URI that is used in the WAR's deployment descriptor. This is not the URI for the TLD file. This URI can be used in the JSP taglib directives to refer to this taglib. For example, <code>/mytags</code> .
TLD file name	Specify the name of the TLD to create.
TLD directory	Specify the directory location where the wizard should create the TLD file.

Option	What to do
Archive location	Specify the directory location for the TLD within the archive: <ul style="list-style-type: none">• When deployed inside a JAR file, the TLD must be in the META-INF directory• When deployed directly in a WAR file, TLDs are usually placed in the \WEB-INF directory or a separate \WEB-INF\tlds directory
JSP Version to support	Choose the radio button that represents the version of the JSP specification that the TLD will support. If you are using a WAR project for J2EE 1.2 (servlet2.2 and JSP1.1) you cannot change the wizard's choice of JSP1.1.

3. Click **Next** to go to the next wizard panel. What panel displays next depends on whether you checked the attributes, scripting variables, or body tag check boxes on the first wizard panel. Follow the first option that fits:
 - If the custom tag uses the tag element's body content, see "Specifying the body type" on page 67.
 - If the custom tag uses tag element attributes, see "Specifying tag handler attributes" on page 67.
 - If the custom tag uses or creates scripting variables, see "Specifying tag handler scripting variables" on page 68.
 - Otherwise, see "Specifying TagExtraInfo class" on page 69.

Specifying the body type

➤ To specify the body type:

1. Specify values for the following options:

Option	What to do
JSP	Specify this option when you want to use JSP code, HTML tags, plain text, other custom tags, and any other valid Web page content in the body of the custom tag.
Tag dependent	Specify this option when you want to use non-JSP code (like SQL) in the body of the custom tag. The tag's body content will be passed directly to the tag handler class without any runtime evaluation.

2. Click **Next** to go to the next wizard panel. What panel displays next depends on whether you checked the attributes or scripting variables check boxes on the first wizard panel.
 - If the custom tag uses tag element attributes, see “Specifying tag handler attributes” on page 67.
 - If the custom tag uses or creates scripting variables, see “Specifying tag handler scripting variables” on page 68.
 - Otherwise, see “Specifying TagExtraInfo class” on page 69.

Specifying tag handler attributes

➤ To specify the tag handler attributes:

1. Specify values for the following options:

Option	What to do
Attribute	Specify the name of the attribute. This value corresponds to the <attribute> element's <name> element within the TLD entry for this custom tag.
Type	Specify the data type of the attribute. Corresponds to the TLD file's element. The value must be a nonprimitive type.

Option	What to do
Required	Specify whether the attribute is required when the custom tag is used. When checked the attribute is required. Corresponds to the TLD file's <required> element.
Runtime expression	Specify whether you can use a JSP scriptlet expression in the JSP page to set the value of the attribute. This corresponds to the TLD file's <rtexprvalue> element.

- Click **Next** to go to the next wizard panel. What panel displays next depends on whether you checked the scripting variables check box on the first wizard panel.
 - If the custom tag uses or creates scripting variables, see “Specifying tag handler scripting variables” on page 68.
 - Otherwise, see “Specifying TagExtraInfo class” on page 69.

Specifying tag handler scripting variables

➤ To specify tag handler scripting variables:

- Specify the values for the following options:

Option	What to do
Variable	Enter the variable's name.
Type	Enter the variable's data type.
New Object	Specify whether the variable refers to a new or existing object instance.
Scope	Specify the availability of the variable. Values can be: <ul style="list-style-type: none"> NESTED—The variable is available between the start and end tags AT_BEGIN—The variable is available from the start tag until the end of the page AT_END—The variable is available after the end tag until the end of the page

- Click **Next** to go to the next wizard panel. See “Specifying TagExtraInfo class” on page 69.

Specifying TagExtraInfo class

➤ To specify whether or not to create a TagExtraInfo class:

1. Specify values for the following options:

Option	What to do
Do not create TagExtraInfo class	Choose this option if you do not want the wizard to create a TagExtraInfo class.
Create TagExtraInfo class	<p>Choose this option if you want the wizard to create a TagExtraInfo class. This option might be disabled if your tag's configuration requires a TagExtraInfo class.</p> <p>(Optional) Choose the appropriate checkbox if you want the wizard to implement either of the following methods:</p> <ul style="list-style-type: none"> • <code>getVariableInfo()</code> • <code>isValid()</code>

2. Click **Finish**. When the final wizard panel reports it has finished creating the TagExtraInfo class, click **OK**.

See “What happens” on page 44.

What happens

When you click **Finish**, the wizard:

- Generates the tag handler class and displays it in the Java Editor.
- Generates the TagExtraInfo class associated with the tag handler (if specified). The class is generated in the same directory with the name *TagHandlerClassNameExtraInfo.java*.
- Adds the tag handler class (and the TagExtraInfoClass if present) to the specified project.
- Creates a new TLD file (and the WAR's deployment descriptor is modified) or updates an existing TLD file, depending on what you chose.

5

Web Service Wizard

This chapter describes the Web Service Wizard of SilverStream eXtend Workbench, which you can use to generate files for implementing and invoking **Web Services**. Topics include:

- About the wizard
- Using the wizard
- Panel sequence
- Panel details



For an introduction to Web Service concepts, standards, and technologies, see the chapter on understanding Web Services in the *Development Guide*.

About the wizard

The Web Service Wizard can perform either of these tasks for you:

- **Generate a standard (SOAP-based) Web Service** that's implemented as a Java remote object. The wizard creates a servlet to handle access to your Web Service and its methods from HTTP SOAP requests.
- **Generate the code needed for a Java-based consumer program** to access any standard (SOAP-based) Web Service. The generated code handles all HTTP SOAP processing under the covers, enabling your consumer program to call the Web Service as a Java remote object and invoke its methods.

In both cases, the wizard produces Java source files based on JAX-RPC (Java API for XML-based RPC) and jBroker Web (the JAX-RPC implementation included with SilverStream eXtend). JAX-RPC is the J2EE specification that provides Web Service support.

You can use the generated files as is or modify them when necessary. The advantage of this Java-oriented approach is that you can deal with Web Services using the familiar technologies of RMI and J2EE instead of coding lower-level SOAP APIs.

How it works Behind the scenes, the Web Service Wizard uses several different compilers to generate the output you request:

The wizard uses this compiler	To generate
Remote interface generator	A Java remote interface from a JavaBean, Java class, or EJB session bean
wSDL2java (from jBroker Web)	A Java remote interface from a WSDL file
xSD2java (from jBroker Web)	Type classes (JavaBeans, marshalers) and mapping files from complex types defined in a WSDL file's XML Schema
rmi2soap (from jBroker Web)	Skeleton and tie classes, stub and service classes, as well as marshalers (for complex types) from a Java remote interface
rmi2wSDL (from jBroker Web)	A WSDL file from a Java remote interface

The wizard determines which compilers to run and in what order depending on the type of input you provide and options you select when filling in its panels.

Alternatives to the wizard You can also run the individual wSDL2java, xSD2java, rmi2soap, and rmi2wSDL compilers manually from a command line. For more information, see the jBroker Web help.

Using the wizard

Here's where you'll learn about preparing to use the Web Service Wizard, running it, and working with its output:

For instructions on	See
Using the wizard to create a new Web Service based on one of these: <ul style="list-style-type: none"> • A JavaBean or other Java class • An EJB session bean • A Java remote interface • A WSDL file 	The chapter on generating Web Services in the <i>Development Guide</i>
Using the wizard to create code for accessing an existing Web Service based on its WSDL file	The chapter on generating Web Service consumers in the <i>Development Guide</i>

Panel sequence

This section lists the panels you need to complete in the Web Service Wizard, depending on your scenario:

If you start with	You step through these panels
A JavaBean or other Java class	<ol style="list-style-type: none"> 1. Project location (and possibly WAR project selection) 2. Class selection 3. Method selection 4. Class-generation and SOAP options

If you start with	You step through these panels
The home interface of an EJB session bean	<ol style="list-style-type: none"> 1. Project location (and possibly WAR project selection) 2. Class selection 3. EJB lookup information 4. Class-generation and SOAP options
The remote interface of an EJB session bean or the SessionBean class itself	<ol style="list-style-type: none"> 1. Project location (and possibly WAR project selection) 2. Class selection 3. EJB home interface selection 4. EJB lookup information 5. Class-generation and SOAP options
A Java remote interface	<ol style="list-style-type: none"> 1. Project location (and possibly WAR project selection) 2. Class selection 3. Class-generation and SOAP options
A WSDL file	<ol style="list-style-type: none"> 1. Project location 2. WSDL file selection (and possibly Multiple namespace mapping) 3. Class-generation and SOAP options

Panel details

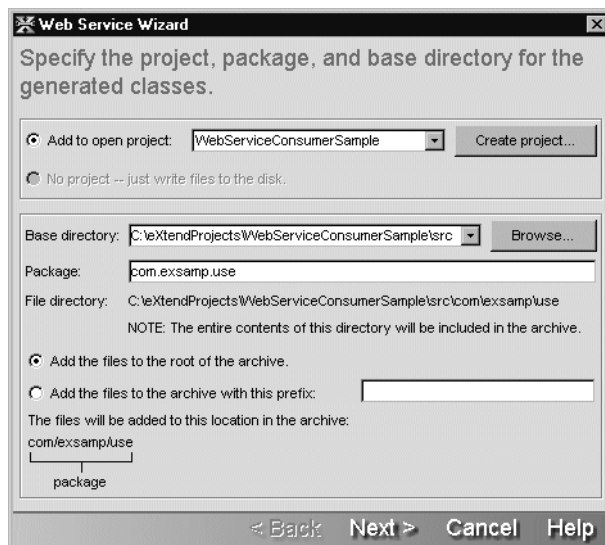
This section describes the options on each panel of the Web Service Wizard. The panels are:

- Project location
- WAR project selection
- Class selection
- WSDL file selection
- Multiple namespace mapping
- EJB home interface selection
- EJB lookup information
- Method selection
- Class-generation and SOAP options

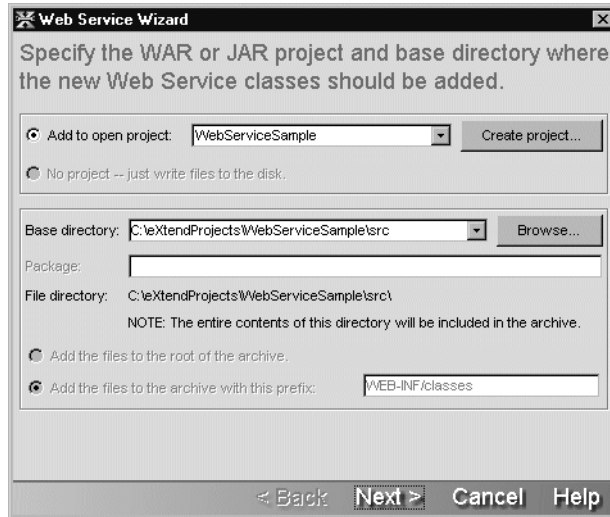
Project location

This panel is used to specify details about the project location (project, directory, package) where the wizard is to store Web Service files it generates. There are two variations of this panel:

- **If you start with a WSDL file, you'll see:**




- **If you start with anything else** (JavaBean, Java class, EJB session bean, or Java remote interface), you'll see:



In this variation, you don't specify a package (because the wizard will get this information from your class or interface, which you supply on an upcoming panel).

➤ **To complete this panel:**

1. Specify the **project**:

Option	What to do
Add to open project	<p>Select a project where the wizard is to store generated files. This option lets you choose from a list of the projects currently open in Workbench.</p> <p>If you're generating a Web Service, you'll typically select a WAR project.</p> <p>When appropriate, you can select a JAR project instead, but then the wizard will prompt for a WAR project to map the Web Service's servlet. See "WAR project selection" on page 9.</p> <p>(When you generate a Web Service from a WSDL file, the wizard does not currently support selecting a JAR project. It requires you to select a WAR project.)</p> <p>If you're generating a Web Service consumer, you can select any type of project.</p>
Create project	<p>Click this button if you want to create a new project to use. It displays the New Project dialog.</p> <p> See "Creating projects and subprojects" on page 56.</p>
No project -- just write files to the disk	<p>(This option is disabled. In the Web Service Wizard, generated files must be added to an open project.)</p>

2. Specify the **directory and package**:

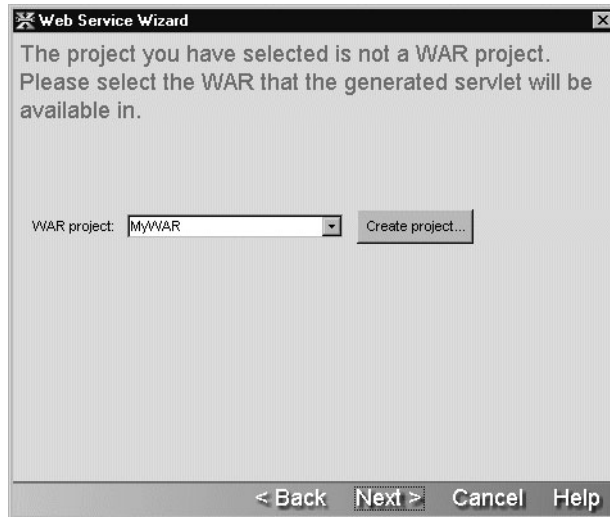
Option	What to do
Base directory	<p>The default base directory is a src subdirectory located right under the project directory on your file system. If you want to select a different file system location, click Browse.</p>
Package	<p>(If enabled) Specify the fully-qualified Java package name to be used for generated classes (for example, com.myco.mypkg).</p>

Option	What to do
File directory	This informational field shows the file system location where generated files will be stored. It is the result of combining Base directory and Package .
Add the files to the root of the archive	(If enabled) Choose this option to place the generated files (and their package path, if any) at the root of the project archive.
Add the files to the archive with this prefix	Choose this option to place the generated files (and their package path, if any) under a specified directory structure (prefix) in the project archive. For a WAR project, the prefix is automatically set to WEB-INF/classes .
The files will be added to this location in the archive	(If displayed) This informational field shows the project archive location where generated files will be stored. It is the result of combining Prefix and Package .

3. Click **Next**.


WAR project selection

This panel is used to specify the required WAR project for a Web Service stored in a JAR project. The wizard will update this WAR's deployment descriptor (web.xml) with the **servlet mapping** for the Web Service.



➤ To complete this panel:

1. Specify the following:

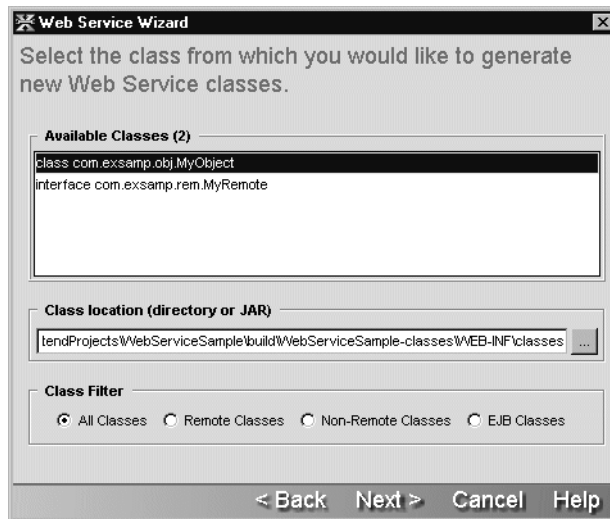
Option	What to do
WAR project	Select the WAR project for the Web Service's servlet mapping. This option lets you choose a WAR project currently open in Workbench.
Create project	Click this button if you want to create a new WAR project to use. It displays the New Project dialog.  See "Creating projects and subprojects" on page 56.

2. Click **Next**.

Class selection

This panel is used to select a compiled class from which the wizard is to generate Web Service files. Supported choices include:

- A JavaBean or other Java class
- An EJB session bean interface or class
- A Java remote interface



By default, this panel finds the compiled classes in the selected project's build directory and lists them in the Available Classes box. For a WAR project, this list comes specifically from WEB-INF/classes in the build directory.

➤ To select from the current list:

1. Click an item in the **Available Classes** list.
2. Click **Next**.

➤ To refine the current list:

- Click one of the **Class Filter radio buttons** to narrow the Available Classes list to classes of a particular kind.

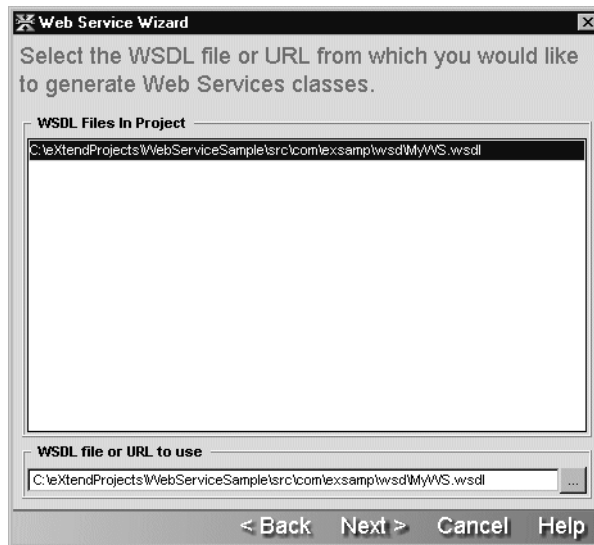
➤ **To list classes from another location:**

- Click the browse (...) button for **Class location (directory or JAR)** to select a different directory or JAR file.
This refreshes the Available Classes list to show just the compiled classes from that new location.

WSDL file selection

This panel is used to select a WSDL file from which the wizard is to generate Web Service files. You can select it from your project, from your file system, or from the Web (by specifying an URL).

NOTE If you're planning to generate a **new Web Service** from a WSDL file, you may need to edit that WSDL file beforehand to make sure the SOAP address in the service definition specifies the correct **binding URL**. The Web Service Wizard will use this URL in the files it generates for your Web Service.



By default, this panel finds the .wsdl files in the selected project and lists them in the WSDL Files In Project box.

➤ **To select from the current list:**

1. Click an item in the **WSDL Files In Project** list to make it the WSDL file to use.
2. Click **Next**.

➤ **To select from the file system:**

1. Click the browse (...) button for **WSDL file or URL to use** to select a WSDL file from your file system.
2. Click **Next**.

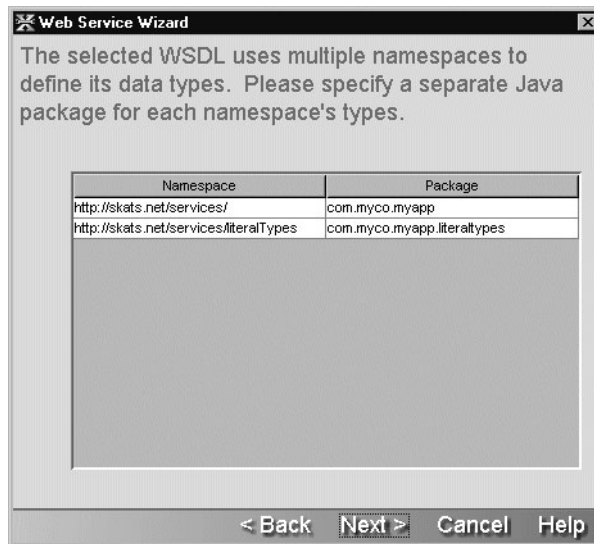
➤ **To specify a file by URL:**

1. Type the URL for the target WSDL file in **WSDL file or URL to use**. For example:
`http://upload.eraserver.net/circle24/autoloan.asmx?wsdl`
2. Click **Next**.

Multiple namespace mapping

This panel is used when you're generating from a WSDL file that uses multiple namespaces for the complex types in its XML schema. It lets you map each namespace to a separate Java package.

NOTE The mappings on this panel are used only if the option **Map complex XML types to Java types** is checked on the next panel (Class-generation and SOAP options).



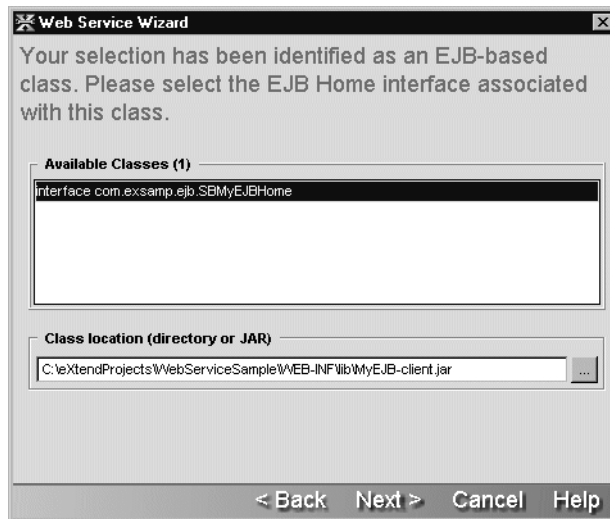
This panel lists the appropriate namespaces and fills in a default package name for each one. You can edit any or all of these package names. Just make sure you specify a unique package name for each namespace.

➤ To edit the namespace-to-package mappings:

1. Double-click any name in the **Package** column to edit it, then type the text you want. (You can't edit the names in the Namespace column.)
2. When you're done editing package names, click **Next**.

EJB home interface selection

This panel is used to select the home interface that corresponds to an EJB session bean class or remote interface you've specified on the class selection panel.



By default, this panel looks in the location of the EJB session bean class or remote interface to find home interfaces (compiled classes that extend `javax.ejb.EJBHome`). If there are any, it lists them in the Available Classes box.

➤ **To select from the current list:**

1. Click an item in the **Available Classes** list to make it the Selected Class.
2. Click **Next**.

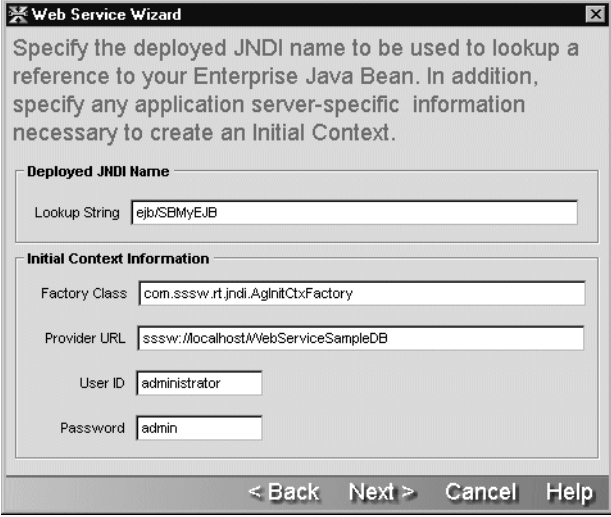
➤ **To list classes from another location:**

- Click the browse (...) button for **Class location (directory or JAR)** to select a different directory or JAR file.

This refreshes the Available Classes list to show just the compiled classes from that new location.

EJB lookup information

This panel is used to specify information that the Web Service will need to do a JNDI lookup for a selected EJB session bean. (JNDI is the Java Naming and Directory Interface.)



The image shows a dialog box titled "Web Service Wizard". The main text reads: "Specify the deployed JNDI name to be used to lookup a reference to your Enterprise Java Bean. In addition, specify any application server-specific information necessary to create an Initial Context." The dialog is divided into two sections: "Deployed JNDI Name" and "Initial Context Information".

Deployed JNDI Name

Lookup String:

Initial Context Information

Factory Class:

Provider URL:

User ID:

Password:

At the bottom of the dialog are four buttons: "< Back", "Next >", "Cancel", and "Help".

This panel displays default initial context values appropriate for looking up a session bean deployed to the SilverStream eXtend Application Server. For information on what other J2EE servers require, consult their documentation.

➤ **To complete this panel:**

1. Specify the **Deployed JNDI Name**:

Option	What to do
Lookup String	Specify the subcontext and JNDI name under which the session bean is registered on the target J2EE server. For example, to look up the session bean whose JNDI name is SBMyEJB in the ejb subcontext: ejb/SBMyEJB

The wizard includes this information in the **ejb-ref** declaration it generates within the deployment descriptor **web.xml**. To learn how it is used at runtime to do a JNDI lookup, see the **getSessionBean()** method of *xxxDelegate.java* (the delegate class generated for the tie servlet).

2. Specify **Initial Context Information**:

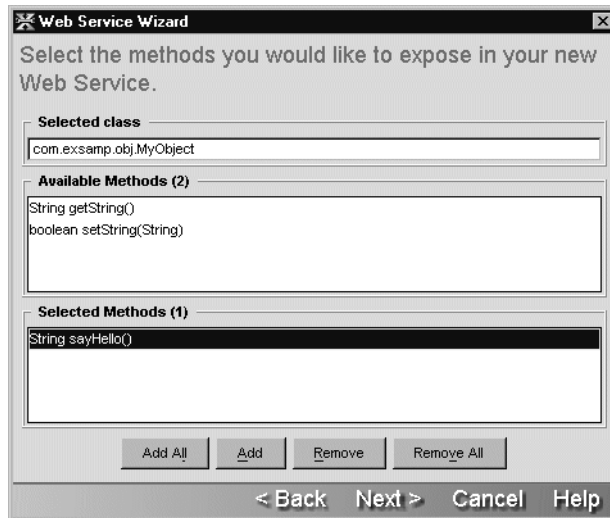
Option	What to do
Factory Class	Specify the package prefix and name of an InitialContext factory class that's appropriate for the target J2EE server.
Provider URL	Specify the URL for the JNDI namespace of the target J2EE server.
User ID	Specify a valid user name that has authority to log on to the target J2EE server and access the session bean.
Password	Specify the password for User ID.

The wizard includes this information in the **servlet** declaration it generates within the deployment descriptor **web.xml**. To learn how these values are used at runtime, see the **getInitialContext()** method of *xxxDelegate.java*.

3. Click **Next**.

Method selection

This panel is used to select the methods you want to expose when generating a Web Service from a JavaBean or other Java class.



This panel examines the selected class and lists its eligible methods in the Available Methods box.

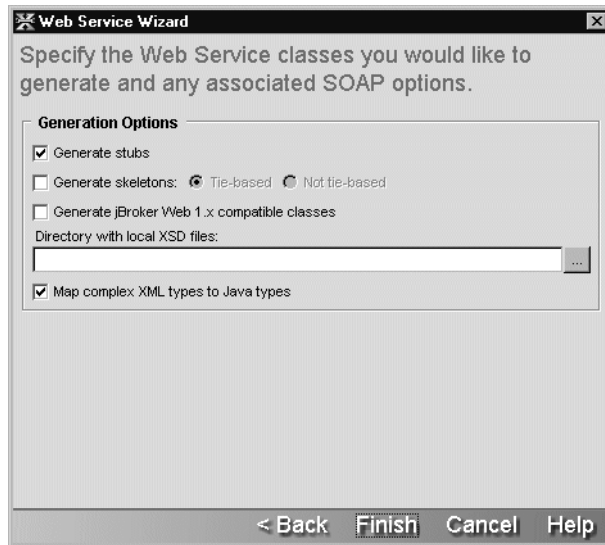
➤ To select methods to expose:

1. Use the **Add** and **Add All** buttons to move one or more items from Available Methods to Selected Methods.
If necessary, you can use **Remove** and **Remove All** to move one or more items back.
2. Click **Next**.

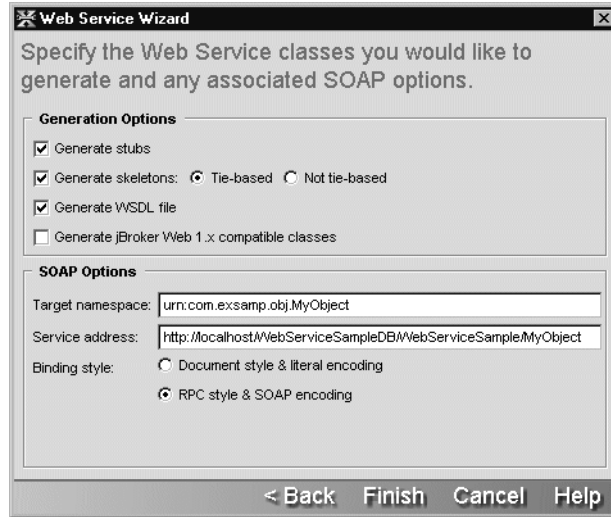
Class-generation and SOAP options

This panel is used to select the Web Service files to generate (including skeleton, tie, and stub classes) and to specify SOAP implementation details to encode in those files. There are two variations of this panel:

- **If you start with a WSDL file, you'll see:**



- **If you start with anything else** (JavaBean, Java class, EJB session bean, or Java remote interface), you'll see:





Note that only this variation provides the SOAP options and the ability to generate a WSDL file.

➤ **To complete this panel:**

1. Specify **Generation Options**:

Option	What to do
Generate stubs	<p>Check this option to generate classes for consuming the Web Service, including service classes, a stub class, and a simple client application. You'll get the following source files:</p> <ul style="list-style-type: none"> • <code>xxxService.java</code> • <code>xxxServiceImpl.java</code> • <code>xxx_Stub.java</code> • <code>xxxClient.java</code>
Generate skeletons	<p>Check this option to generate classes for implementing the Web Service. Then choose one of these implementation models:</p> <ul style="list-style-type: none"> • Tie-based Generates skeleton and tie servlet classes used to handle requests for your Web Service and delegate method calls to a separate implementation class. You'll get the following source files: <ul style="list-style-type: none"> • <code>xxx_ServiceSkeleton.java</code> • <code>xxx_ServiceTieSkeleton.java</code> • <code>xxxTie.java</code> <p>If you start with a JavaBean, Java class, or EJB session bean, you'll also get this source file (used to delegate to your class):</p> <ul style="list-style-type: none"> • <code>xxxDelegate.java</code> • Not tie-based Generates just a skeleton servlet class used to handle requests for your Web Service. You'll get the following source file: <ul style="list-style-type: none"> • <code>xxx_ServiceSkeleton.java</code>

Option	What to do
Generate WSDL file	<p>(If displayed) Check this option to generate the following file:</p> <ul style="list-style-type: none"> • <code>xxx.wsdl</code> <p>It describes your Web Service in standard WSDL format, which is useful when publishing to a registry. The wizard stores this file in the base directory of your source tree (commonly named src).</p>
Generate jBroker Web 1.x compatible classes	<p>Check this option to generate the specified files according to the original jBroker Web (Version 1.x) conventions for:</p> <ul style="list-style-type: none"> • File names • Stub access in client code <p>Except for these conventions, the generated files will conform to the latest version of jBroker Web.</p> <p>This option is appropriate only if you're maintaining an application that originated in jBroker Web 1.x and aren't yet ready to switch to the current conventions (which are based on JAX-RPC and may require some changes to existing code).</p> <p> For details on what this option generates, see the chapter on generating Web Services in the <i>Development Guide</i>.</p>
Directory with local XSD files	<p>(If displayed) When the selected WSDL file relies on imported XSD files for its type definitions, you can optionally specify a local directory that contains copies of them. If the wizard can't access a particular XSD file based on the location specified in the WSDL file, it will look for that XSD file in your local directory.</p> <p> For more information about XSD files, see the WSDL specification.</p>
Map complex XML types to Java types	<p>(If displayed) Check this option if the wizard should try to map complex types defined in the selected WSDL file (via XML Schema) to specific Java types.</p> <p>Uncheck this option if the wizard should map all complex XML types to the <code>org.w3c.dom.Element</code> Java type.</p>

2. Specify **SOAP Options** (if displayed):

Option	What to do
Target namespace	<p>Specify the target namespace for SOAP messages produced by the generated stub and skeleton classes. Method and parameter names are scoped to this namespace when SOAP messages go over the wire.</p> <p>You can accept the default value or specify any string for the namespace. It doesn't have any special semantics beyond providing a scope for SOAP messages.</p> <p>When generating a WSDL file, the wizard uses this value for the targetNamespace definition.</p>
Service address	<p>Specify the URL to be used as the binding for accessing your Web Service. The wizard includes this binding information in the following generated files:</p> <ul style="list-style-type: none"> • The stub class (<i>xxx_Stub.java</i>) and service implementation class (<i>xxxServiceImpl.java</i>) use it as the default URL for binding to the Web Service. • The WSDL file (<i>xxx.wsdl</i>) uses it as the SOAP address in the service definition. <p>The default value for this option includes the name of the selected WAR project and the servlet mapping for the Web Service. For example:</p> <pre>http://localhost/WebServiceSample/MyObject</pre> <p>If you plan to deploy the Web Service to the SilverStream eXtend Application Server, you need to insert the name of the target database in the URL:</p> <pre>http://localhost/WebServiceSampleDB/WebServiceSample/MyObject</pre>

Option	What to do
Binding style	<p>Choose one of the following:</p> <ul style="list-style-type: none"><li data-bbox="564 331 1268 453">• Document style & literal encoding In this format, the SOAP message body contains just the XML document being exchanged and message parts map to elements literally defined in the WSDL file's XML schema<li data-bbox="564 475 1268 597">• RPC style & SOAP encoding In this format, the SOAP message body contains argument and return values, individually wrapped in ad hoc elements that the recipient must interpret by applying specified encoding rules to each message part's type <p>When making this choice, consider the requirements of any other Web Service environments your Web Service must interoperate with. In most cases either style should work, but some environments may favor a particular style.</p>

3. Click **Finish**.

6

Source Editors

Workbench provides two sets of editors, one set based on open-source NetBeans editors and the other set native to Workbench:

- **NetBeans-based editors**
 - Java Editor
 - JSP Editor
 - HTML Editor
 - XML-related editors
- **Native editors**
 - Text Editor
 - Text views of the Deployment Descriptor Editor, Deployment Plan Editor, and WSDL Editor
 - Non-default versions of the Java, JSP, and HTML editors

This chapter describes the basic functionality of these editors:

- Common features
- The NetBeans-based editors
- The native editors

Specialized features Other chapters in this guide cover the specialized features of the following editors:

- Chapter 7, “XML Editors”
- Chapter 8, “WSDL Editor”
- Chapter 10, “Deployment Descriptor Editor”
- Chapter 11, “Deployment Plan Editor”

Common features

This section describes features common to all Workbench editors.

- Standard editing features
- Editor preferences
- Searching across multiple files
- Using text abbreviations
- Changing case
- Changing spaces, tabs, and indentation

Standard editing features

All Workbench editors provide these text-editing features:

To perform this function	Use this menu item
Cut, copy, and paste	Under Edit : <ul style="list-style-type: none">• Cut (or Ctrl+X)• Copy (or Ctrl+C)• Paste (or Ctrl+V)
Undo and redo	Under Edit : <ul style="list-style-type: none">• Undo (or Ctrl+Z)• Redo (or Ctrl+Y)
Select all text	Under Edit : <ul style="list-style-type: none">• Select All (or Ctrl+A)
Toggle the display of line numbers	Under View : <ul style="list-style-type: none">• Line Numbers (or Ctrl+L)

To perform this function	Use this menu item
Find and replace	Under Search : <ul style="list-style-type: none"> • Find (or Ctrl+F) • Find Next (or F3) • Replace (or Ctrl+R) <p>NOTE You can also search across multiple files. See “Searching across multiple files” on page 3.</p> <p>NOTE The native editors provide support for regular expression search. See Regular Expressions for Text Searches in the <i>Reference</i>.</p>
Moving cursor to a line	Under Edit : <ul style="list-style-type: none"> • Go To Line (or Ctrl+G)

Editor preferences

Much of the editor display and behavior can be configured in the Text Editing tab on the Preferences dialog, which you can access using **Edit>Preferences**. This tab contains settings such as font size displayed in the editors, spaces per tab character, whether to show line numbers, and so on.

 For details, see “Text editing preferences” on page 16.

Searching across multiple files

You can search across multiple files at once. You can search through:

- All files that are open in Workbench
- Files that are in all open projects
- Files in a specified open project
- Specified files on the file system

➤ **To search across multiple files:**

1. Select **Search>Find in Files** or press **Ctrl+Shift+F**.

The multiple-search Find dialog displays.

NOTE You can also access the multiple-search feature from the standard Find dialog in the native editors by clicking the **Find in Files** button.

2. Specify the following:

Field	Description
Search for	The text to search for. You can select previously searched text.
Search in	The set of files you want to search through
Direction	Whether to search forward or backward
Match case	Whether the found text must match the case of the search text
Match whole word	Whether the found text must be complete words
Regular expression	Regular expression to search for. For details, see <i>Regular Expressions for Text Searches</i> in the <i>Reference</i> .

3. Click **OK**.

Workbench searches through the specified files. All lines of text containing matching text are listed in the Find tab of the Output Pane.

4. To display the found text, double-click the line of text in the Output Pane. You can view each instance of found text in its corresponding source file:
 - Select **Search>Next Occurrence** (or press **F4**)
 - Select **Search>Previous Occurrence** (or press **Shift+F4**)

Using text abbreviations

You can define abbreviations that can be expanded to one or more lines of text. For example, you can specify that a word can expand to a predefined language construct. For example, the abbreviation **main** might be defined to expand to this code in a Java file:

```
public static void main(String args[])
{
}
}
```

The abbreviations defined in Workbench appear on the Abbreviations tab of the Preferences dialog (**Edit>Preferences**). From this dialog you can define new abbreviations and change or delete existing abbreviations. For details, see “Abbreviations preferences” on page 19.

Once you have defined an abbreviation, you can replace its name with the associated expanded text using **Edit>Text Tools>Complete Abbreviation** (or by pressing **Ctrl+U**).

Changing case

You can easily change the case of text.

To perform this function	Use this menu item
Changing case of word containing insertion point	Under Edit>Text Tools : <ul style="list-style-type: none"> • To Uppercase • To Lowercase

Changing spaces, tabs, and indentation

You can change how the native editors use spaces, tabs, and indentation using the menu items under **Edit>Text Tools**.

To make this change in your text file	Do this
To change spaces to tabs or tabs to spaces	Under Edit>Text Tools : <ul style="list-style-type: none"> • Spaces to Tabs • Tabs to Spaces <p>If you select text before choosing these menu items, only that text is affected; if nothing is selected, the entire file is affected</p>

To make this change in your text file	Do this
To remove trailing whitespace	Under Edit>Text Tools : <ul style="list-style-type: none">• Remove Trailing Whitespace If nothing is selected, this action works on the current line; otherwise it acts on all selected lines
To change the indentation level	Under Edit>Text Tools : <ul style="list-style-type: none">• Shift Right• Shift Left You must select text on at least one line before you can select either of these menu items

The NetBeans-based editors

The core Workbench editors are based on NetBeans, an open-source Java-based framework and set of editors. The core Workbench editors are:

- Java Editor
- JSP Editor
- HTML Editor
- XML-related editors (the XML-related editors have different features than the other NetBeans-based editors and are described in Chapter 7, “XML Editors”)

NOTE Previous releases of Workbench used native versions of the Java, JSP, and HTML editors. The native versions are still provided with Workbench, and you can configure Workbench to use them instead of the NetBeans versions. See “Using the native Java, JSP, or HTML editor” on page 15.

The following sections describe the NetBeans-based editors.

- Color coding
- Code completion
- Adding files types edited by NetBeans-based editors
- Other editing support

Color coding

The NetBeans-based Java, JSP, and HTML editors color-code syntactic elements to make it easy for you to read your code.

The Java Editor uses special colors for these elements:

Syntactic element	Color
Java keyword	Blue
Method call	Bold black
String literal	Red
Numeric literal	Gray
Matching brace	Magenta
Comment	Green italics

The HTML Editor uses colors for these elements:

Syntactic element	Color
Tag	Blue
Tag attribute	Green
Attribute value	Red
Character reference	Red
SGML declaration	Orange
Matching brace	Magenta
Comment	Gray italics

The JSP Editor uses the same colors for Java components as the Java Editor and the same colors for HTML components as the HTML Editor, plus:

Syntactic element	Color
Block of Java	Orange background
JSP tag/directive	Bold blue with gray background
JSP tag attribute	Green
JSP tag attribute value	Magenta
JSP comment	Bold gray

Code completion

As you are coding your Java in the Java Editor (or coding Java in a JSP page using the JSP Editor), you can use the Workbench's **code completion** feature. As you type, you can display a list of possible classes, methods, variables, and so on that can be used to complete the Java expression.

The elements displayed in the Java code completion box are defined by Workbench **parser database files**. Workbench ships with predefined parser files that include the following classes:

- J2EE 1.2
- JDK 1.3
- Servlet 2.3
- xalan and xerces (the versions that ship with Workbench)
- Ant (the version that ships with Workbench)
- jBroker Web (the version that ships with Workbench)

You can create your own parser database files to make your own classes available for code completion. For details, see "Creating parser database files" on page 10.

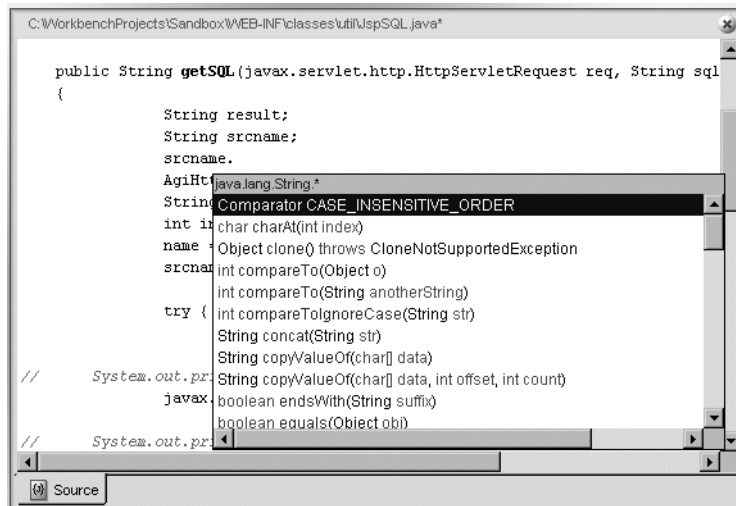
➤ **To complete a Java expression:**

1. In the Java Editor or in a block of Java code in the JSP Editor, type the first few characters of the expression, such as:

```
String srcname;
srcname.
```

2. Press **Ctrl+Space** or **Ctrl+,** or pause after typing a period, a comma, or the keyword **new** or **import** (followed by a space).

The code completion box is displayed, providing a scrolling list of possible classes, methods, variables, and so on that can complete your expression.



In the preceding screen shot, the box lists methods and fields available for strings.

For methods and fields, the code completion box displays only static or only nonstatic options, depending on the context of your code. The options are color-coded:

- Classes, methods, and exceptions are red
- Arguments are magenta
- Interfaces are gray
- Fields are blue

3. While the code completion box is displayed, you can do the following:

Do this	In order to
Continue to type	Dynamically update the list of items in the code completion box based on your current entry
Select an item and press Enter	Insert an item into your code and close the code completion box
Select an item and press Shift+Enter	Insert an item into your code and keep the code completion box open
Press Tab	Insert into your code the letters that are common to all the items in the list and keep the box open
Press Escape	Close the box without inserting anything

Inserting methods If you select a method with arguments, the method is inserted with replacement text for the first argument. If you specify that argument and type a comma, the completion box opens again so you can insert in the next argument, and so on.

Workbench displays the data type of the entered arguments in the title of the code completion box; it displays a question mark if it can't recognize the type. If you enter an argument that does not match any of the recognized argument lists for the method name, all the recognized methods are displayed (along with their argument list), and an asterisk (*) appears for the unrecognized argument(s) in the title of the code completion box.

Creating parser database files

The items displayed in the code completion box are defined by the Workbench **parser database files**. You can have Workbench create database files of your own classes so that they are listed in the code completion box when appropriate.

➤ To create the parser database files:

1. Open a project.
2. Build the project.
3. Select **Edit>Preferences**.
4. Select the **NetBeans Directories** tab.

5. To add your parser files to the same database (directory) as the predefined files, select the directory in the Java Completion Directories box. To put the files in a different database (directory), click **Add**, specify the directory, and select it.
6. Click **Create**.
The Update Parser Databases dialog displays.
7. Specify the following information:

Setting	Description
Parser database file prefix	The names of the parser files that will be created. Workbench will create two files: <i>prefix.jcb</i> and <i>prefix.jcs</i> .
Java source file directory	The root of the directory containing your project's source files, such as <code>c:\myProject\src</code>
Classes, Methods, and Fields	The visibility of the objects you want to include in the database

8. Click **OK** to add the files to the parser database.

If you later make changes to your project and want the changes to be reflected in the code completion lists, you must recreate the parser database files.

Adding files types edited by NetBeans-based editors

By default, Workbench is configured to edit `.java`, `.jsp`, and `.html` files with the NetBeans-based editors. You can specify additional file types to edit with these editors. For example, you might have `.htm` files that you want to edit with the NetBeans-based HTML Editor. You would add `.htm` as a file type for the HTML Editor.

➤ To edit additional file types with NetBeans-based editors:

1. Select **Edit>Preferences**.

2. Select the **Editor Setup** tab.

This tab lists the NetBeans kits that are installed and allows you to specify which you want to use.

The list at the bottom of the tab maps file extensions to a NetBeans kit.

3. To add a file type, click **Add**.

4. Specify the file extension (wild cards are not supported) and the appropriate NetBeans editor kit.
5. Click **OK**.

The additional file type is listed in the Extension mappings table. When you open a file with the specified extension, it will open in the associated NetBeans-based editor.

NOTE You can also specify that you do not want to use a NetBeans-based editor to edit .java, .jsp, or .html files, in which case Workbench will use the corresponding native editor. For details, see “Using the native Java, JSP, or HTML editor” on page 15.

Other editing support

The NetBeans-based editors also provide the following special editing features.

Navigating and selecting text

Keys	Description
Alt+Shift+T	Moves the insertion point to the top of the window
Alt+Shift+M	Moves the insertion point to the middle of the window
Alt+Shift+B	Moves the insertion point to the bottom of the window
Alt+J	Selects the word the insertion point is on, or deselects any selected text
Ctrl+Up Arrow	Scrolls the window up without moving the insertion point
Ctrl+Down Arrow	Scrolls the window down without moving the insertion point

Deleting text

Keys	Description
Ctrl+E	Deletes the current line
Ctrl+H	Deletes the character preceding the insertion point
Ctrl+W	Deletes the current word or the word preceding the insertion point

Searching for text

Keys	Description
Ctrl+F3	Searches for the word the insertion point is in and highlights all occurrences of that word
F3	Moves the insertion point to the next occurrence of the found word
Shift+F3	Moves the insertion point to the previous occurrence of the found word
Alt+Shift+H	Toggles highlighting of words

Changing indentation

Keys	Description
Ctrl+T	Shifts text in line containing the insertion point to the right
Ctrl+D	Shifts text in line containing the insertion point to the left

Bookmarks

Keys	Description
Ctrl+F2	Sets or unsets a bookmark at current line
F2	Goes to next bookmark

The native editors

The core Workbench editors are based on NetBeans and have the features described above. The following editors are native to Workbench and have a different feature set:

- Text Editor
- Text views of the Deployment Descriptor Editor, Deployment Plan Editor, and WSDL Editor
- Non-default versions of the Java, JSP, and HTML editors

The following sections describe the native editors.

- Changing line ending characters
- Multiple clipboard support
- Viewing and changing read-only and read-write attributes
- Using the native Java, JSP, or HTML editor
- Inserting custom tags in a JSP page

Changing line ending characters

You can convert all DOS-style line ending characters to UNIX-style line ending characters by selecting **Edit>Text Tools>Convert to UNIX Line Endings**. To convert all UNIX-style line endings to DOS-style endings, select **Convert to DOS Line Endings**.

NOTE Changing line endings causes no visual change in the editor.

Multiple clipboard support

You can copy or move multiple instances of text. The editor keeps track of your most recently used clipboards. Copying and cutting multiple times creates a clipboard with multiple listings. When you press **Control+Shift+V**, the editor lets you select which text to paste.

Viewing and changing read-only and read-write attributes

When you open a file in a native editor, Workbench displays in the bottom-right corner whether the file has read-only (**RO**) or read-write (**RW**) permission. If a file is in **RO** mode, you cannot make changes to it in the editor. You can switch between **RO** and **RW** mode by clicking on this indicator.

Switching from **RO** to **RW** mode enables you to make changes in the editor. However, the ability to write to the file (for example, using **File>Save**) is still controlled by the file system permissions for that file. You cannot save changes to a file unless the file is marked writable by the file system.

Using the native Java, JSP, or HTML editor

By default, Workbench uses NetBeans-based Java, JSP, and HTML editors (see “The NetBeans-based editors” on page 6). If you want, you can use the native versions of these editors in order to get the functionality described above for the native editors (plus, with the native JSP Editor, you can use the Custom Tag Wizard; see “Inserting custom tags in a JSP page” on page 16).

➤ To use the native Java, JSP, or HTML editors:

1. Select **Edit>Preferences**.
2. Select the **Editor Setup** tab.

This tab lists the NetBeans kits that are installed and allows you to specify which you want to use.

The list at the bottom of the tab maps file extensions to a NetBeans kit.

3. If you do not want to use the NetBeans-based editor for a file type (and instead use the native editor), select the file type in the **Extension mapping** table and click **Remove**.
4. Click **OK** to confirm.

File types no longer in the list will use the corresponding native editor, providing the features described for the native editors.

Inserting custom tags in a JSP page

In JSP pages, custom tags enable you to extend the functionality provided by standard JSP tags, either by writing your own tag library or by using a tag library provided by a third party, such as the Jakarta project. Tag libraries consist of the Java classes that provide functionality for the tags and a tag library descriptor file, an XML document that describes the tag library.

You import a tag library into a JSP page using a **taglib** directive that specifies the location of the tag library descriptor file and declares an identifier that you can use as a prefix to reference the various tags in that library. For example:

```
<%@ taglib uri="/WEB-INF/tlds/app.tld" prefix="apptags" %>
```

references a tag library called **app.tld**, located in the /WEB-INF/tlds directory in the archive. You can refer to tags in the library using the **apptags** prefix. For example, if the tag library contains a tag called **AskUserName**, you could create an instance of that tag in the JSP page using this line:

```
<apptags:AskUserName></apptags:AskUserName>
```

The Custom Tag Wizard

The native Workbench JSP Editor provides a Custom Tag Wizard that enables you to easily insert custom tags into a JSP page.

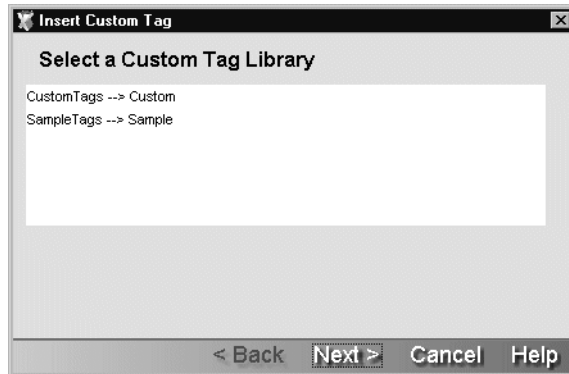
➤ To insert JSP custom tags using the Custom Tag Wizard:

1. Create the classes and descriptor files for your tag library.
2. Add the classes and descriptor files to your Workbench project. A typical location for class files is a WEB-INF/classes directory; for descriptor files, it is typically WEB-INF/tlds.
3. Edit the JSP file in which you want to use the custom tags, adding a **taglib** directive to import the tag library.
4. Position the cursor at the point in the JSP file where you want to insert a custom tag.

5. Select **Edit>Insert Custom Tag>Custom Tag Wizard**.

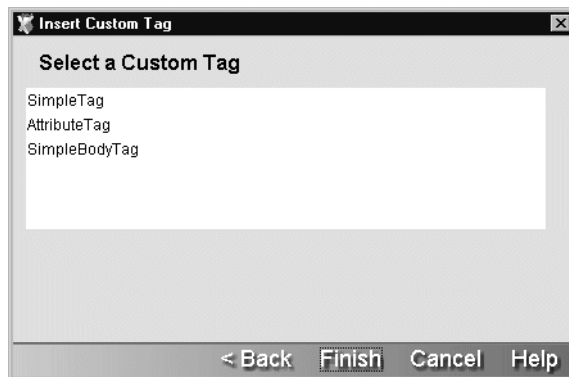
NOTE You must be using the native JSP Editor to access this wizard. See “Using the native Java, JSP, or HTML editor” on page 15).

If the page has more than one **taglib** directive, a list of all tag libraries specified on the page displays. For example:

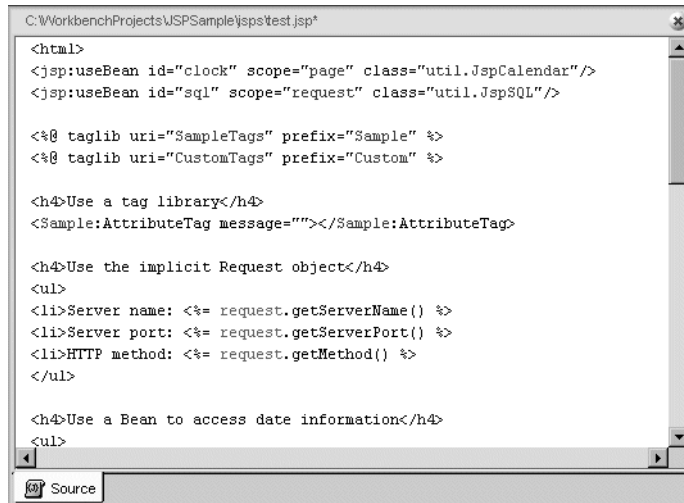


Select the tag library you want to use and click **Next**.

6. If Workbench cannot find the tag library specified, it prompts you to locate it on your file system.
7. Once you have specified the tag library, a list of all tags contained in that library displays. For example:



8. Select the tag you want to insert and click **Finish**. The custom tag code appears in the JSP file. For example:



```
C:\WorkbenchProjects\JSP\Sample\jps\test.jsp
<html>
<jsp:useBean id="clock" scope="page" class="util.JspCalendar"/>
<jsp:useBean id="sql" scope="request" class="util.JspSQL"/>

<%@ taglib uri="SampleTags" prefix="Sample" %>
<%@ taglib uri="CustomTags" prefix="Custom" %>

<h4>Use a tag library</h4>
<Sample:AttributeTag message=""></Sample:AttributeTag>

<h4>Use the implicit Request object</h4>
<ul>
<li>Server name: <%= request.getServerName() %>
<li>Server port: <%= request.getServerPort() %>
<li>HTTP method: <%= request.getMethod() %>
</ul>

<h4>Use a Bean to access date information</h4>
<ul>
```

In this example, the following lines were added manually in Step 3:

```
<%@ taglib uri="SampleTags" prefix="Sample" %>
<%@ taglib uri="CustomTags" prefix="Custom" %>
```

The wizard added this line to instantiate the custom tag:

```
<Sample:AttributeTag message=""></Sample:AttributeTag>
```

9. If necessary, modify the code inserted by the wizard to complete the tag specification. For example, in the tag in the preceding example you would specify a value for the **message** attribute.

This chapter describes the facilities that Workbench provides to work with XML and XML-related files. It contains the following topics:

- About XML
- XML support in Workbench
- Using the XML Editor
- Creating and opening XML documents
- Associating Schemas and DTDs with XML documents
- Converting a DTD to a Schema
- Editing an XML document
- Using the Schema Guide
- Validating an XML document
- Searching an XML document
- Maintaining the XML catalog
- Using the XSL Editor
- Keyboard shortcuts

About XML

XML (Extensible Markup Language) is a language designed to facilitate the exchange of data between computer systems (which can be of different types) and applications on the Web. XML is a project of the World Wide Web Consortium (W3C). It is a standard, public format.

Unlike HTML, XML is **extensible**. It is a **metalanguage**, a language that describes other languages. With XML, you can define customized markup languages to describe any type of document structure. XML can be used to specify the structure of anything from a recipe (which might consist of descriptions, ingredients, preparation steps, and so on) to a Web application (WAR deployment descriptors are XML documents).

The definition of an XML document is specified by either a Document Type Definition (DTD) or a Schema. DTDs, which are older, specify the structure of an XML document. They specify the names of elements, attributes, and entities that can exist in a conforming XML document. DTDs also specify where the elements can be used, whether they are required, and so on.

Schemas are more recent and more powerful. They can specify the structure as well as the content (data types) allowed in XML documents. Unlike DTDs, Schemas are themselves XML documents.



The complete XML standard can be found at <http://www.w3.org/XML>.

TIP If you are new to XML, you might want to read the XML FAQ at <http://www.ucc.ie/xml>. Among other topics, it describes the differences between Schemas and DTDs.

XML support in Workbench

Workbench provides comprehensive support for working with XML files, including:

- XML-related wizards:
 - XML Wizard to create an XML file
 - XML Catalog Wizard to create a catalog entry file
 - DTD to Schema Wizard to convert a DTD to an XML Schema
- XML-related editors:
 - XML Editor
 - XML Catalog Editor
 - XSL Editor

This chapter describes these XML facilities.

Using the XML Editor

The XML Editor lets you:

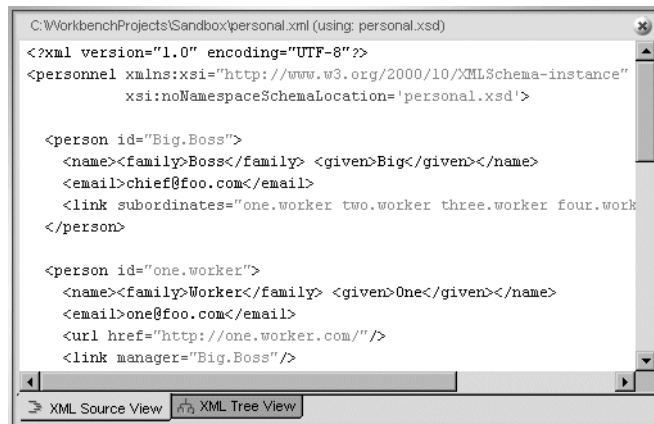
- View and edit XML documents in a syntax-colored Source View or a Tree View
- Easily create and modify document elements through the editor's context-based code completion and the Schema Guide window
- Attach a Schema or DTD to an XML document
- Detach a Schema or DTD
- Validate an XML document against a Schema or DTD
- Convert a DTD to a Schema

Using the Source View

The Source View provides you with a powerful XML source editor. In addition to standard text editing features—including cut-and-paste editing, undo and redo, and searching and replacing text—it supports these specialized features for editing XML files:

- Context-sensitive code completion (see “Editing an XML document” on page 11)
- Formatting of XML elements (see “Modifying text” on page 35 and “Changing indentation” on page 37)
- Navigating by XML elements (see “Moving the insertion point” on page 33)
- Finding matching tags (see “Moving the insertion point” on page 33)
- Bookmarks (see “Bookmarks” on page 37)
- Specifying colors to display different types of information (see “XML Editor color preferences” on page 24)

The XML Editor displays the current XML document in the Source View if you click the **XML Source View** tab.



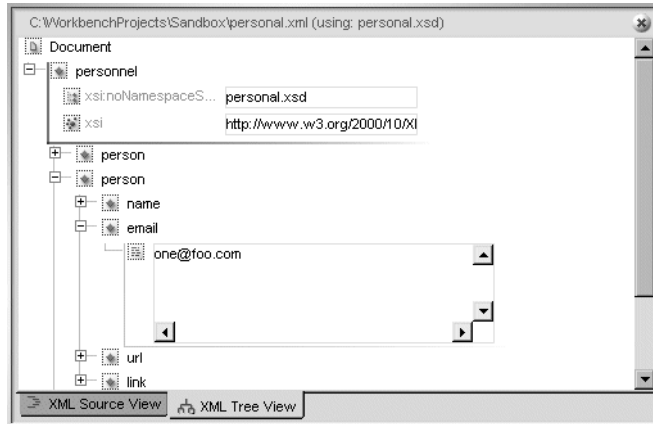
Using the Tree View

The Tree View has special features designed to help you create valid XML documents quickly and easily based on XML Schemas or DTDs. The Tree View supports:

- Context-sensitive editing (see “Editing an XML document” on page 11)
- Cut-and-paste editing (see “Editing objects” on page 16)
- Drag and drop (see “Editing objects” on page 16)

- Searching by name, value, or XPath (see “Searching an XML document” on page 23)
- Finding matching elements (see “Navigation and display” on page 30)

The XML Editor displays the current XML document in the Tree View if you click the **XML Tree View** tab.











NOTE The Tree View does not show or manipulate XML comments.

Tree View display buttons

Icon	Description
	Expand element list (show subelements as well as the text value and CDATA for the element)
	Collapse element list
	Display menu allowing you to toggle the display of all elements' attributes and namespace declarations
	Display menu allowing you to show or hide an element's attributes and namespace declarations

Tree View icons

Icon	Description
	Document
	Element
	Text value of an element (for example, the text value of <code><myTag>the text</myTag></code> is the text)
	CDATA
	Attribute
*	Required attribute
	Namespace declaration
	Search result
	Indicates that the XML cannot be parsed

Creating and opening XML documents

You can create new XML documents or work with existing ones.

➤ To create a new XML document:

1. Select **File>New**.
2. On the **XML** tab, select **XML file**.

3. To create a blank XML document, deselect **Use Wizard** and click **OK**. An empty XML document is created and displayed in the XML Editor.
To use the XML Wizard, select **Use Wizard** and click **OK**. The XML Wizard displays. Go through the wizard as follows.
4. Specify the name and location of the XML file and click **Next**.
5. Specify a Schema or DTD to associate with the XML file. You can:
 - Select a Schema URI from the list of Schemas in the Workbench catalog; the corresponding file name is displayed in the File Name field
 - Select a public or system identifier from the list of DTDs in the Workbench catalog; the corresponding file name is displayed in the File Name field
 - Select a Schema or DTD directly from the file system by clicking **Browse** and selecting the file



For more information about the Workbench catalog, see “Maintaining the XML catalog” on page 24.

6. Click **Finish**.
The XML Editor displays the Schema Guide.
7. You can use the Schema Guide, or click **Close** to edit the file manually.



For information about the Schema Guide, see “Using the Schema Guide” on page 16.


➤ To open an XML document:

1. Select **File>Open**.
2. In the Open dialog, select the XML file and click **Open**.
The file extension must be .XML, .XSD (for a Schema file), or .TLD (for a tag library descriptor file).
The file opens in the XML Editor. If you opened an .XML or .TLD file, there is a new XML Editor item on the menu bar.

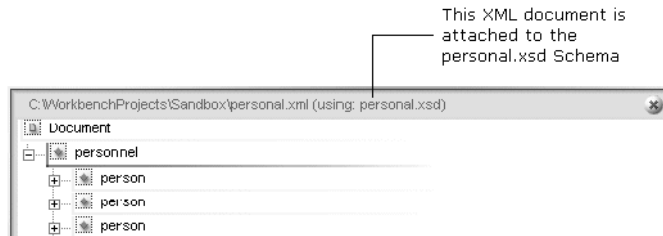
NOTE An XML file might instead be opened by a specialized XML editor, such as the XML Catalog Editor or Deployment Descriptor Editor.

Finding Schemas and DTDs If the XML document specifies a Schema or DTD, Workbench searches for it when opening the document. If Workbench finds the Schema or DTD, it **attaches** it to the XML document. If the reference is unqualified, Workbench first looks in its XML catalog; if Workbench doesn’t find the Schema or DTD there, it looks in the directory containing the XML document.

If the XML Editor cannot find the referenced Schema or DTD, you receive an error message in the Output Pane and the document is opened without being attached to the Schema or a DTD.

 For more information, see “Associating Schemas and DTDs with XML documents” on page 7.

The window title for an XML document specifies whether the document is attached to a Schema or DTD.




Associating Schemas and DTDs with XML documents

In order to use context-sensitive code completion and to validate your document, an XML Schema (.XSD file) or a DTD (.DTD file) **must be** attached to the document.

If Workbench didn't attach a Schema or DTD when opening an XML document, you can manually attach a Schema or DTD or modify your XML document to specify a Schema or DTD and refresh Workbench.

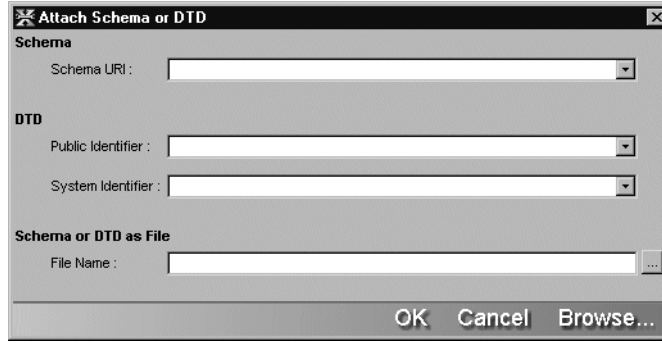
Attaching a Schema or DTD to a document

You can attach a Schema or DTD that is in the Workbench XML catalog or elsewhere on the file system to an open XML document.

 For more information about the Workbench XML catalog, see “Maintaining the XML catalog” on page 24.

➤ **To attach a Schema or DTD to an XML document:**

1. Select **XML Editor>Attach Schema or DTD**.



2. Specify a Schema or DTD to associate with the XML document. You can:
 - Select a Schema URI from the list of Schemas in the Workbench catalog; the corresponding file name is displayed in the File Name field
 - Select a public or system identifier from the list of DTDs in the Workbench catalog; the corresponding file name is displayed in the File Name field
 - Select a Schema or DTD directly from the file system by clicking **Browse** and selecting the file
3. Click **OK**.

The Schema or DTD is now attached to your XML document. You can use the XML Editor's context support for editing, and you can validate your document.

NOTE Attaching a Schema or DTD to an XML document is only for the purpose of context editing and validation in the XML Editor; it doesn't modify the XML document itself. See the next section for permanently associating a Schema or DTD with the document.

Errors Any errors that occur when attaching a Schema or DTD are reported in the Messages tab of the Output Pane.

Specifying a Schema or DTD in the XML document

After opening an XML document, you might want to permanently associate the document with a Schema or DTD and make Workbench aware of the association.

➤ **To associate the document with a Schema or DTD and update Workbench:**

1. Edit the open XML document to specify the associated Schema or DTD. For example, to associate the document with a DTD, edit its DOCTYPE statement.
2. Update Workbench to use the association by selecting **XML Editor>Refresh Schema Handler**.

Workbench parses the XML document and updates the DTD or Schema information associated with the document.

Errors Any errors that occur when updating the Schema or DTD information are reported in the Messages tab of the Output Pane.

Detaching a Schema or DTD

You can detach a Schema or DTD from an open XML document.

➤ **To detach a Schema or DTD:**

- Select **XML Editor>Detach Schema or DTD**.

The Schema or DTD definition is no longer used by the XML Editor. Context editing and validation are no longer provided for the open document.

The Schema or DTD is not permanently detached. The next time you open the XML document, if the document specifies a Schema or DTD that Workbench can find, the Schema or DTD will be attached again.

Converting a DTD to a Schema

Schemas are more powerful than DTDs and are becoming the standard for defining the structure and allowable content type for XML documents. Also, unlike DTDs, Schemas are themselves XML documents and can be edited and validated in the XML Editor.

If you have been using DTDs, you can use Workbench to convert a DTD to a Schema. You can:

- Convert a DTD on the file system to a Schema
- Convert the DTD attached to an open XML document to a Schema

➤ To convert a DTD on the file system to a Schema:

1. Select **File>New**.
2. On the **XML** tab, select **DTD to Schema** and click **OK**.
3. Specify the DTD to convert. You can click the ellipses button to browse the file system for the DTD file. The file must have the extension **.DTD**.
4. Specify the name of the Schema file to generate. Don't provide a file extension; the file will be given the extension **.XSD**.
5. Specify the location to save the Schema file. You can click the ellipses button to browse the file system.
6. Specify whether you want the Schema opened in the XML Editor after it is created.
7. Click **Finish**.

Workbench converts the DTD to a Schema, stores the Schema in the specified location, and displays the Schema in the XML Editor if you specified to open it.

➤ To convert a DTD attached to an open document to a Schema:

1. Attach a DTD to an open XML document.
2. Select **XML Editor>Convert DTD to Schema**.
A file save dialog displays.
3. Specify the name and location of the Schema. Don't provide a file extension; the file will be given the extension **.XSD**.
4. Click **Save**.
The Schema is saved.

What to do next You can edit the generated Schema file in the XML Editor and attach it to an XML document for context editing and validation. If you want to permanently associate the Schema with an XML document, edit the XML document to specify the Schema.

Editing an XML document

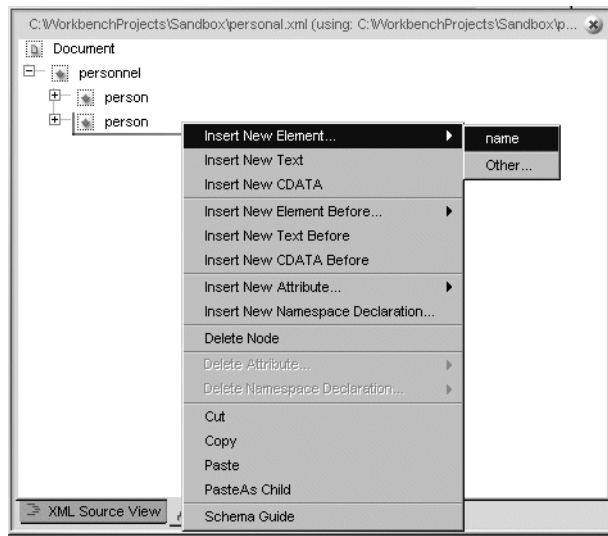
You can edit an XML document using either Tree View or Source View. If you have attached a Schema or DTD, you can use the XML Editor's **context support**.

About context support

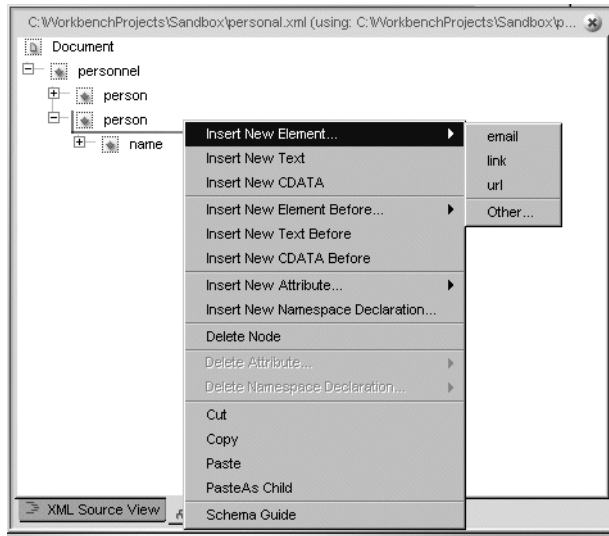
Workbench provides context editing in both the Tree View and the Source View.

Context support in Tree View

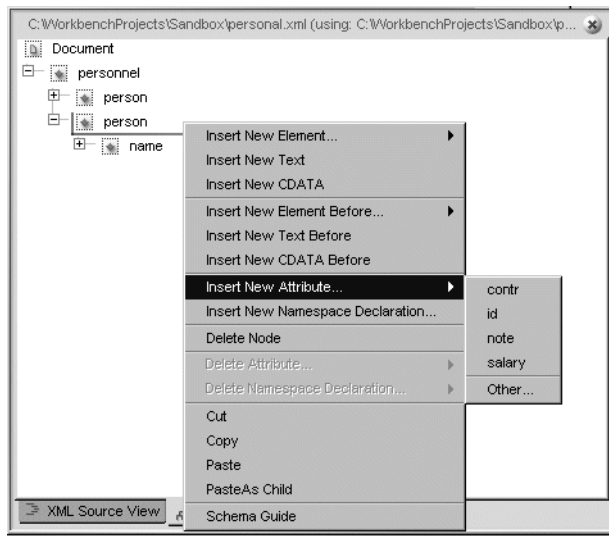
In Tree View, right-click at the appropriate location in the document. In the following illustration, a new person is being added to the document, and the XML Editor detects from the Schema that the next valid element is **name**.



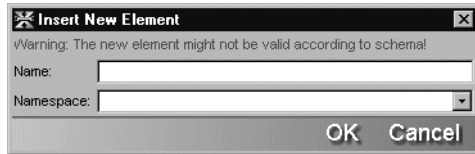
Once the name has been added, the XML Editor presents the new list of valid elements, according to the Schema.



Similarly, the editor presents valid attributes when you have an element selected.



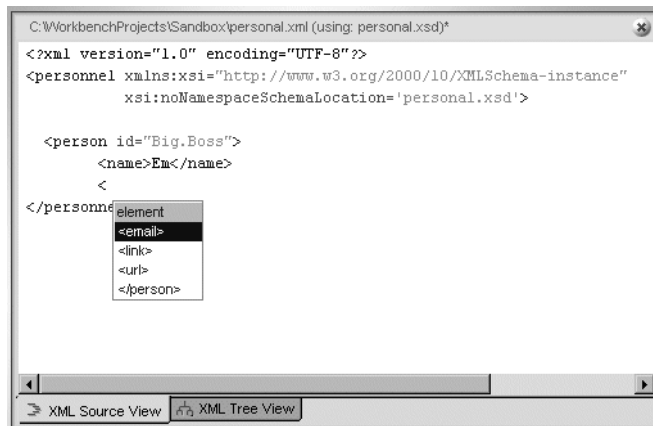
Notice that the editor also provides the choice **Other**, allowing you to define an entry that does not conform to the Schema. If you choose Other, you see a dialog similar to the following:



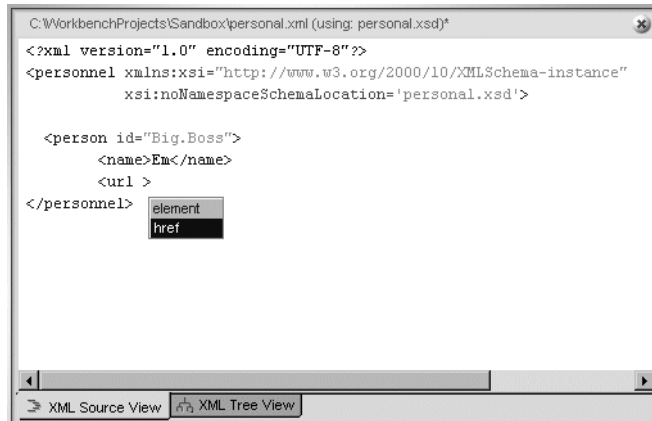
Using the Schema Guide In addition to using the context menu to edit your XML document, you can use the Schema Guide for more comprehensive context support. See “Using the Schema Guide” on page 16.

Context support in Source View

In Source View, after you type < (to start an element tag) or a single space within an element (to define an attribute), the editor displays the valid entries (if there are any). For example:



Here a space has been typed in the url element, which results in a display of the valid attribute, href:



Adding elements

➤ To add an element in Tree View:

1. Select where you want to insert the element.
2. Right-click and select **Insert New Element** to insert an element inside the current element, or select **Insert New Element Before** to insert an element before the current element at the same level.

If valid elements can be inferred from the definition of the document, they will be listed.

If no element can be inferred, you can add an element by choosing **Other**. You are warned that the new element might not be valid.

➤ To add an element in Source View:

1. Position the insertion point where you want to insert the element.
2. Type <.

If valid elements can be inferred from the definition of the document, they will be listed.

If no element can be inferred, you can add an element by typing it.

Adding attributes

➤ To add an attribute in Tree View:

1. Select the element to contain the new attribute.
2. Right-click and select **Insert New Attribute**.

If valid attributes can be inferred from the definition of the document, they will be listed.

If no attribute can be inferred, you can add an attribute by choosing **Other**. You are warned that the new attribute might not be valid.

➤ To add an attribute in Source View:

1. Position the insertion point inside an element where you want to insert the attribute.
2. Type a space.

If valid attributes can be inferred from the definition of the document, they will be listed.

If no attribute can be inferred, you can add an attribute definition by typing it.

Adding namespace declarations

➤ To add a namespace declaration in Tree View:

1. Select the element for the namespace declaration.
2. Right-click and select **Insert New Namespace Declaration**.
The Insert Namespace Declaration dialog displays.
3. Specify the prefix, URL, and Schema location for the namespace.
4. Click **OK**.

➤ To add a namespace declaration in Source View:

- Add the namespace declaration to the element definition.

Editing objects

➤ To copy, move, or delete objects in Tree View:

- You can use drag-and-drop to move objects, or use the right-mouse-button menu to perform the following actions:
 - **Cut** or **Copy** to place an object on the clipboard, then **Paste** to insert it before a selected object or **Paste As Child** to insert it as the last child of a selected object
 - TIP** Cut and Copy also place contents on the system clipboard, so you can paste a textual representation of the tree contents into other applications. Similarly, you can paste textual XML contents from other applications into Tree View.
 - **Delete Node** to delete an element and all its subelements
 - **Delete Attribute** to delete an attribute
 - **Delete Namespace Declaration** to delete a namespace declaration

In all cases, you will be informed if the edit would result in an invalid document. You can choose whether to continue.

➤ To copy, move, or delete objects in Source View:

- Use the standard editing features, including cut and paste, in the editor.

Reversing changes All editing actions can be undone by selecting **Edit>Undo** or pressing **Ctrl+Z**, or redone by selecting **Edit>Redo** or pressing **Ctrl+Y**.

Using the Schema Guide

The context editing functionality described above is very useful when editing XML documents, but doesn't always provide all the information you might want. For example:

- It doesn't show exactly how a Schema (or DTD) is put together and what elements and attributes are allowable at different locations.
- It doesn't indicate whether an element must include a sequence of child elements before it is legal. For example, say element A must have elements B, C, and D as children to be valid. When you insert an instance of A, the standard context support described above suggests element B as a valid subelement. If B is inserted alone the document becomes invalid until you have inserted C and D. With the standard context support, you wouldn't know this unless you perform a full validation of the document.

- If you are looking for a specific element to insert, for instance D in the example above, with the standard context support you wouldn't be informed about D unless you have inserted B and C first.
- If an element contains illegal children, the standard context support doesn't suggest new elements to insert, so you must perform a full validation to find out where the problem is and then correct it.

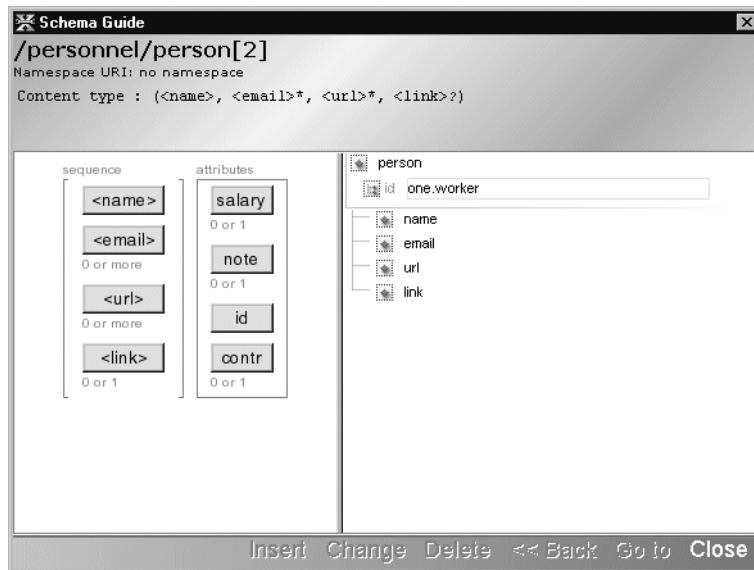
The Schema Guide addresses these situations.

➤ To invoke the Schema Guide:

1. Select Tree View in the XML Editor or XML Catalog Editor.
2. Do one of the following:
 - Right-click an element whose contents you want to edit and select **Schema Guide**.
 - Select an element and press **Ctrl+Shift+G**.

The Schema Guide opens in a new window.

The Schema Guide window



The Schema Guide window consists of four parts:

- The top of the window displays the XPath for the selected element, its namespace, documentation for the element's type (if any, taken from comments in the DTD or annotation elements in the Schema), and a textual DTD-like description of the element's allowed contents



For more information about XPaths, see “XPaths” on page 22.

- The left side contains a graphical representation of the definition of the selected element
- The right side contains a tree representation of the actual instance of the selected element, including its attributes and children (but not its children's children)
- The bottom of the window contains wizard-style buttons

In the screen shown above, the second person element (`/personnel/person[2]`) was selected when the Schema Guide was invoked.

About the left pane

The left pane shows the element's subelements as well as the Schema model groups they belong to (Choice, Sequence, or All) or the model group declarations (for example, `schemaTop`).

- Choice groups are shown with two elements on each row, with a horizontal bracket above and below
- Sequence groups are shown with one element in each row and a vertical bracket on the left and right hand side of the contained elements
- Attributes and All groups are displayed in boxes

Positioning the mouse pointer over an element displays a tool tip describing the element if there is documentation for the element in the Schema or DTD.

The Schema Guide also displays how many instances of each subelement and attribute are allowed. If exactly one of the subelement or attribute is required, no enumeration is shown. Otherwise, the Schema Guide displays the requirement (such as “0 or more”, “0 or 1”, or “1 or more”).

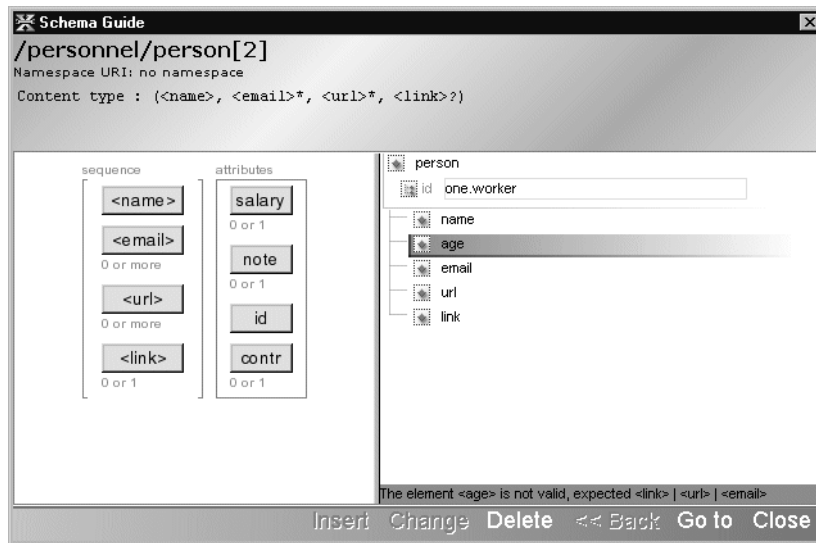
The Schema Guide is invoked automatically when you use the XML Wizard to create an XML document. You can also invoke it when the document is empty and has a Schema attached. In this situation, the Schema Guide lists in the left pane possible root elements. If using a DTD, the description in the header will show the suggested root elements (that is, those elements that are not in the content model of other elements).

About the right pane

The right pane displays the standard Tree View of the XML Editor to show the element that was selected when the Schema Guide was invoked, its attributes, and its immediate children.

Subelements that are not legal are shown with a red background. If the selected element contains an illegal attribute, the element itself is marked with red. Clicking on a colored element displays a similarly colored region of text along the bottom of the tree. The text describes the issue in more detail.

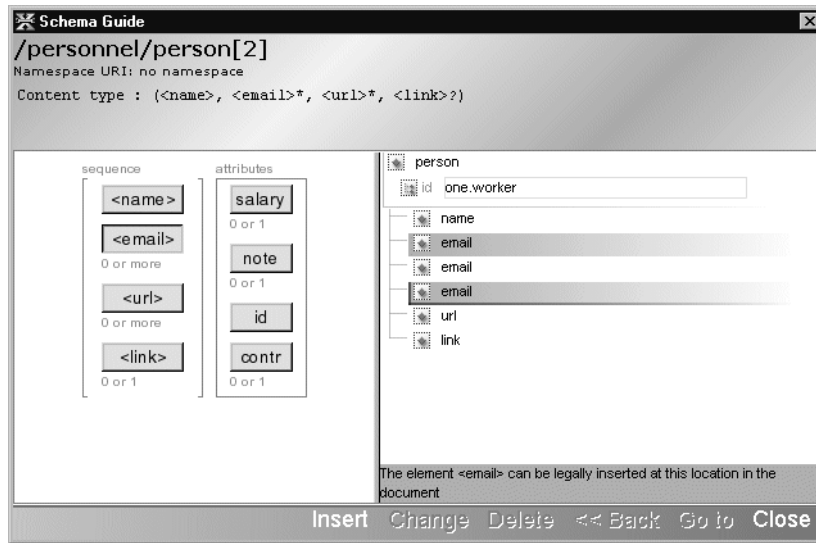
In many cases, the Schema Guide can fix validation errors, either by removing illegal elements or attributes, or by moving an element from a wrong namespace into a correct one. In the following example, the Schema Guide is indicating that the age element is invalid in the person element. You can delete the invalid element by clicking Delete.



Namespace errors are treated separately. These errors are common when dealing with Schemas, because Schemas can contain elements from several namespaces and have different rules for whether specific elements or attributes are required to be in a namespace. An element that has the correct local name for the document to validate correctly but whose namespace is incorrect is shown with a yellow background. You can use the Change button to move the element to the correct namespace.

Adding elements and attributes

Elements When selecting an element in the left pane, the tree view shows where the element can be legally inserted by displaying one or more green nodes in the tree. The following screen shows that an email element can be legally inserted above or below the existing email element.



To insert an element, select one of the green elements in the tree and click **Insert**. If you don't want to insert the element, simply select another object in the left pane to consider.

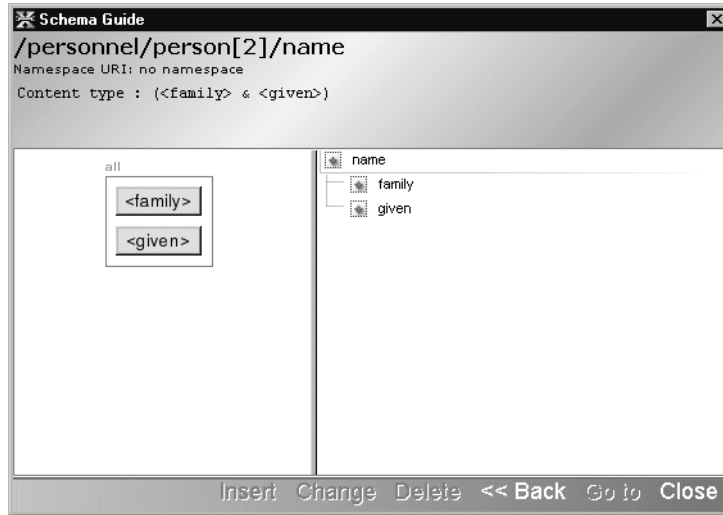
If you click an element in the left pane that cannot be legally inserted, you will not see any green entries in the right pane.

Attributes To add an attribute, select it in the left pane. If it is legal to add, you will see a green attribute in the right pane. Specify the attribute's value and click **Insert**.

Looking at different elements

You can navigate the element hierarchy by selecting a subelement in the right pane and clicking **Go to**. The subelement becomes the selected element and its definition is now shown in the left pane and the tree structure for the selected instance is shown in the right pane. You can work with it the same way you worked with the parent element.

The following screen shows the Schema Guide after the person element's name subelement was selected and Go to was clicked.



Click **Back** to return to working with the parent element.

Validating an XML document

As you type in Source View, the editor automatically highlights in red any areas of the document that are not well formed. The Tree View creates well-formed documents by design.

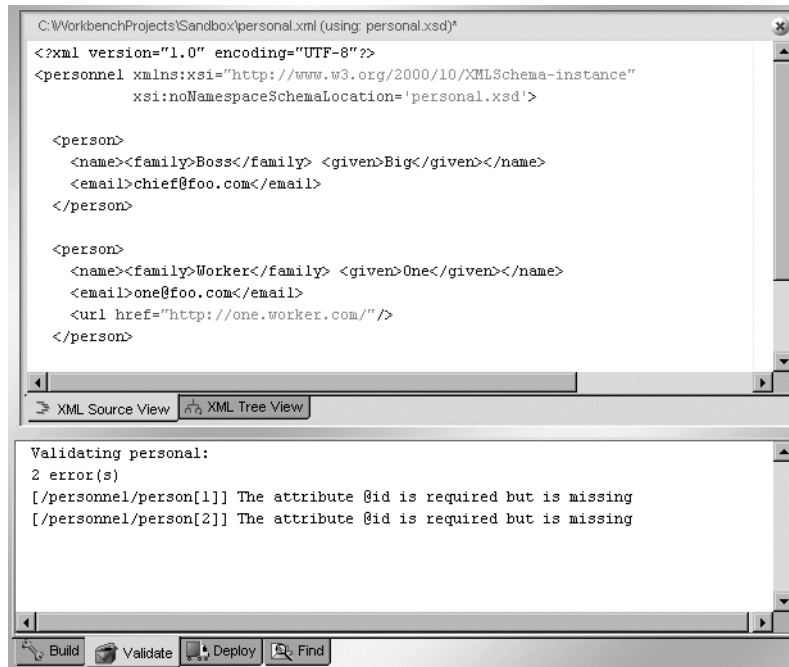
You can also manually validate the document for conformance to the Schema or DTD.

➤ To validate an XML document:

- Select **XML Editor>Validate** to validate an XML document.

NOTE The menu item is enabled only if the XML document is attached to a Schema or DTD.

The editor validates the XML document against the attached Schema or DTD.



The results The report identifying any malformed statements displays in the Validate tab of the Output Pane.

XPaths References to errors are reported as **XPaths**. XPath (XML Path Language) is the W3C-endorsed language for addressing parts of an XML document. It uses a path notation similar to an URL for navigating through the structure of the document. (For more information, see <http://www.w3c.org/TR/xpath>.)

For example, the XPath `/personnel/person[1]` indicates the first instance of person in the XML document, the XPath `/personnel/person[2]` indicates the second instance of person, and so on.

In the preceding example, the `id` attribute is reported as missing from the first two person elements.

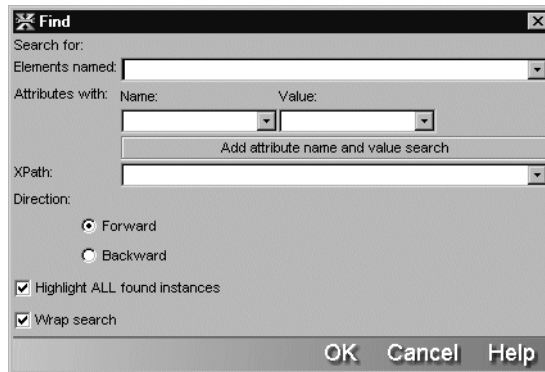
TIP You can search for specific XPaths in Tree View. See “Searching an XML document” on page 23.

Searching an XML document

You can search your document in either Source View or Tree View.

➤ To search an XML document:

1. In either Source View or Tree View, select **Search>Find** or press **Ctrl+F**.
The Find dialog displays.
2. In Source View, you can perform standard text searches. In Tree View, you can specify the following:
 - Element names
 - Attribute names and/or values (see “Searching for attributes in Tree View” on page 23)
 - XPath (see “XPath” on page 22)



3. Click **OK** to search.
If there is a match, the first match is selected and all matching occurrences are indicated:
 - In Source View, matches are highlighted
 - In Tree View, elements containing the found text are indicated with the Search Result icon (🔍)
4. To go to the next occurrence, press **F3**.

Searching for attributes in Tree View

When searching for attributes, you can specify:

- Attribute name only—This will find all attributes of that name, in any element
- Attribute value only—This will find all attributes having the specified value, regardless of the attribute name
- Attribute name and value—This will find all attributes with the given name **and** value

You can click **Add attribute name and value search** to search for elements containing more than one attribute with a given name and/or value. To be matched, an element must match **all** of the specified entries. For example, you could search for all elements having an id attribute **and** a salary attribute.

Maintaining the XML catalog

Workbench provides a built-in catalog of widely used Schemas and DTDs. For example, the catalog includes the Schemas for XSL, WSDL, and XML Schemas; the Sun J2EE DTDs; and the SilverStream deployment plan DTDs.

When you open an XML document that references a Schema or DTD, if the Schema or DTD is in the catalog, Workbench associates it with the XML document and enables context editing and validation.

The Workbench catalog is based on the OASIS XML catalog standard. The OASIS XML catalog standard specifies a format for mapping external identifiers (public and system identifiers) and URI references to other URI references. This makes it possible to map, for example, a URI of a namespace to a local Schema file. The standard specifies that catalogs consist of one or more **catalog entry files**, each file specifying a set of catalog entries.



For information on the OASIS standard, see <http://www.oasis-open.org/committees/entity/spec.html>.

The built-in Workbench catalog consists of three directories in the Workbench Resources directory:

- SchemaCatalog, which contains a set of Schemas
- DTDCatalog, which contains a set of DTD files
- CatalogFiles, which contains catalog entry files

About the catalog entry files There are four built-in catalog entry files:

- dtdcatalog.xml, which lists all the preinstalled DTDs in the DTDCatalog directory
- schemacatalog.xml, which lists all the preinstalled Schemas in the SchemaCatalog directory
- user-dtdcatalog.xml and user-schemacatalog.xml, which are initially empty; you can use them to add entries to the catalog

The two DTD-related catalog entry files both point to DTD files in the DTDCatalog directory, that is, their **base directory** is specified as **../DTDCatalog**. Similarly, the two Schema-related catalog entry files both point to Schemas in the SchemaCatalog directory, that is, their base directory is **../SchemaCatalog**.

An example For example, say you are working with the personal.xsd document that contains this declaration:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Because the built-in Workbench catalog entry file schemacatalog.xml lists this URI and maps it to XMLSchema.xsd in the SchemaCatalog directory, when you open personal.xsd, Workbench locates its Schema in the local catalog without having to go out to the Internet for it.

Adding to the catalog

You might have Schemas and/or DTDs that you want to add to the Workbench catalog so they can be located when you open XML documents that use them. You can add Schemas and DTDs using the existing catalog structure or by extending the structure.

Maintaining the existing structure The easiest way to add entries to the Workbench catalog is by using the existing catalog directory structure.

➤ To add to the Workbench catalog using the existing structure:

1. Add the .DTD or .XSD file to the DTDCatalog directory or SchemaCatalog directory.
2. Open the corresponding user-editable catalog entry file in the Workbench Resources\CatalogFiles directory.
 - user-dtdcatalog.xml, whose base directory is the Workbench DTDCatalog directory
 - user-schemacatalog.xml, whose base directory is the Workbench SchemaCatalog directory
3. Add the catalog entries to the file.

You edit catalog entry files with the XML Catalog Editor, as described in “Using the XML Catalog Editor” on page 26.

Extending the catalog structure You can also add entries to the XML catalog by extending the directory structure, that is, by creating additional directories of Schemas and DTDs and additional catalog entry files.

➤ **To add to the catalog by extending the directory structure:**

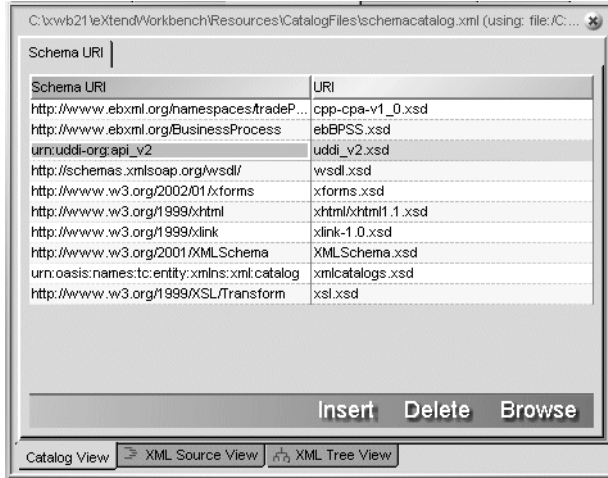
1. Add the .DTD and/or .XSD files you want in the catalog to directories on your file system. You can organize the DTDs and Schemas any way you want, but you will need to create a catalog entry file for each directory containing DTDs or Schemas.
2. Create catalog entry files for each of the directories using the XML Catalog Wizard:
 1. Select **File>New**.
 2. On the **XML** tab, select **XML Catalog file** and click **OK**.
The XML Catalog Wizard displays.
 3. Specify the name of the catalog entry file.
 4. Specify its location. In order to have Workbench read the catalog entry file, place the file in the Workbench Resources\CatalogFiles directory.
 5. Specify the base URI, that is, the path to the directory containing the DTD or Schema files. It is through this base URI that Workbench is able to find the DTDs or Schemas listed in the catalog entry file.
 6. Click **Finish**.
The catalog entry file is opened in the XML Catalog Editor.
 7. Add entries as described in “Using the XML Catalog Editor” on page 26.

Using the XML Catalog Editor

When you open a catalog entry file, Workbench displays it in the XML Catalog Editor.

The XML Catalog Editor has three views:

- A Tree View and Source View, which are the same as the corresponding views in the core XML Editor
- A Catalog View, which presents an interface to the catalog entries



The Catalog View has one or more tabs:

- A catalog entry file whose base directory is Resources/DTDCatalog has two tabs: Public Identifier and System Identifier
- A catalog entry file whose base directory is Resources/SchemaCatalog has one tab: Schema URI
- A catalog entry file whose base directory is any other directory has three tabs: Schema URI, Public Identifier, and System Identifier

➤ To add a catalog entry:

- Depending on whether the entry is for a Schema or DTD, select the appropriate tab and click **Insert**.
 - For a Schema, specify the Schema URI and the resolved URI
 - For a DTD, specify either the public or system identifier and the resolved URI

You can also edit and remove entries from the catalog entry file.

CAUTION *Don't delete preexisting DTDs or Schemas from the catalog, because Workbench might require them.*

➤ **To edit an entry:**


- Double-click the entry and make whatever edits you want. If you double-click a resolved URI value, the **Browse** button is enabled, allowing you to pick another file.

➤ **To delete an entry:**

- Select the entry and click **Delete**.
The entry is removed from the catalog entry file (the Schema or DTD itself is unaffected).

Using the XSL Editor

Workbench provides an XSL Editor for you to create and maintain XSL style sheets.


 For complete information about XSL, see <http://www.w3.org/Style/XSL>.



➤ **To create an XSL file:**

1. Select **File>New**.
2. On the **XML** tab, select **XSL file** and click **OK**.

Workbench generates a skeleton XSL document and displays it in the XSL Editor.

Using the XSL Editor The XSL Editor provides an environment for editing, validating, and testing XSL files.

Task	Description
Editing	Use the XSL tab. You can use the keyboard shortcuts listed under “In Source View” on page 33.
Validating the style sheet	Select XSL Editor>Validate . Workbench displays the results in the Validate tab in the Output Pane.  For more information, see “Validating an XML document” on page 21.

Task	Description
Attaching or detaching Schemas/DTDs	<p>Select XSL Editor>Attach Schema or DTD or XSL Editor>Detach Schema or DTD.</p> <p> For more information, see “Associating Schemas and DTDs with XML documents” on page 7.</p>
Testing transformations	<p>The Result tab allows you to see the result of the transformation specified by the XSL file:</p> <ol style="list-style-type: none">1. Select XSL Editor>Transform or press Ctrl+Shift+T and specify a file you want to apply the XSL style sheet to.2. Click the Result tab to see the result. <p>Workbench displays the result of the transformation in the Edit Pane. If the transformation failed, errors are listed in the Messages tab in the Output Pane.</p> <ol style="list-style-type: none">3. View the result of the transformation rendered in the default browser by selecting XSL Editor>View in browser. (You can also select this menu item from the XSL tab to specify an XML file to transform and render in a browser.) <p> For information about the default browser, see “General preferences” on page 14.</p>

Keyboard shortcuts

Here are the keyboard shortcuts provided in the XML Editor, XML Catalog Editor, and XSL Editor.


In Tree View

Navigation and display

Keys	Description
Ctrl+A	Expands all
Ctrl+Shift+A	Collapses all
Ctrl+E	Expands element group
Ctrl+Shift+E	Collapses element group
Up Arrow	Navigates to previous visible node
Down Arrow	Navigates to next visible node
Left Arrow	Collapses element group
Right Arrow	Expands element group
Alt+Up Arrow	Navigates to previous sibling (element within an element group)
Alt+Down Arrow	Navigates to next sibling
Alt+Left Arrow	Navigates to parent
Alt+Right Arrow	Navigates to first child
Alt+Page Up	Navigates to previous cousin (element with the same element path to the root)
Alt+Page Down	Navigates to next cousin
Ctrl+Up Arrow	Hides the selected element's attributes

Keys	Description
Ctrl+Down Arrow	Displays the selected element's attributes
Ctrl+Left Arrow	Hides the selected element's namespace declarations
Ctrl+Right Arrow	Displays the selected element's namespace declarations
Ctrl+Shift+Up Arrow	Hides the selected element's attributes and namespace declarations
Ctrl+Shift+Down Arrow	Displays the selected element's attributes and namespace declarations
Ctrl+Q	Toggles the display of attributes for all displayed elements
Ctrl+Shift+Q	Toggles the display of namespace declarations for all displayed elements
Ctrl+Alt+Shift+Q	Toggles the display of attributes and namespace declarations for all displayed elements
Ctrl+Shift+G	Displays the Schema Guide for the selected element

Searching for text

Keys	Description
Ctrl+F	Shows Find dialog
F3	Navigates to next search result
Shift+F8	Finds matching elements (displays Search Result icon, )
Alt+Shift+H	Toggles display of Search Result icon
Ctrl+Alt+Shift+H	Clears the search
F9	Finds next matching element
Shift+F9	Finds previous matching element

Editing text

Keys	Description
Ctrl+X	(Cut) Cuts the current selection to the clipboard
Ctrl+C	(Copy) Copies the current selection to the clipboard
Ctrl+V	(Paste) Pastes the contents of the clipboard at the insertion point
Ctrl+Shift+V	Pastes the contents of the clipboard as the last child of the selected element
Del	(Delete) Deletes the current selection
F5	Refreshes and collapses the tree
Ctrl+Z	Reverses editor actions (except save)
Ctrl+Y	Reverses Undo actions
Ctrl+L	Inserts new element as last child
Ctrl+T	Inserts new text as last child
Ctrl+D	Inserts new CDATA as last child
Ctrl+Shift+L	Inserts new element before selected node
Ctrl+Shift+T	Inserts new text before selected node
Ctrl+Shift+D	Inserts new CDATA before selected node
Ctrl+K	Inserts new attribute
Ctrl+Shift+K	Deletes selected attribute
Ctrl+M	Inserts new namespace declaration
Ctrl+Shift+M	Deletes selected namespace declaration

In Source View

Moving the insertion point

Keys	Description
Left Arrow, Right Arrow	Moves the insertion point one character to the left or right
Ctrl+Right Arrow	Moves the insertion point one word to the right
Ctrl+Left Arrow	Moves the insertion point one word to the left
Home	Moves the insertion point to the beginning of the line
End	Moves the insertion point to the end of the line
Up Arrow	Moves the insertion point one line up
Down Arrow	Moves the insertion point one line down
Alt+Shift+T	Moves the insertion point to the top of the window
Alt+Shift+M	Moves the insertion point to the middle of the window
Alt+Shift+B	Moves the insertion point to the bottom of the window
Ctrl+Home	Moves the insertion point to the beginning of the document
Ctrl+End	Moves the insertion point to the end of the document
PgUp	Moves the insertion point one page up
PgDn	Moves the insertion point one page down
Alt+Shift+F8	Moves the insertion point to matching begin/end tag
Alt+Up Arrow	Moves the insertion point to previous sibling (element within an element group)
Alt+Down Arrow	Moves the insertion point to next sibling
Alt+Right Arrow	Moves the insertion point to first child

Keys	Description
Alt+Left Arrow	Moves the insertion point to parent
Ctrl+G	Displays Go to Line dialog
Ctrl+L	Toggles display of line numbers

Selecting text

Keys	Description
Ctrl+A	Selects all text in the document
Shift+Right Arrow	Selects the character to the right of the insertion point
Shift+Left Arrow	Selects the character to the left of the insertion point
Alt+J	Selects the word the insertion point is on, or deselects any selected text
Ctrl+Shift+Right Arrow	Selects the word to the right
Ctrl+Shift+Left Arrow	Selects the word to the left
Shift+Home	Selects text to the beginning of the line
Shift+End	Selects text to the end of the line
Shift+Up Arrow	Selects text to the previous line
Shift+Down Arrow	Selects text to the next line
Ctrl+Shift+Home	Selects text to the beginning of the document
Ctrl+Shift+End	Selects text to the end of the document
Shift+PgUp	Selects text one page up
Shift+PgDn	Selects text one page down

Scrolling text

Keys	Description
Alt+U T	Scrolls line containing insertion point to top of window TIP Press and release Alt+U, then press T
Alt+U M	Scrolls line containing insertion point to middle of window TIP Press and release Alt+U, then press M
Alt+U B	Scrolls line containing insertion point to bottom of window TIP Press and release Alt+U, then press B
Ctrl+Up Arrow	Scrolls the window up without moving the insertion point
Ctrl+Down Arrow	Scrolls the window down without moving the insertion point

Modifying text

Keys	Description
INSERT	Switches between insert text and overwrite text modes
Alt+U U	Makes the selected characters or the character to the right of the insertion point uppercase TIP Press and release Alt+U, then press U
Alt+U L	Makes the selected characters or the character to the right of the insertion point lowercase TIP Press and release Alt+U, then press L
Alt+U R	Reverses the case of the selected characters or the character to the right of the insertion point TIP Press and release Alt+U, then press R
F11	Reformats the tag the insertion point is on

Keys	Description
Shift+F11	Reformats the entire document
Ctrl+Alt+O	Opens the tag (for example, converts <a/> to <a>)
Ctrl+Alt+C	Closes the tag (for example, converts <a> to <a/>)

Cutting, copying, pasting, and deleting text

Keys	Description
Ctrl+Z	(Undo) Reverses (one at a time) a series of editor actions, except Save
Ctrl+Y	(Redo) Reverses (one at a time) a series of Undo commands
Ctrl+X	(Cut) Cuts the current selection and places it on the clipboard
Ctrl+C	(Copy) Copies the current selection to the clipboard
Ctrl+V	(Paste) Pastes the contents of the clipboard at the insertion point
Delete	(Delete) Deletes the current selection
Ctrl+E	Deletes the current line
Ctrl+H	Deletes the character preceding the insertion point
Ctrl+Shift+Backspace	Deletes text in the following sequence: <ol style="list-style-type: none">1. Text preceding insertion point on same line2. Indentation on same line3. Line break4. Text on previous line
Ctrl+W	Deletes the current word or the word preceding the insertion point

Searching for text

Keys	Description
Ctrl+F3	Searches for the word the insertion point is in and highlights all occurrences of that word
F3	Moves the insertion point to the next occurrence of the found word
Shift+F3	Moves the insertion point to the previous occurrence of the found word
Alt+Shift+H	Toggles highlighting of words
Ctrl+F	Displays Find dialog
Ctrl+R	Displays Replace dialog

Changing indentation

Keys	Description
Tab	Shifts all text to right of insertion point to the right
Ctrl+T	Shifts text in line containing the insertion point to the right
Ctrl+D	Shifts text in line containing the insertion point to the left

Bookmarks

Keys	Description
Ctrl+F2	Sets or unsets a bookmark at current line
F2	Goes to next bookmark

Specifying transformation (XSL Editor)

Keys	Description
Ctrl+Shift+T	Displays dialog for specifying file to transform

In Catalog View, XML Catalog Editor

Modifying text

Keys	Description
Ctrl+B	Displays dialog for changing the base URI, that is, the path to the directory containing the DTD or Schema files for the catalog entry file

The WSDL Editor provides a quick and easy way to create, edit, and view WSDL documents. This chapter contains the following topics:

- About WSDL
- About the WSDL Editor
- Creating a new WSDL document
- Adding elements to a WSDL document
- Validating a WSDL document
- Displaying a stylized view
- Publishing to a registry
- Generating Web Service files from WSDL

About WSDL

WSDL (Web Services Description Language) is a general-purpose XML vocabulary for describing Web Services. Using WSDL, it is possible to describe (concisely and in a standardized manner) the interface, protocol bindings, and deployment details of Web-based services, at a level of detail sufficient for businesses to begin to interact online.



The complete WSDL standard can be found at <http://www.w3.org/TR/wsdl>.

About the WSDL Editor

The WSDL Editor lets you:

- Create and edit WSDL documents (files with the .WSDL extension)
- Easily create any of the four canonical WSDL document elements (message, port type, binding, or service)
- Validate WSDL documents
- View WSDL documents in stylized view and color-coded text view
- Publish WSDL documents to Web Service registries

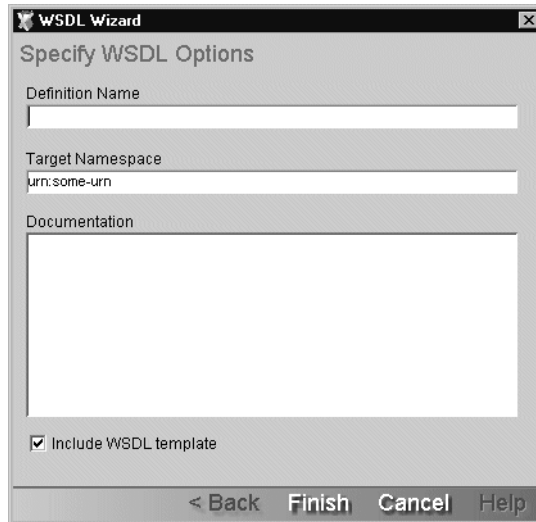
The WSDL Editor also supports the editing features described in Chapter 6, “Source Editors”.

Creating a new WSDL document

➤ **To create a new WSDL document:**

1. Select **File>New**.
2. On the Web Services tab, select **WSDL** and click **OK**.

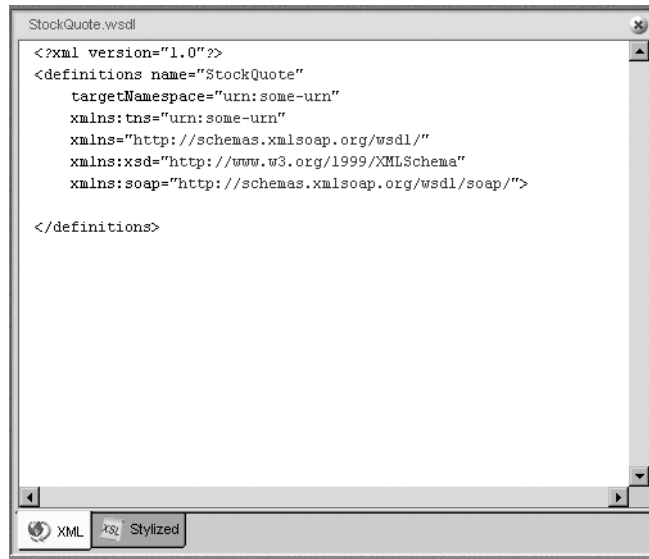
The WSDL Wizard displays.



3. (Optional) Enter a **Definition Name**.
4. (Optional) Enter a **Target Namespace**. This can be the Uniform Resource Name associated with this WSDL document. You cannot specify a relative URN.
5. (Optional) In the **Documentation** text box, enter any human-readable comment or descriptive language you would like to associate with the definition element.
6. Select the **Include WSDL template** check box if you want a skeleton document to be created for you using values provided in this wizard. Leave the check box unselected to start with a blank document.

7. Click **Finish**.

A new WSDL document opens in the WSDL Editor.



Adding elements to a WSDL document

WSDL documents can contain four standard element types: message, port type, binding, and service. These element types build on one another with cascading references; so when you create a WSDL file, you should create the message section first, followed by the port type section, then the binding section, and finally the service section.

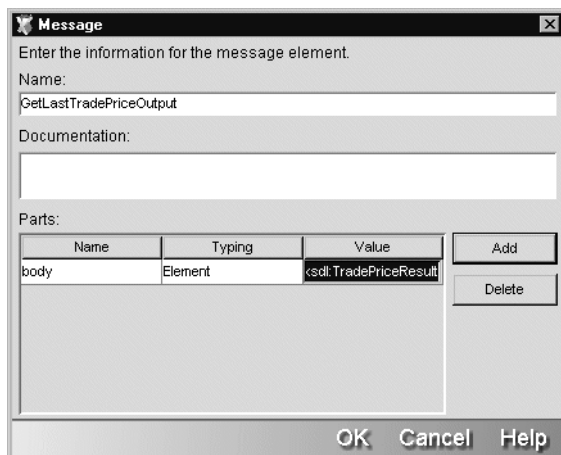
The WSDL Editor offers dialog-based assistance in creating each of the four types.

Adding a message element

In WSDL, a **message** is an abstract definition of the data being exchanged.

➤ To add a message element to a WSDL document:

1. Position the insertion point where you want to insert the definition and right-click. A popup menu displays.

2. Select **Insert WSDL Element>Message**.

3. Specify the following information in the Message dialog:

Option	What to do
Name	Specify the value of the name attribute of the <message> element.
Documentation	(Optional) Specify any human-readable comment or descriptive language you would like to associate with this message.
Parts	Specify this information for each <part> element of your message: <ul style="list-style-type: none"> • The name attribute • The typing value (Element or Type) • Under Value, the element attribute <p>To add another part entry to the message, click Add. To remove an entry, select the entry and click Delete.</p>

4. Click **OK**.

A new message section is added to your document.

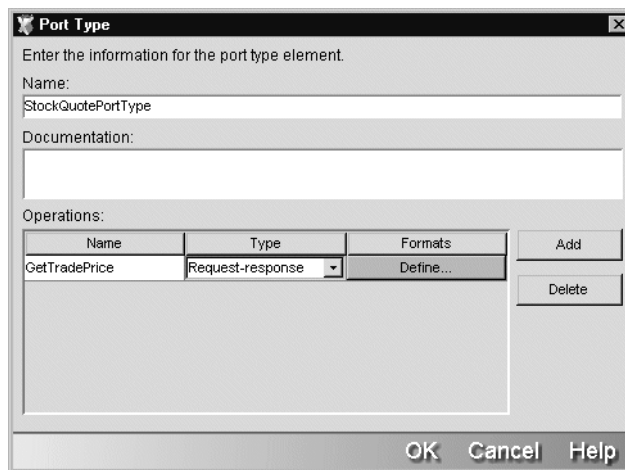
```
<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd:TradePriceResult"/>
</message>
```

Adding a port type element

A WSDL **port type** is an abstract definition of the operations supported by a service and the communications mode (one-way, request-response, and so on) that will be used in the service.

➤ **To add a port type to a WSDL document:**

1. Position the insertion point where you want to insert the definition and right-click. A popup menu displays.
2. Select **Insert WSDL Element>Port Type**.



3. Specify the following information on the Port Type dialog:

Option	What to do
Name	Specify the value of the name attribute of the <portType> element.
Documentation	(Optional) Specify any human-readable comment or descriptive language you would like to associate with this port type.

Option	What to do
Operations	<p>Specify this information for each <operation> element of your port type:</p> <ul style="list-style-type: none"> • The name attribute • The type (One-way, Request-response, Solicit-response, or Notification) • Under Formats, click the Define button to specify the operation's messages using the Define dialog <p>The dialog has several control groups. Only those that are appropriate to the type of operation are enabled. For example, if you chose Notification as the type, only the Output control group is enabled. For each enabled group, you must specify a Name and Message appropriate to the operation for Input and Output. Specifying values for the Fault group is optional.</p> <p>To add another operation entry to the port type, click Add. To remove an entry, select the entry and click Delete.</p>

4. Click **OK**.

A new port type section is added to your document.

```
<portType name="StockQuotePortType">
  <operation name="GetTradePrice">
    <input name="input" message="tns:GetLastTradePriceInput"/>
    <output name="output" message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>
```

Adding a binding element

A WSDL **binding** specifies concrete protocol and data format specifications for the operations and messages defined by a particular port type.

➤ **To add a binding to a WSDL document:**

1. Position the insertion point where you want to insert the definition and right-click. A popup menu displays.

2. Select **Insert WSDL Element>Binding**.

The screenshot shows a dialog box titled "Binding" with the following fields and options:

- Name:** StockQuoteSoapBinding
- Documentation:** (empty text area)
- Port Type:** StockQuotePortType (dropdown menu)
- Binding Protocol:**
 - SOAP Binding
 - Style:** rpc (dropdown menu)
 - Transport:** http://schemas.xmlsoap.org/soap/http (text field)
 - HTTP Binding
 - Verb:** get (dropdown menu)
 - User Defined

Buttons: OK, Cancel, Help

3. Specify the following information on the Binding dialog:

Option	What to do
Name	Specify the value of the name attribute of the <binding> element.
Documentation	(Optional) Specify any human-readable comment or descriptive language you would like to associate with this binding element.
Port Type	Specify the port type for this binding. The dropdown list displays the names of the port types that you have created for this document (see “Adding a port type element” on page 5).
SOAP Binding	If your WSDL document will specify a SOAP binding, select SOAP Binding , then select a Style (RPC or Document) and specify a Transport value.
HTTP Binding	If an HTTP binding will be used, select HTTP Binding and enter the appropriate Verb (GET or POST).

Option	What to do
User Defined	Select if you want to specify a custom binding protocol manually.

4. Click **OK**.

A new binding section is added to your document.

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"
namespace="http://example.com/stockquote.xsd"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="literal"
namespace="http://example.com/stockquote.xsd"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
</binding>
```

Adding a service element

A WSDL **service** names the entry-point address (or addresses) for the Web Service in question. These addresses are in the form of URIs and constitute **ports**.

➤ To add a service to a WSDL document:

1. Position the insertion point where you want to insert the definition and right-click. A popup menu displays.

2. Select **Insert WSDL Element>Service**.

Service

Enter information for the service element.

Name:
StockQuoteService

Documentation:

Ports:

Name	Binding	Address Type	Location
StockQuotePort	StockQuoteSoapBi...	SOAP	http://example.com

Add
Delete

OK Cancel Help

3. Specify the following information on the Service dialog:

Option	What to do
Name	Specify the value of the name attribute of the <service> element
Documentation	(Optional) Specify any human-readable comment or descriptive language you would like to associate with this service.
Ports	<p>Specify this information for each <port> element of your service:</p> <ul style="list-style-type: none"> • The name attribute • The binding value; the dropdown list displays the names of the bindings you have created for this document (see “Adding a binding element” on page 6) • The address type (None, SOAP, or HTTP) • The location (the URI by which your service will be available) <p>To add another port entry to the service, click Add. To remove an entry, select the entry and click Delete.</p>

4. Click OK.

A new service entry is added to your document.

```
<service name="StockQuoteService">  
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">  
    <soap:address location="http://example.com/stockquote"/>  
  </port>  
</service>
```

Validating a WSDL document

When a WSDL document is displayed in the Edit Pane, you can validate the document by clicking the **Validate** button (which looks like a check mark) in the toolbar. If the document is validated, you see this dialog:



Otherwise, you see a dialog giving information identifying the malformed statement(s) in the document.

CAUTION *You should carefully review your WSDL even if the document validation is successful. The W3C WSDL specification allows for extensibility elements throughout all levels of a WSDL document. So if you build the document without using the dialogs or do a lot of cut-and-paste from other sources, it is possible that the document will test as valid but not be what you want.*

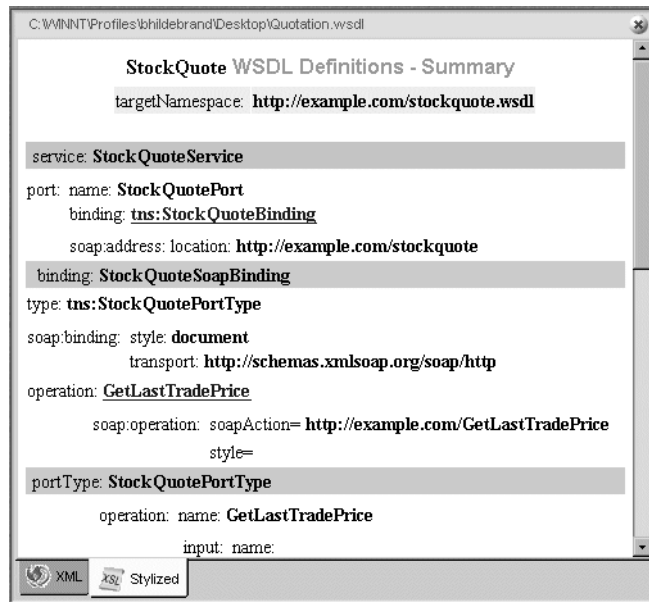
Displaying a stylized view

By default, WSDL documents are displayed in a color-coded text-edit view—the normal view for working on WSDL documents. You can also display a stylized view of WSDL documents, created by applying an XSL style sheet to your document. The WSDL Editor comes with two built-in style sheets: Summary and Detail.

➤ To display a stylized view of a WSDL document:

1. Open the WSDL document.
2. Click the **Stylized** tab at the bottom of the WSDL Edit Pane.

The view changes to stylized.

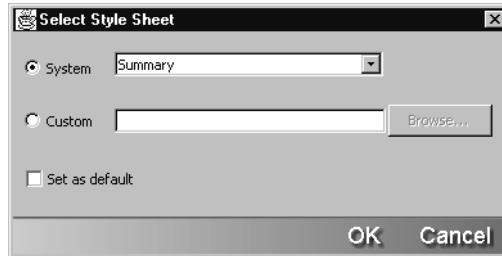


In this example, the Summary style sheet has been applied to the document.

➤ To choose a different style for the stylized view:

1. With the **Stylized** tab selected, right-click in the WSDL Edit Pane.
A popup menu displays.

2. Select an item from the Stylesheets submenu:
 - **Details** provides a detail-oriented plain-text view of the WSDL document (with no XML tags)
 - **Summary** provides a more concise view of WSDL contents
 - **Custom** opens a dialog that allows you to choose your own XSL style sheet for rendering a custom view, and/or setting a default style sheet



Choose one of the following:

Option	What to do
System	Select to use one of the built-in style sheets (Summary or Details) as the basis for the stylized view
Custom	Select to use the style sheet of your choice, then enter the path to the style sheet (or use the Browse button to open a standard file navigation dialog)

TIP You can optionally select the **Set as default** check box to apply the style sheet you've chosen as the default in stylized views. Your preference will persist across Workbench sessions.

Publishing to a registry

When you have created a WSDL document, you can publish it to a registry.

 For more information, see “Publishing to a registry” on page 293.

Generating Web Service files from WSDL

A WSDL document describes a Web Service. You can invoke the Web Service Wizard from the WSDL Editor to generate the Java classes needed to implement or consume that Web Service.

➤ **To generate Java classes:**

1. Make sure a Workbench project is open.
2. Open the WSDL document in text view.
3. Click the **Generate Java Class** button.



The Web Service Wizard is invoked.



For more information, see Chapter 5, “Web Service Wizard”.

9 Registry Manager

This chapter describes the registry browsing and publishing functionality provided in SilverStream eXtend Workbench. It contains the following topics:

- About UDDI
- About the Registry Manager
- Defining registry profiles
- Browsing registries
- Retrieving WSDL from the registry
- Publishing to a registry

About UDDI

The business registry standard covering Web Services is **UDDI** (Universal Description, Discovery, and Integration). UDDI is designed to give businesses a uniform way to describe their services, discover other companies' services, and understand the methods needed to conduct e-business in an automated or semiautomated way with remote partners. UDDI forms the basis for the registry management functionality.



To learn more about UDDI, see the complete standard at <http://www.uddi.org>.

About the Registry Manager

Workbench provides a Registry Manager, accessible through the Registries tab in the Navigation Pane, and a facility for defining registry profiles.

Workbench registry capabilities include:

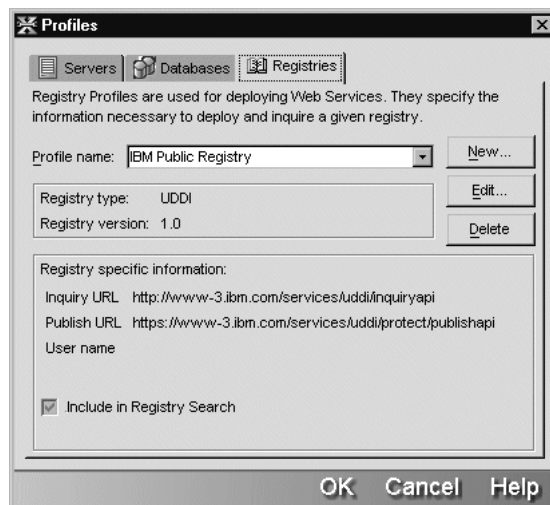
- Defining registry profiles
- Selecting registries to include in the search process
- Viewing business information on selected businesses in a given registry
- Viewing information on Web Services offered by a given business
- Searching for businesses or services within a registry or group of registries, optionally using extended query parameters
- Publishing new services to a registry

Defining registry profiles

Registries are specified by URL and can be local or Web-based. Before accessing a registry in Workbench, you define a **profile** for that registry. Workbench comes with some predefined registry profiles.

➤ **To define a registry profile:**

1. Select **Edit>Profiles** from the menu.
The Profiles dialog opens.
2. Select the **Registries** tab.



- If you are editing or deleting an existing profile, select it from the **Profile name** list box and click **Edit** or **Delete**. If you are creating a new profile, click **New**.

- Specify the following information:

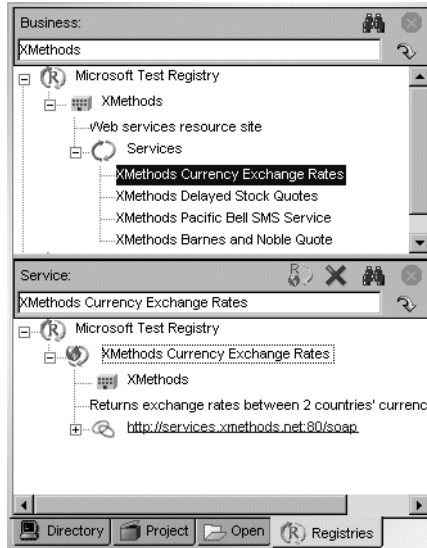
Option	Description
Profile name	Name of the profile
Registry type	Type of registry (default is UDDI)
Inquiry URL	The URL through which the registry can be queried
Publish URL	The URL through which new services can be published to the registry
User name and Credential	The information (if any) that the registry provider assigned to you for publishing access
Include in Registry Search	Specifies whether you want to include this registry in the default search set

- Click **OK**.

Once you have defined a registry profile, you can use the Registry Manager to browse the registry and you can publish services to the registry.

Browsing registries



The Registry Manager allows you to browse registries through the Registries tab in the Navigation Pane. There are two subpanes within the Registries tab: the Business Pane and the Service Pane.





Information displayed






The Registry Manager displays the following types of information.

Business Pane The business section of a registry might include these types of information:

Information	Icon	Description
Business name		Business name used in this registry
Description		Short phrase describing the business
Categories		Categories to which the business belongs Classification schemes come from at least three sources: NAICS codes for industry segments, UNSPSC for product and service classifications, and geographic information

Information	Icon	Description
Identifiers		Information about the business, such as a DUNS number
Services		A list of services offered by the business, such as Web Services callable via HTTP and other services such as sales and technical support contact information You can select a service name to display its details in the Service Pane


Service Pane A service entry in a registry might include these types of information:

Information	Icon	Description
Service name		The name of the service
Business name		The business offering the service
Description		A short phrase describing the service
Binding		The URL for invoking the service
tModel		Data describing the service A UDDI registry stores the data as a tModel, which is a set of name/value pairs; the tModel node may be followed by a description
Overview URL		The URL of a document describing how to use the tModel data For a Web Service, this is usually a WSDL document
Categories		Categories for the service The categorization has two parts: a name (for example, uddi-org:types) and a value (for example, wsdlSpec). The value wsdlSpec specifies that a WSDL document is available for the service. Other types of services can use other classification schemes.


Popup menus

Each pane in the Registry Manager has a popup menu.

Business Pane To view the popup menu for Business, place the cursor in an entry in the Business Pane and right-click. The following menu displays.

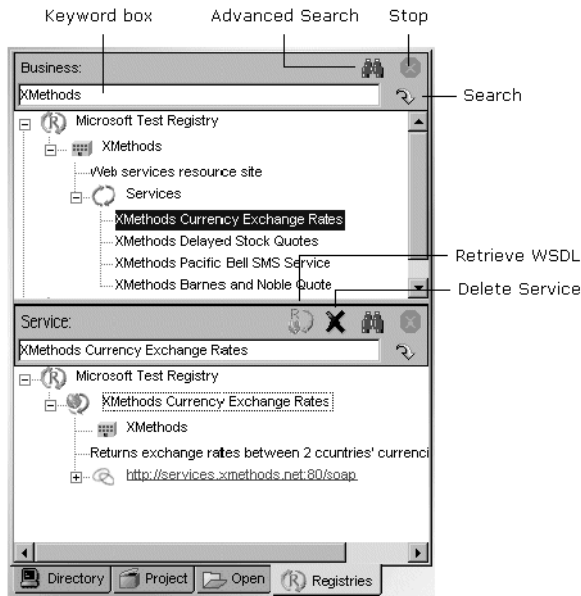
Menu item	Description
Copy Text	Allows you to copy text from the currently selected business tree node to another area or file
Clear Tree	Clears the pane of business information that you retrieved from your search
Advanced Search	Allows you to perform a sophisticated search by business  For more information, see “Searching by business” on page 7

Service Pane To view the popup menu for Service, place the cursor in an entry in the Service pane and right-click. The following menu displays.

Menu item	Description
Copy Text	Allows you to copy text from the currently selected service tree node to another area or file
Clear Tree	Clears the pane of service information that you retrieved from your search
Retrieve WSDL	Retrieves the WSDL for the selected service from the registry. You can also do this using the Retrieve WSDL button.
Delete Service	Deletes the selected service (if you have permission). Asks you to confirm before deleting the service.
Advanced Search	Allows you to perform a sophisticated search by service  For more information, see “Searching by service” on page 10

Action buttons

The following illustration shows the location of the various action buttons on the Business and Service panes.



Searching by business

You search by business in the Business Pane.

➤ To search businesses by name or keyword:

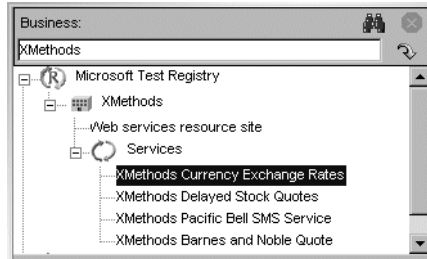
1. Enter a complete or partial business name or keyword in the text box below **Business**.

TIP You can also enter a group of business names separated by a vertical bar, which allows you to search for multiple groups of businesses. For example:
XMethods|IBM|Sun.

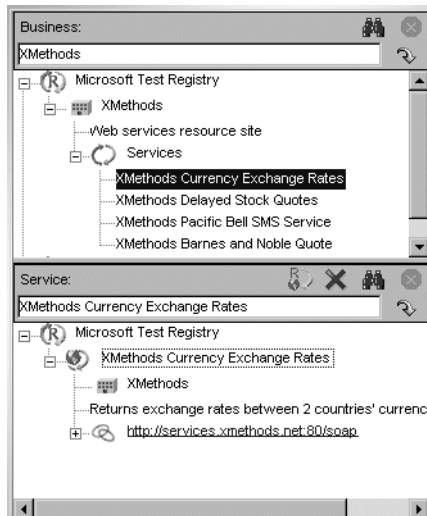
2. Click the **Search** button (shaped like a downturned arrow).

While the search is underway, the Stop button (normally grayed out) is red. The search can take several minutes. To interrupt the search, click the **Stop** button; partial search results will display in the Business Pane.

A list of matching businesses appears in tree-view form. Each top-level node in the tree is a registry, each child of a registry is a business name, and below each business is detail information consisting of descriptions, categories, and services.



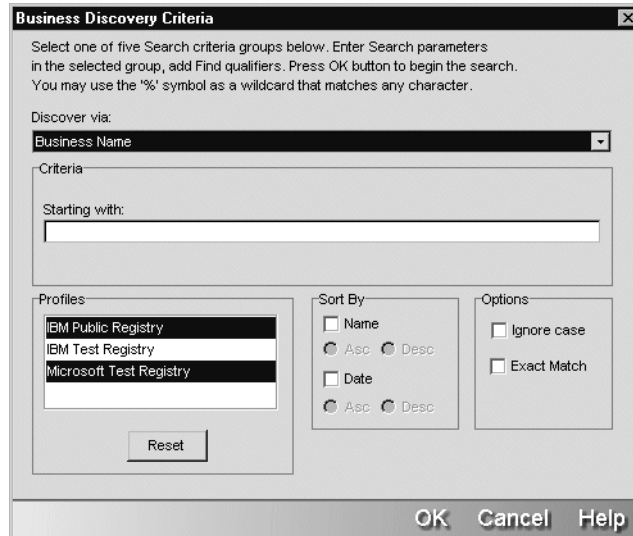
Clicking a service entry in the (upper) Business tree causes that service's detail information to appear in tree form in the (lower) Service Pane.



➤ **To set advanced business search criteria:**

1. Leave the keyword text box blank.
2. Click the **Advanced Search** button (shaped like binoculars).

The Business Discovery Criteria dialog displays.



3. Select one of the search-criteria options:

- **Business Name**—Enter a complete or partial business name or list of names separated by a vertical bar (|) in the **Starting with** text box.
- **Identifier**—Select one of the following: D-U-N-S or Thomas Register (catalog names) from the dropdown list. Enter a key from the catalog (partial or complete) in the **Starting with** text box; this entry can contain numeric values and dashes.
- **Locator**—Select one of the following from the dropdown list: NAICS (North American Industry Classification System), UNSPSC (United Nations Standard Products and Services Classification), or GEO (geographical).
Enter a key from the catalog (partial or complete) in the **Starting with** text box if you selected NAICS or UNSPSC; this entry can contain numeric values. Enter a country (region) abbreviation for GEO. If you selected NAICS or UNSPSC, you can click the ellipses and pick an item from a list of choices.
- **Service Type Name**—Select to search businesses associated with a particular tModel.
- **Discovery URL**—Enter an IP address or portion of an IP address for the URL in the **Starting with** text box.

4. Select search and sort options:
 - In **Sort By**, specify whether to sort by name or date, in ascending or descending order. The most common technique is to sort by name (alphabetically) in ascending order or by date (numerically) in descending order. Sorting by date works within groups of businesses with identical names.
 - In **Options**, select Ignore Case and/or Exact Match.
5. Under **Profiles**, select the registry or registries to search from the dropdown list. Those you specified in the Profiles dialog for automatic searching are already selected. To override the search list, select one or all of the registries in the list. To return to the original (default) registries, click **Reset**.
6. Click **OK**.

The search begins.

Searching by service

You search by service in the Service Pane.

➤ **To search services by name or keyword:**

1. Enter a complete or partial service name or keyword in the text box below **Service**.

TIP You can also enter a group of service names separated by a vertical bar, which allows you to search for multiple groups of services.
2. Click the **Search** button (shaped like a downturned arrow).

While the search is underway, the Stop button (normally grayed out) is red. The search can take several minutes. To interrupt the search, click the **Stop** button; partial search results display in the Service Pane.

A list of matching services appears in tree-view form. Each top-level node in the tree is the registry that was searched; each immediate child of a registry is a service name; and children of the service node(s) contain detail information consisting of the business name associated with the service, a description of the service, and bindings for the service.

Clicking a service node in the (lower) Service tree causes that business's detail information to appear in tree form in the (upper) Business Pane.

➤ **To set advanced service search criteria:**

1. Leave the keyword text box blank.
2. Click the **Advanced Search** button (shaped like binoculars).

The Service Discovery Criteria dialog box displays.

3. Select one of the search-criteria options:
 - **Service Name**—Enter a complete or partial service name in the **Starting with** text box.
 - **Locator**—Select one of the following: NAICS (North American Industry Classification System), UNSPSC (United Nations Standard Products and Services Classification), UDDITYPE, or GEO (geographical) from the dropdown list.
Enter a key from the catalog (partial or complete) in the **Starting with** text box if you selected NAICS, UNSPSC, or UDDITYPE; this entry can contain numeric values. Enter a country (region) abbreviation for GEO. If you selected NAICS, UNSPSC, or UDDITYPE, you can click the ellipses and pick an item from a list of choices.
 - **Service Type Name**—Allows the search of services associated with a particular tModel.

4. Select search and sort options:
 - In **Sort By**, specify to sort by name or date, in ascending or descending order. The most common technique is to sort by name (alphabetically) in ascending order or by date (numerically) in descending order. Sorting by date works within groups of services with identical names.
 - In **Options**, select Ignore Case and/or Exact Match.
5. Under **Profiles**, select the registry or registries to search from the dropdown list. Those you specified in the Profiles dialog for automatic searching are already selected. To override the search list, select one or all of the registries in the list. To return to the original (default) registries, click **Reset**.
6. Click **OK**.

The search begins.

A tree of matching services is built in the Service Pane. Clicking a service node in the (lower) Service tree causes that business's detail information to appear in tree form in the (upper) Business Pane.

Retrieving WSDL from the registry

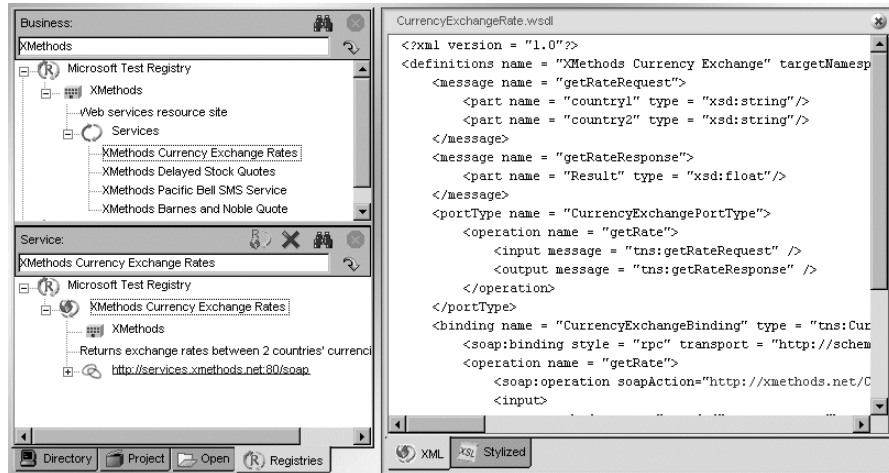
After you have found the service you searched for, you can retrieve the WSDL definition for this service from the registry. For this you use the Service Pane.


➤ To retrieve a WSDL definition from the registry:

1. Highlight the service node.
2. Click the **Retrieve WSDL** button in the Service Pane.



If a definition for the service exists, the WSDL Editor displays the WSDL information.



 For information about the WSDL Editor, including different ways to view the WSDL, see Chapter 8, “WSDL Editor”.

Publishing to a registry

When you have created a WSDL document in the WSDL Editor, you can publish it to a registry.

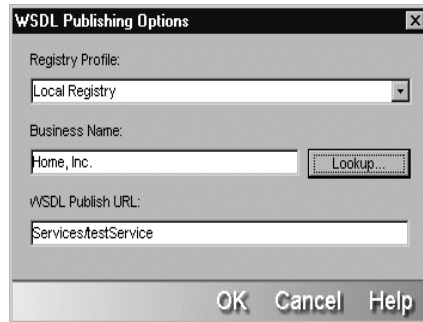
 For information about the WSDL Editor, see Chapter 8, “WSDL Editor”.

➤ To publish to a registry:

1. Open the WSDL document in the WSDL Editor.
2. Click the **Publish to Registry** button on the toolbar.



The WSDL Publishing Options dialog displays.



3. Specify these options:
 - **Registry Profile**—Select the registry you want to publish to.
 - **Business Name**—Specify the business to associate the service with. Click the **Lookup** button to look up businesses in the registry.
 - **WSDL Publish URL**—Specify the URL to which the service will be published.
4. Click **OK**.

If your service was successfully published, you see a confirmation dialog.

If the service was not published, you see a dialog describing the error.

10 Deployment Descriptor Editor

The Deployment Descriptor Editor provides a quick and easy way to construct and populate J2EE-compatible deployment descriptors. This chapter describes the Deployment Descriptor Editor and includes these topics:

- About deployment descriptors
- About the Deployment Descriptor Editor
- Using the Deployment Descriptor Editor

About deployment descriptors

A **deployment descriptor** is an XML document that provides information about the components of a J2EE module (such as a WAR or an EJB JAR) or application (such as an EAR). The deployment descriptor provides data that is required for both of the following:

- Application assembly—to describe how a component is or should be used
- Deployment—to define deployment needs such as roles and resource references

Sun has defined the contents and structure for a deployment descriptor for each J2EE component. For more information, see *J2EE Deployment Descriptor DTDs* in the online *Reference*.

How deployment descriptors are created Workbench automatically constructs and adds a J2EE-compatible deployment descriptor file to your project in the appropriate location, as follows:

J2EE component	Deployment descriptor file	Directory location
Application client	application-client.xml	/META-INF
EAR	application.xml	
EJB JAR	ejb-jar.xml	
RAR	ra.xml	
WAR	web.xml	/WEB-INF

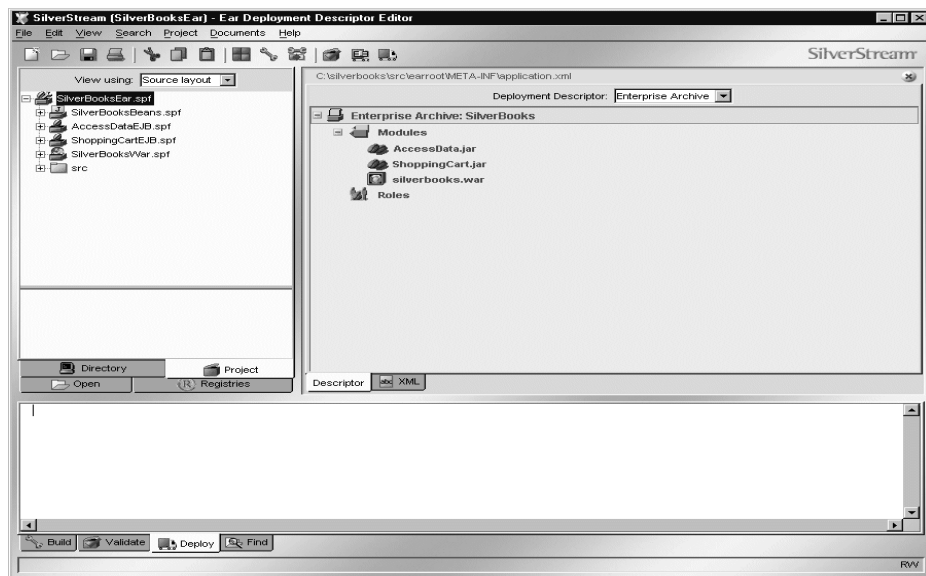
As you add J2EE components to a project, Workbench adds the corresponding elements to the deployment descriptor when it has enough information to do so.

About the Deployment Descriptor Editor

The Deployment Descriptor Editor allows you to fine-tune the deployment descriptor by modifying or completing entries that Workbench is unable to complete automatically.

The Deployment Descriptor Editor displays the deployment descriptor elements as expandable nodes. The nodes correspond to elements of the deployment descriptor DTD. All possible deployment descriptor entries are represented graphically, so you can use the interface to help you add the appropriate entries without having to memorize the DTD.

Here is a sample Deployment Descriptor Editor for an EJB project:



TIP You can view or edit the deployment descriptor in raw XML by choosing the **XML** tab. The Deployment Descriptor Editor opens in the mode (raw XML or tree view) in use when it was last saved.

Nodes displayed in bold (like Environment and Persistent Fields) allow child nodes to be added or removed. You can add or remove these nodes by right-clicking and selecting from the popup menu.

Many of the nodes require additional information, and you can provide this information by completing a property sheet. To display the Property Inspector for a node, highlight the node, right-click, and select **Properties**.

To save the changes to the deployment descriptor file in the archive, select **File>Save** (or click the Save icon).

Using the Deployment Descriptor Editor

You can use the Deployment Descriptor Editor either to fine-tune the default deployment descriptor created by Workbench or to create a new deployment descriptor.

➤ To create a deployment descriptor:

1. Open the project for which you want to create the new deployment descriptor.
2. Select **File>New**.
3. Select the **J2EE** tab.
4. Select **Deployment Descriptor** and click **OK**.

Workbench constructs the deployment descriptor shell based on the contents of the project and displays the shell in the Edit Pane.

➤ To associate a deployment descriptor with a project:

NOTE If you created a deployment descriptor outside the Workbench environment, you can still use it with a Workbench project by following these steps.

1. Open the Workbench project that you want to associate the deployment descriptor with.
2. Choose the **Directory Pane**.
3. Double-click the deployment descriptor you want.

You are prompted to associate the descriptor with the current project, a different project (which you can choose), or to edit the deployment descriptor in XML mode.

4. Choose the option to associate the descriptor with the current project, then choose **OK**.
Workbench opens the deployment descriptor in the Deployment Plan Editor.
5. Save the deployment descriptor to complete the association.

➤ **To modify a deployment descriptor:**

1. Open the project whose deployment descriptor you want to modify.
2. Highlight the project file (the SPF), right-click, and select **Open Deployment Descriptor** from the popup menu.
You are prompted for your Build preferences.
Once you specify the Build preferences, the Deployment Descriptor Editor opens the file ready for editing.

➤ **To add a deployment descriptor element:**

1. Open the deployment descriptor for editing.
2. Highlight the descriptor element, right-click, and choose **Add** from the popup menu.
The editor adds a new element with the title UntitledXXX.
3. Highlight the new element, right-click, and choose **Properties** from the popup menu to launch the Property Inspector so you can define any necessary values.

➤ **To remove a deployment descriptor element:**

1. Open the deployment descriptor for editing.
2. Make sure the **Descriptor** tab (not XML tab) is selected.
3. Highlight the descriptor element you want to remove, right-click, and select **Delete** from the popup menu.

NOTE If Delete is not available as a menu option, the element is not removable.

Validating a deployment descriptor

The Deployment Descriptor Editor automatically checks your work as follows:

When you	The Deployment Descriptor Editor
Switch mode (GUI to XML and vice versa)	Checks the syntax of the deployment descriptor
Save the deployment descriptor	Validates the current deployment descriptor against the DTD

But you can force validation anytime.

➤ **To force validation of a deployment descriptor:**

- Choose **Validate Archive** from the Project menu.
This validates both the deployment descriptor and the archive.

11 Deployment Plan Editor

The Deployment Plan Editor provides a quick and easy way to construct and populate deployment plans needed for deploying J2EE modules and applications to a SilverStream eXtend Application Server. This chapter describes how to use the Deployment Plan Editor and includes these topics:

- About Deployment Plans
- Using the Deployment Plan Editor

About Deployment Plans

A SilverStream **deployment plan** is an XML document that describes how a J2EE module (such as a WAR or an EJB JAR) or application (such as an EAR) should run in the SilverStream eXtend Application Server environment.

A SilverStream deployment plan allows you to map declarative data from the deployment descriptor to the appropriate resource in the target server environment. For example, you can map resource references to actual data sources or map roles to actual users or groups.

SilverStream has defined the contents and structure for a deployment plan for each J2EE component. For more information, see SilverStream Deployment Plan DTDs in the online *Reference*.

NOTE Other application servers will require different types of information (possibly in different formats) for deployment. To deploy J2EE components on another J2EE application server, see the application server vendor's documentation.

Using the Deployment Plan Editor

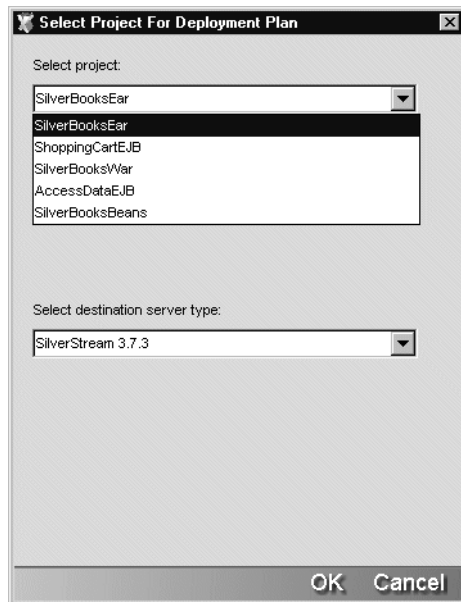
This section describes how to use the deployment plan editor to perform these tasks:

- Create a deployment plan
- Modify a plan
- Associate a deployment plan with a project
- Validate a deployment plan

➤ **To create a deployment plan:**

1. Make sure you have built the archive and added appropriate items to the deployment descriptor.
2. Open the project for which you want to create the deployment plan.
3. Choose **File>New**.
4. Choose the **J2EE** tab.
5. Choose **SilverStream Deployment Plan** and click **OK**.

The Select Project For Deployment Plan dialog displays.

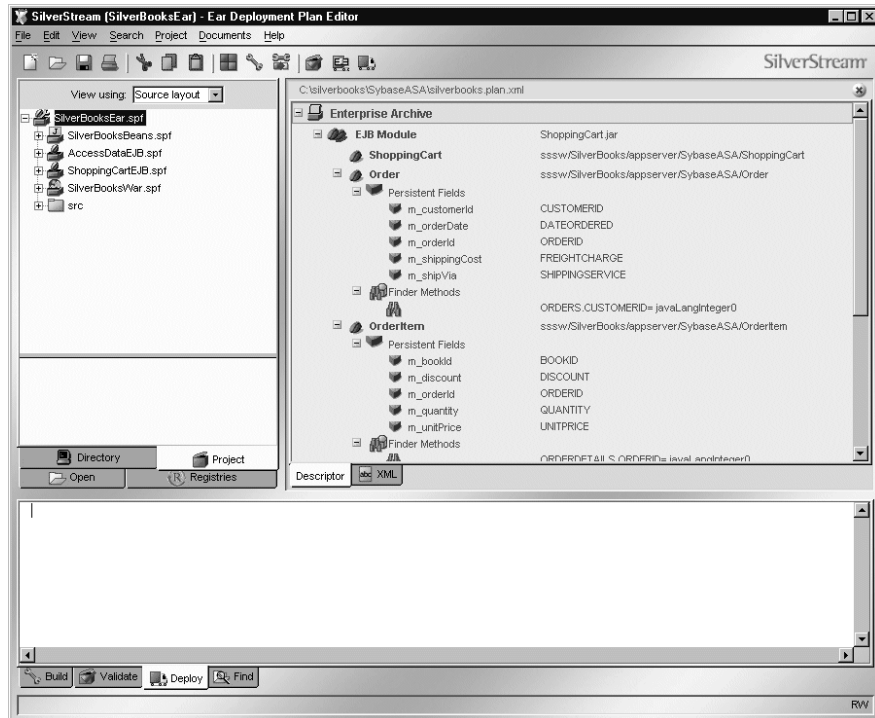


6. Choose a project from the **Select project** dropdown.
NOTE When the project is an EAR, you see multiple files in the dropdown.
7. Choose the destination server type from the server type dropdown and click **OK**.
NOTE The server type listed when the dialog opens is the one specified as the default in Deployment preferences (**Edit>Preferences**). For more information, see “Deployment preferences” on page 18.

The Deployment Plan Editor constructs a deployment plan based on the project type. The editor uses the project’s compiled code and the deployment descriptor to determine the deployment plan entries for a specified project. The deployment plan elements are displayed in a tree structure.

Whenever you change the deployment descriptor, Workbench updates the deployment plan to match the changes when you next edit the deployment plan.

Here is a sample deployment plan for an EAR. It contains three EJB modules and a WAR:



TIP You can view or edit the deployment plan in raw XML by choosing the **XML** tab. The Deployment Plan Editor opens in the mode (raw XML or tree view) in use when it was last saved.

8. Choose **File>Save** (or click the Save icon).

If there are other SilverStream deployment plans associated with this project, you will be asked whether you want to make the new deployment plan the current one.

9. Click **Yes** to make it the current deployment plan.

➤ To modify an existing deployment plan:

1. Open the project whose deployment plan you want to modify.
2. Highlight the project file (the SPF), right-click, and select **Open Deployment Plan** from the popup menu.

3. If your project has more than one deployment plan, choose the deployment plan from the dropdown and click **OK**.

Workbench displays the deployment plan in the Edit Pane.

4. Highlight a deployment plan element, then right-click and select **Properties** from the popup menu.

Use the Property Inspector to modify values for different elements. In some cases, you can double-click an element to open a dialog that lets you enter data more quickly than through the Property Inspector.

➤ To associate a deployment plan with a project:

NOTE If you created a deployment plan outside the Workbench environment, you can still use it with a Workbench project by following these steps.

1. Open the Workbench project you want to associate the deployment plan with.
2. Choose the **Directory Pane**.
3. Double-click the deployment plan you want.
You are prompted to associate the plan with the current project, a different project (which you can choose), or to edit the deployment plan in XML mode.
4. Choose the option to associate the plan with the current project, then choose **OK**.
Workbench opens the deployment plan in the Deployment Plan Editor.
5. Save the deployment plan to complete the association.
6. Choose **Yes** when prompted to mark the deployment plan as current.

Validating a deployment plan

The Deployment Plan Editor automatically checks your work as follows:

When you	The Deployment Plan Editor
Switch mode (choose the Descriptor or the XML tab)	Checks the syntax of the deployment plan
Save the deployment plan	Validates the deployment plan against the DTD

But you can force validation anytime.

➤ **To force validation of a deployment plan:**

- Choose **Validate Archive** from the Project menu.
This validates both the deployment plan and the archive.

12 Debugger

The SilverStream Debugger lets you find runtime errors in your Java applications by controlling and monitoring the execution of Java code. You can debug server-side objects (such as J2EE applications) and client-side objects, either on a local host machine or remotely on distributed machines.

NOTE By default, when you launch a debugger from within Workbench, the SilverStream Debugger is launched. If you want to use a different debugger, you can specify that debugger so it is launched from within Workbench. For more information, see “Specifying a debugger” on page 43.

This chapter describes the following:

- Concepts you need to know
- About the Debugger
- Debugging server applications
- Debugging client applications
- Managing program execution
- Analyzing the behavior of the application
- Debugger keyboard shortcuts

Concepts you need to know

You should review the following concepts before you use the Debugger.

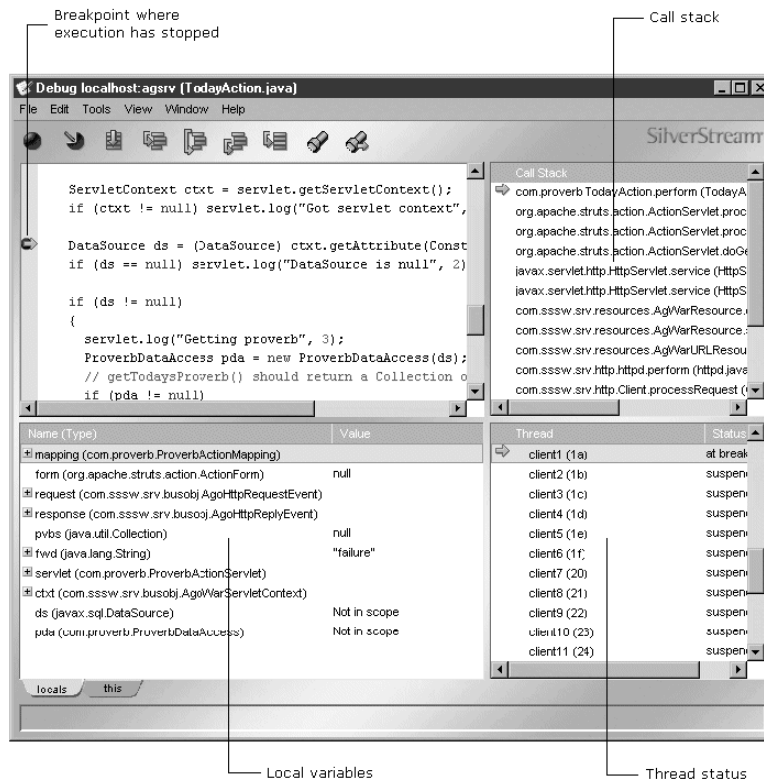
Concept	Description
Breakpoint	You can set a breakpoint on an executable line in your code where you want execution to stop temporarily. When you set a breakpoint in code, the program stops running before executing the line containing the breakpoint, then turns control over to the Debugger.
Deadlock	A deadlock is a condition that occurs when two processes are each waiting for the other to complete before proceeding. Both processes hang.

Concept	Description
Instance variable	An instance variable is a field declared within a class declaration without using the keyword static .
Local variable	A local variable is declared inside a particular method. Only code that is contained in the method can access a local variable.
Monitor	In Java, a monitor is an exclusive lock on an object. Locks are used in thread synchronization. When a method is programmed to be synchronized, it cannot be run in multiple threads concurrently. When a synchronized method is entered, it acquires a monitor on the current object (the object whose method was called). The monitor prevents other synchronized methods in the object from executing. When the synchronized method returns, its monitor is released, allowing other synchronized methods in the same object to run.
Thread	A thread is a single execution stream in a program. Java is a multithreaded language, which means that you can have many execution streams operating at one time. In Java, threads are represented by the Thread object.

About the Debugger

The SilverStream Debugger provides several capabilities for diagnosing problems with your Java applications.

Here is a sample Debugger window, showing many of its features:



Debugging server objects and client objects You can debug the following objects:

Server objects	Objects in deployed J2EE applications: <ul style="list-style-type: none">• Servlets (including compiled JavaServer Pages)• Enterprise JavaBeans• Other Java code in deployed J2EE archives Other Java applications executing on a server
Client objects	Client Java applications, including J2EE application clients

Local and remote debugging With the SilverStream Debugger you can troubleshoot Java applications running on a local host or remotely in a distributed network. You can configure a process to be debugged either on the local machine or on a remote machine, but not for both in the same session.

There are many situations that require remote debugging. For example:

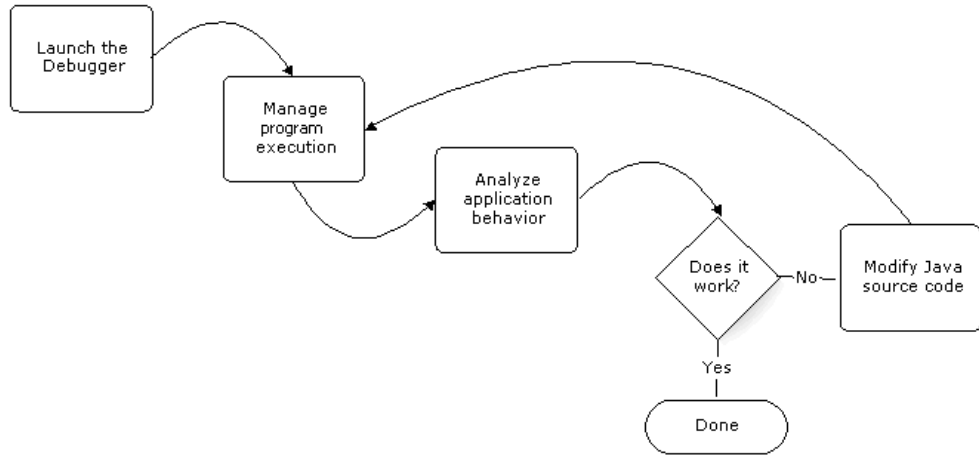
- The Java application runs error-free on some machines but not others in a network, requiring you to debug the application while it runs on the problematic remote system.
- The application runs in a network where the server is in a remote location.
- The application must be tested on a remote machine that does not have the resources to run the Debugger locally.

Controlling program execution The Debugger provides methods for exercising your code by:

- Setting and managing breakpoints in Java application source code
- Stepping in, out, and over source code
- Running applications to a location in source code marked by the cursor
- Running applications to the next breakpoint
- Suspending and resuming threads

Monitoring program status To help you isolate problems in source code, the Debugger provides viewers for examining variables, the call stack, and thread status at any point in program execution.

Typical workflow for debugging Although each debugging session poses its own unique challenges, you typically follow this workflow when troubleshooting applications:



Debugging server and client applications The techniques you use to debug applications running on a server as opposed to client applications are different. The next two sections describe the techniques:

- Debugging server applications
- Debugging client applications

Debugging server applications

If you are debugging objects running on a server (such as J2EE applications), you should start the application on your server and then attach the Debugger to that application, rather than trying to start the application from the Debugger. You can debug applications running on any J2EE application server supported by Workbench.

Starting the server

In some cases, you must start your application server in debug mode before you can debug server objects. This section provides instructions for starting the SilverStream eXtend Application Server and BEA WebLogic for debugging.



For further details about starting these and other application servers in debug mode, consult the documentation for the server.

Starting the SilverStream server for debugging

➤ To start the SilverStream eXtend Application Server for debugging:

- Start the server with one of the following options:
 - **+debug** starts the server for local debugging, using the default debug address **agsrv**. You can specify these variations for local debugging:

Variation	Description
+debug:suspend=y	Suspends the JVM at startup. Use the Continue command in the Debugger to resume execution. This option is helpful for debugging initialization code that would normally get executed before the Debugger is attached.
+debug:suspend=n	Does not suspend the JVM at startup. This is the default.
+debug:address=<i>debug_address</i>	Lets you specify the debug address to use.

- **+debugremote** starts the server for remote debugging, using the default debug port **9901**.

- **+debug:port=port_num** starts the server for remote debugging, using *port_num* instead of the default debug port 9901. Use this option if port 9901 is not available.

Examples This command line launches the SilverStream server for local debugging:

```
ServerInstallDir\bin\SilverServer +debug
```

This command line launches the SilverStream server for remote debugging:

```
ServerInstallDir\bin\SilverServer +debugremote
```

Starting the WebLogic server for debugging

➤ To start WebLogic for debugging:

1. Modify your **startWebLogic.cmd** file by adding the options in bold to the command line for starting the server:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -Xdebug -Xnoagent  
-Xrunjwp:transport=dt_socket,server=y,suspend=n -Djava.compiler=NONE  
-classpath ...
```

2. Start the server.

When starting, the server will display the debug address:

```
Listening for transport dt_socket at address: address
```

You will use this value when launching the Debugger, as described next.

Launching the Debugger

You can launch the Debugger from within Workbench or from the command line.

➤ To launch the Debugger from Workbench:

1. Select **Edit>Launch Debugger**.



2. Select **Attach to running process**.
3. If debugging an application on a local server, select **Shared Memory Debugging** and provide the debug address.

If debugging an application on a remote server, select **Remote Socket Debugging** and specify the machine name and debug port.

NOTE If debugging an application running on WebLogic, even on a local WebLogic server, select **Remote Socket Debugging** and specify the machine name (**localhost** if local) and the address (port) the server reported when starting.

4. Open files you want to debug in the Debugger, set breakpoints, then run your application.

➤ To launch the Debugger from the command line:

1. Make current the *WorkbenchInstallDir\bin* directory.
2. If debugging an application on a local server, enter:

```
SilverDebugger -attach localhost debug_address
```

If debugging an application on a remote server (or a local WebLogic server), enter:

```
SilverDebugger -attach machine_name:debug_port
```

3. Open files you want to debug in the Debugger, set breakpoints, then run your application.

Now you are ready to manage program execution and analyze the behavior of your application.

A sample debugging session

In the following scenario, you'll see how to debug ProverbFinal, the J2EE application that is built in the Workbench's Web application tutorial. The scenario shows debugging the application on a local SilverStream server. (This application is provided as a Workbench project. For details, see *Tutorials* in the Workbench help.)

1. Start the local server in debug mode.

```
ServerInstallDir\bin\SilverServer +debug
```

2. Open the ProverbFinal project in Workbench.

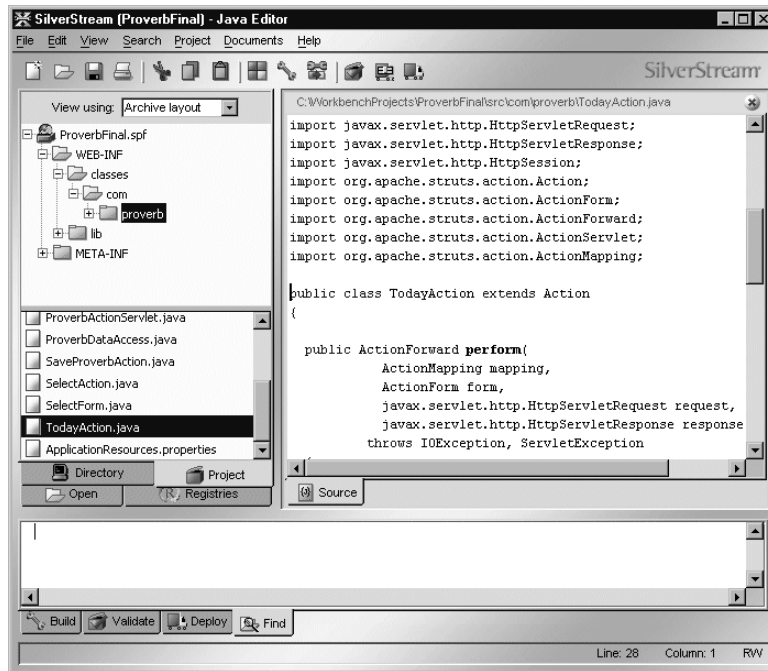
The project is in *WorkbenchInstallDir*\docs\tutorial\ProverbFinal. It is a completed application.

3. Deploy the application to the server.

In this scenario, the application was deployed to ProverbsCloud, the Cloudscape database provided with the tutorial.

4. Open the file to be debugged.

If you have a file open when you launch the Debugger from Workbench, that file opens in the Debugger automatically. You can also open other files in the Debugger to debug them.



Here, TodayAction.java was opened. This code is executed when someone asks to see today's proverb.

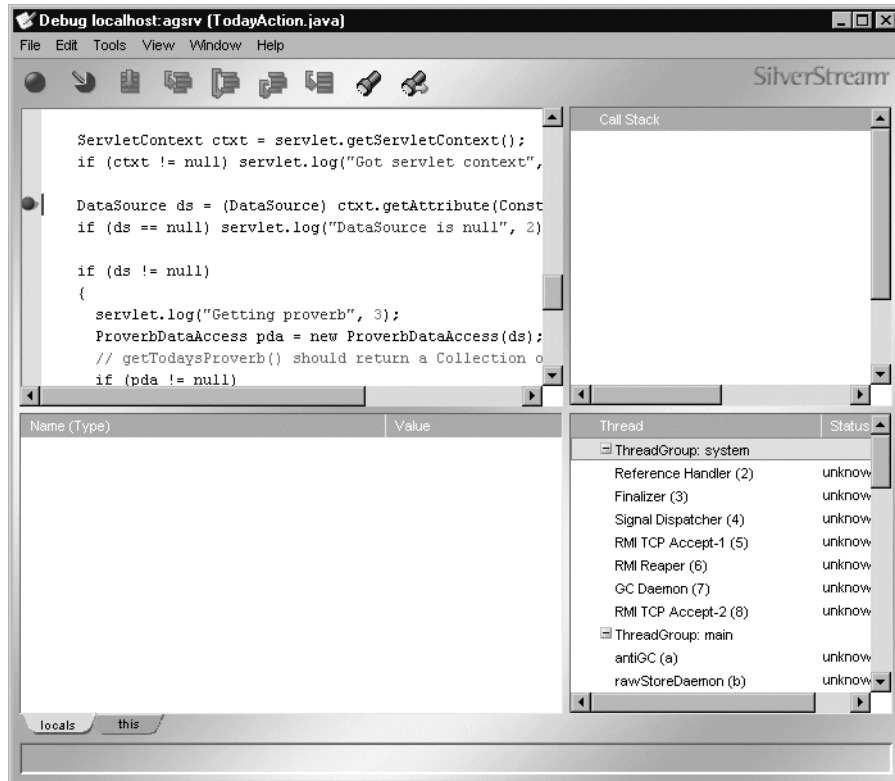
5. Select **Edit>Launch Debugger** to start the Debugger.
6. Specify the following:



Because the SilverStream server was started using the default debug address, **agsrv** was specified as the debug address.

The Debugger opens and displays the file.

7. Set a breakpoint on a line of code by placing the cursor on the line and clicking the **Toggle Breakpoint** icon in the Debugger toolbar:

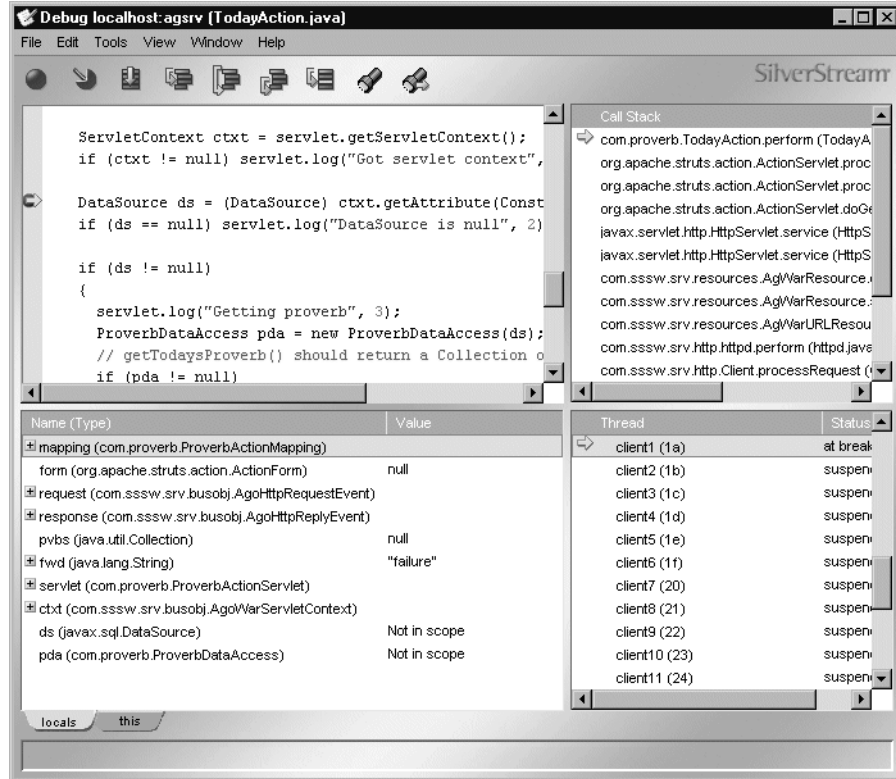


8. Run the application. To do that, open a browser and specify this URL:

`http://localhost/ProverbsCloud/ProverbFinal/index.jsp`



9. Click **Today's Proverb**. This action invokes code in TodayAction.java. Execution stops (you will see that the browser has not completed processing the page) and the Debugger window is updated.



You can see that the execution arrow is at the breakpoint.

10. At this point, step through the code, looking at variable values, the call stack, and so on, as described in the rest of this chapter.
11. When finished debugging, continue execution by clicking the **Continue** icon:



The page completes execution.

Debugging J2EE applications

Using the techniques shown in the preceding sample debugging session, you can debug your J2EE applications. You debug EJBs and servlets exactly as shown above: open the source code for the EJB or servlet in the Debugger, set one or more breakpoints, then run your application.

Debugging JSP pages

To debug a JSP page, which is translated and compiled into a servlet for execution, you need to locate the Java source file that the server created from the JSP page, open the source file in the Debugger, and set your breakpoints. Different J2EE servers will locate their source files differently. Here is information about the SilverStream eXtend Application Server and WebLogic:

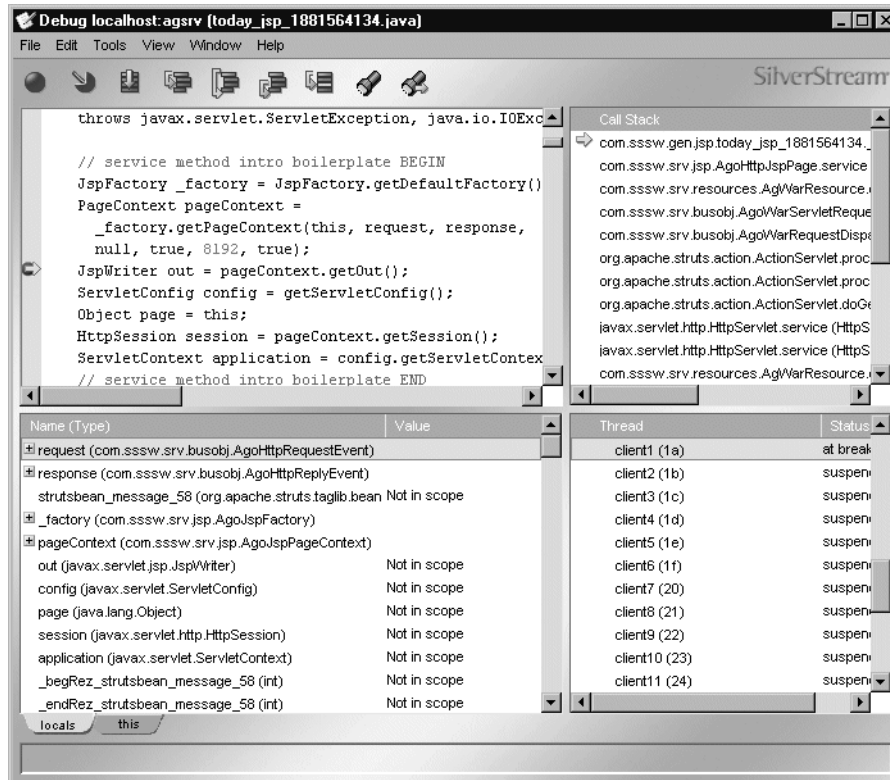
Server	Description
SilverStream eXtend Application Server	<p>The SilverStream server locates its generated source files in its compile cache.</p> <p>To find the Java source files the SilverStream server generated from deployed JSP pages, look in this directory:</p> <pre>ServerInstallDir/compilecache/server/database/temp/sources/application/com/sssw/gen/jsp</pre> <p>For example, to find the source file corresponding to today.jsp, shown above, look in:</p> <pre>ServerInstallDir/compilecache/localhost/ProverbsCloud/temp/sources/ProverbFinal/com/sssw/gen/jsp</pre> <p>You will see a set of .java files corresponding to the JSP pages that the server translated into servlets. Files will be named <i>name_jsp_nnnnnnnnnn.java</i>.</p>
WebLogic	<p>By default, WebLogic does not save the Java source files generated from JSP pages. To instruct WebLogic to keep the generated Java source code for your JSP pages, specify the value true for the keepgenerated parameter in the jsp-descriptor element in weblogic.xml.</p>



For information about other servers, consult their documentation.

To debug a JSP page, open its corresponding .java file in the Debugger, set a breakpoint, and run the JSP page. Execution will stop, and you can examine your JSP page.

Here, execution has stopped in the Java source file generated from today.jsp:



Debugging client applications

In addition to debugging J2EE applications and other applications running on application servers, you can also use the Debugger to debug client Java applications.

When debugging client applications, you can either invoke the Debugger to start the application or attach the Debugger to a running application.

Invoking the Debugger to start the application

You can start an application from within Workbench or from the command line.

➤ To start a client application from Workbench:

1. In Workbench, open the project that defines the application.
Make sure the project's classpath is set up correctly.
2. Open the Java source file you want to debug.
3. Select **Edit>Launch Debugger**.



4. Select **Launch new process**.
5. Specify the class to execute and specify any needed arguments.
6. Click **OK**.

➤ To start a client application from the command line:

1. Make current the *WorkbenchInstallDir\bin* directory.
2. Enter the following at the command line:

```
SilverDebugger [options] class class_arguments
```

These are the SilverDebugger arguments for launching an application:

Argument	Description
options	<p>-sourcepath <i>directory_list</i></p> <p>The list of the directories (separated by semicolons) the Debugger searches for source files</p> <p>NOTE These directories must be source tree roots</p>
	<p>-sourcefile <i>filename</i></p> <p>The fully qualified name of the Java source file to display in the Debugger</p>
	<p>-?</p> <p>Display usage information for the SilverDebugger command</p>
	<p>-classpath <i>directory_list</i></p> <p>The list of the directories (separated by semicolons) the Debugger searches for Java classes used in the application</p>
	<p>-Dname=value</p> <p>A system property setting for the application environment</p>
	<p>-Xoption</p> <p>A JVM option for the application environment</p>
class	The name of the class to debug
class_arguments	Arguments to pass to the main() method of class
<p>NOTE If your source and class files reside on remote network machines, specify paths and file names using Universal Naming Convention (UNC) format:</p> <p style="padding-left: 40px;"><code>\\server-name\shared-resource-pathname</code></p>	

For example, to debug the demo Notepad application that comes with the JDK, enter this command (where *InstallDir* is the installation directory for the demo application, such as `c:\jdk1.3\demo\jfc\Notepad`):

```
SilverDebugger -sourcepath InstallDir\src
               -sourcefile InstallDir\src\Notepad.java
               -classpath InstallDir\Notepad.jar
               Notepad
```

The Debugger opens on your desktop displaying the Java source file specified.

3. Set one or more breakpoints in your code or in exceptions.
4. Select **Tools>Continue** from the menu or click the **Continue** icon in the toolbar of the Debugger window:



The application opens on your desktop and execution stops at the first breakpoint encountered.

Now you are ready to manage program execution and analyze the behavior of your application.

Attaching to a running application

You can also start the application, then attach the Debugger to it. To do this, you must start the application specifically for debugging.

➤ To attach the Debugger to a running Java application:

1. Launch the Java application either on the local host or on a remote machine:

To launch the application on the local host:

- Type this command on the command line:

```
java -Xdebug -Xnoagent
     -Xrunjdpw:transport=dt_shmem,server=y,suspend=n
     -classpath classpath classname
```

The JVM returns a **debug address**. For example:

```
Command Prompt - java -Xdebug -Xnoagent -Xrunjdpw:transport=dt_shmem,server=y,suspend=n -clas...
C:\jdk1.3.1_01\demo\jfc\Notepad>java -Xdebug -Xnoagent -Xrunjdpw:transport=dt_shmem,server=y,suspend=n -classpath . -jar Notepad.jar
Listening for transport dt_shmem at address: javadebug
```


In this example, the JVM returns **javadebug** as the debug address. The option **transport=dt_shmem** specifies shared-memory transport, which is required for local debugging.

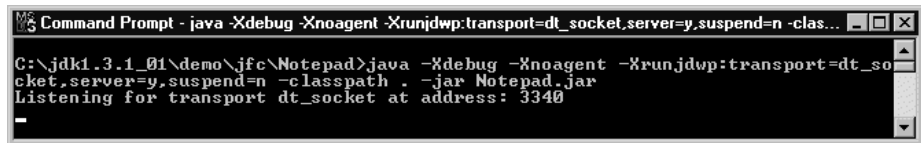
- Note the debug address; you must pass it to the Debugger in Step 2.

To launch the application on a remote network machine:

- Type this command in a command line:

```
java -Xdebug -Xnoagent
-Xrunjdp:transport=dt_socket,server=y,suspend=n
-classpath classpath classname
```

The JVM returns a **debug port number**. For example:



In this example, the JVM returns **3340** as the debug port. The option **transport=dt_socket** specifies socket-based transport, which is required for remote debugging.

- Note the debug port number; you must pass it to the Debugger in Step 2.
2. Invoke the Debugger either from within Workbench or from the command line.

To invoke the Debugger from within Workbench:

1. Select **Edit>Launch Debugger**.
2. Select **Attach to running process**.
3. If debugging locally, select **Shared Memory Debugging** and specify the **debug address** returned by the JVM. If debugging remotely, select **Remote Socket Debugging** and specify the **debug port** returned by the JVM.



To invoke the Debugger from the command line:

- Use the appropriate **SilverDebugger** command:

For	Type this command
Local debugging	<pre>SilverDebugger -attach localhost <i>debug_address</i></pre> <p>NOTE <i>debug_address</i> is the address returned by the JVM when you launched your Java application for local debugging.</p> <p>For example:</p> <pre>SilverDebugger -attach localhost javadebug</pre>
Remote debugging	<pre>SilverDebugger -attach <i>machine_name:debug_port</i></pre> <p>NOTE <i>debug_port</i> is the port number returned by the JVM when you launched your application for remote debugging.</p> <p>For example:</p> <pre>SilverDebugger -attach mymachine:3340</pre>

The Debugger opens on your desktop.

3. Open a source file for debugging using **File>Open**.

Now you are ready to manage program execution and analyze the behavior of your application.

Debugging against SilverJRunner or SilverJ2EEClient This is just like debugging against a SilverStream server. You launch SilverJRunner or SilverJ2EEClient with an appropriate debug startup option (described in “Starting the server” on page 6), then attach the Debugger to the resulting debug address (default is **agjrn**) or debug port (default is **9901**).

Managing program execution

Once you have started the Debugger, there are several ways to manage program execution to allow you to isolate and diagnose problems in the source code:

- Using breakpoints
- Continuing execution
- Stepping through the code

Using breakpoints

You use breakpoints to specify lines in your code where you want to stop execution and give control to the Debugger. When the code stops executing and the Debugger takes control, you can execute Debugger commands and view the call stack, thread status, and variable values.

You can set breakpoints only on individual lines in your code. If a line contains multiple statements, you can set a breakpoint only on the first statement in the line. To set breakpoints on the subsequent statements, you need to break up the line so that each statement appears on its own line.

NOTE Breakpoints are stored by server name and class name. As a result, like named objects on the same server share breakpoints.

Setting and removing breakpoints in code


There are several ways to set and remove breakpoints in code, either for local classes or classes loaded from external sources. If the Debugger cannot find the source code, it prompts you for a path, as described in “When the Debugger cannot locate source code” on page 26.

This section describes the procedures for setting and removing breakpoints in source code.

➤ To set breakpoints in code using Toggle Breakpoint:

1. Put the cursor at the line in the source code where you want to set a breakpoint.
2. Either choose **Tools>Toggle Breakpoint** from the menu or select the **Toggle Breakpoint** icon in the Debugger toolbar:



The breakpoint indicator  appears in the left margin beside the line you specified.

➤ To set breakpoints in code using the Edit menu:

1. Select **Edit>Show Line Numbers** from the menu.
This makes it easier to specify locations for breakpoints.
2. Choose **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
3. Select the **Code** tab.
The dialog lists all breakpoints that have been set.
4. Click **Add**.

5. Do one of the following:

To insert a breakpoint in	Do this
Your local Java class	<ol style="list-style-type: none"> In the Specify Breakpoint field, enter the location where you want to add a breakpoint, in this form: <code><fully qualified class name>:<line number></code> Example: <code>com.myapp.gui.CheckBoxes:99</code> Click OK.
Loaded classes	<ol style="list-style-type: none"> Click Browse. Navigate to the method where you want to insert a breakpoint and click OK. The breakpoint specification appears in the Specify Breakpoint field.

The new breakpoint appears in the Manage Breakpoints dialog.

- Click **OK**.
- Click **Done** to add the breakpoint to the source code.

➤ **To remove individual breakpoints using Toggle Breakpoints:**

- Click in a line that has a breakpoint.
- Select the **Toggle Breakpoint** icon in the Debugger toolbar.

➤ **To remove breakpoints using the Edit menu:**

- Choose **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
- Select the **Code** tab and click to select the breakpoints you want to remove.
- Click **Delete**.
The breakpoints disappear from the list in the Manage Breakpoints dialog.
- Click **Done**.
The breakpoints disappear from the source code.

➤ **To remove all breakpoints:**

- Select **Tools>Clear All Breakpoints** from the menu or click the **Clear All Breakpoints** icon in the Debugger toolbar:



Setting and removing breakpoints in exceptions

In addition to setting breakpoints in the main body of source code in your Java applications, you can set breakpoints in standard Java exceptions or in exceptions that you write to handle specific behaviors.

This capability can help you pinpoint the source of exceptions that are thrown unexpectedly when you run your application. When you set a breakpoint in an exception and then run your application, the Debugger will stop where the exception gets thrown and display the offending source code if it is available for the class you are debugging. If the Debugger cannot find the source code, it prompts you for a path, as described in “When the Debugger cannot locate source code” on page 26.

➤ **To set breakpoints in exceptions:**

1. Choose **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
2. Select the **Exception** tab and click **Add**.
The Add Exception Breakpoint dialog opens.
3. Type the fully qualified name of the exception class on which you want to break, or click **Browse** to select an exception from the list of loaded exceptions.
4. Click **OK** to return to the Manage Breakpoints dialog.
The new breakpoint is listed.
5. Click **Done**.
NOTE Exception breakpoints do not appear in the source code.

➤ **To remove breakpoints from exceptions:**

1. Choose **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
2. Select the **Exception** tab and click the breakpoints you want to remove.

3. Click **Delete**.
The breakpoints are removed from the list.
4. Click **Done**.

Enabling and disabling breakpoints

The Debugger allows you to enable and disable breakpoints in code and exceptions. Disabled breakpoints appear with a grayed icon:



➤ To enable breakpoints:

1. Select **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
2. Select the **Code** or **Exception** tab to locate the breakpoints you want to enable.
3. Click the breakpoints you want to activate and click **Enable**.
4. Click **Done**.
The breakpoints you selected will be enabled when you continue program execution.

➤ To disable breakpoints:

1. Select **Edit>Breakpoints** from the menu.
The Manage Breakpoints dialog opens.
2. Select the **Code** or **Exception** tab to locate the breakpoints you want to disable.
3. Click the breakpoints you want to activate and click **Disable**.
To select multiple breakpoints: hold down the **Ctrl** or **Shift** key when you click each choice.
4. Click **Done**.
The breakpoints you selected will be disabled when you continue program execution.

Continuing execution

When the program stops at a breakpoint you can continue execution using the **Continue** or **Run to Cursor** commands in the Debugger:

- Use the **Continue** command to resume running the program to the next breakpoint it encounters.

- Use the **Run to Cursor** command to continue running the program to a specified location in the code, marked by the cursor.

When the program reaches a breakpoint, the Debugger window is activated and an arrow (called the **execution pointer**) appears in the margin to the left of the code indicating which line is about to be executed.

Along with the execution pointer, the Debugger displays a list of local and instance variables and enables the commands for stepping through your code.

➤ To continue execution:

- Select **Tools>Continue** from the menu or click the **Continue** icon in the toolbar of the Debugger window:



The Debugger executes your source code until it reaches the next breakpoint.

➤ To run to a specified location in the source code:

1. Click in the line of code where you want execution to stop.
The cursor must be located on a line that contains executable statements.
2. Select **Tools>Run to Cursor** from the menu or click the **Run to Cursor** icon in the toolbar of the Debugger window:



Run to Cursor executes your code from the current execution location until it reaches the place where the cursor is located. If you have breakpoints set and a line containing a breakpoint is executed before the line that has the cursor, Run to Cursor stops execution at that breakpoint line.

NOTE If you put the cursor on a line that doesn't get executed—for example, if your cursor is on the first line of code inside an if statement when the if condition evaluates to false—the effect of executing Run to Cursor will be the same as if you select the Continue command.

Stepping through the code

There are three Step commands included in the SilverStream Debugger:

Command	Executes	Details
Step Over	The current line	If the current line contains a method call, that method is executed. The Debugger stops at the line immediately following the line that was executed.
Step In	The current line	If the current line contains a method call, the Debugger stops at the first line in the called method. Otherwise, the Debugger stops at the line immediately following the line that was executed. If the Debugger cannot find the source code for the method it has entered, it prompts you for a path, as described in “When the Debugger cannot locate source code”, next.
Step Out	The remainder of the current method	The Debugger stops at the statement immediately following the statement that called the current method.

When the Debugger cannot locate source code

When the Debugger cannot locate source code for a particular operation, it prompts you to provide the path to the source file. Either provide the path or dismiss the prompt and follow directions to either continue execution or step out of the current method.

Analyzing the behavior of the application

You can examine the current execution state at three points:

- When the Debugger stops at a breakpoint
- Following a Step command
- When you suspend and resume threads

At each of these points you can view the following information:

- Call stack

- Thread status
- Local and instance variables

Viewing the call stack

The Debugger provides a viewer that lets you examine the call stack during program execution. The call stack viewer shows the name of each method in the stack, along with its source file and line number.

When you double-click a method in the call stack viewer, the source file opens in the Debugger, highlighting the line where the method is called. If the Debugger cannot find the source file, it prompts you for a path, as described in “When the Debugger cannot locate source code” on page 26.

The call stack viewer updates the call stack when:

- The Debugger stops at a breakpoint
- You execute a Step command
- You continue execution to the next breakpoint
- You continue execution to a specific location in the code, marked by the cursor
- You select a different thread

➤ To open the call stack viewer:

- In the Debugger, choose **View>Call Stack** from the menu.
A pane for viewing the call stack appears in the Debugger window.

Viewing threads

The Debugger provides a viewer that lets you examine the status of threads during program execution. The thread viewer displays threads organized by groups, showing the name of each thread, along with its identifier and state.

You can suspend and resume threads to isolate problems that occur in thread synchronization, such as deadlocks and infinite loops.

The thread viewer updates thread status when:

- The Debugger stops at a breakpoint
- You execute a Step command
- You continue execution to the next breakpoint

- You continue execution to a specific location in your code, marked by your cursor
- You suspend and resume threads

➤ **To open the thread viewer:**

- In the Debugger, choose **View>Threads** from the menu.
A pane for viewing threads appears in the Debugger window.

Interpreting thread states

Threads can exhibit a variety of states during program execution:

Thread state	Description
At breakpoint	Thread was running when execution stopped at the breakpoint
Running	Thread is running
Sleeping	Thread is sleeping for a specified period of time
Suspended	Thread is suspended
Waiting	Thread is waiting for notification to resume running
Waiting on monitor	Thread is waiting for a monitor to be released by another thread NOTE In a deadlock situation, you will see two or more threads in this state. However, threads that appear in this state do not necessarily indicate a deadlock.

Suspending and resuming threads

If you encounter an infinite loop or deadlock in a thread, you can isolate the problem by suspending the thread and viewing its call stack. You cannot view the call stack of a thread unless it is suspended.

➤ **To suspend a thread and view its call stack:**

1. Open the thread and call stack viewers.

2. Do one of the following:

To suspend	Do this
All threads	<ol style="list-style-type: none"> 1. Stop execution of the code at a breakpoint. 2. Click the suspended thread.
An individual thread	Double-click a running or waiting thread in the thread viewer

The call stack for the selected thread appears in the call stack viewer.

3. Double-click methods in the call stack to view their code in the Debugger.

If you double-click a waiting thread, the method that put the thread in a wait state appears in the call stack viewer.

If the Debugger cannot locate the source code for a method you selected, it prompts you to supply a path, as described in “When the Debugger cannot locate source code” on page 26.

➤ To resume a thread:

- Do one of the following:

To resume	Do this
All threads suspended at a breakpoint	Resume execution of your application by: <ul style="list-style-type: none"> • Stepping in, out, and over source code • Running the application to a particular location in the source code, marked by the cursor • Running the application to the next breakpoint • Continuing execution
An individual thread	Double-click the suspended thread in the thread viewer

Viewing variables

The Debugger provides a viewer that lets you examine local and instance variables when program execution stops at a breakpoint. The variable viewer displays the variable name, type, and value.

The variable viewer updates variable values when:

- The Debugger stops at a breakpoint
- You execute a Step command
- You continue execution to the next breakpoint
- You continue execution to a specific location in your code, marked by your cursor

➤ **To open the variable viewer:**

1. In the Debugger, choose **View>Variables** from the menu.
A pane for viewing variables appears in the Debugger window.
2. Click **locals** to view local variables and click **this** to view instance variables for the current object.

Debugger keyboard shortcuts

Use these keyboard shortcuts.

Keystroke	Description
Ctrl+C	Copy to Clipboard
Ctrl+G	Go to line number
Ctrl+F	Find/Replace
F5	Continue
F10	Step over
F11	Step in
Shift+F11	Step out
Ctrl+F10	Run to cursor

Index

A

- abbreviations
 - see source files
- actions in Todo lists 42
- Apache Ant, using 44
- application client archives (CARs)
 - see archives 9
- Archive Contents view 74
- Archive Layout view 74
- archives
 - about 51
 - creating 91
 - creating projects for 9
 - defining deployment settings 102
 - deleting 115
 - deploying 98, 101, 108
 - deployment descriptors 295
 - deployment documents 101
 - deployment plans 301
 - directory structure considerations 53, 55
 - disabling 115
 - managing content 80, 99
 - rapid deployment 102
 - undeploying 115
 - validating 94
 - Workbench projects 9, 51
- autosave files
 - setting preferences 21

B

- backup files
 - setting preferences 21
- BEA WebLogic server
 - deploying archives to 108
 - deployment documents 101
 - deployment settings 102
- bookmarks in NetBeans-based editors 224
- breakpoints
 - see debugger
- browser preference 14

- Build command 91
- building projects 87

C

- call stacks
 - see debugger
- CARs (application client archives)
 - see archives
- catalog entry files 252
- catalog, XML 252
- class files, opening in Workbench 12
- Class Viewer 12
- classpath
 - specifying the project classpath 88
- code completion for Java expressions 218
- colors, setting in XML Editor 24
- Compile command 91
- compiler preferences 15
- compilers
 - specifying the Java compiler 88
- compiling projects 87
- components
 - see J2EE, source files
- Custom Tag Wizard 226
- custom tags
 - see JavaServer Pages

D

- databases
 - creating profiles 27
 - making a driver class available 27
- deadlock
 - see debugger
- debugger
 - about 307
 - attaching to a Java application 324
 - breakpoints 307, 326, 327, 330
 - call stacks 333, 334
 - controlling program execution 309

- deadlock 307
 - debugging client objects 309
 - debugging concepts 307
 - debugging server objects 309, 312
 - instance variables 308, 335
 - keyboard shortcuts 336
 - local and remote debugging 309
 - local variables 308, 335
 - locating source code 332
 - managing program execution 326, 330
 - monitor 308
 - monitoring program status 309, 332
 - options for launching applications 322
 - specifying your own 43
 - stepping through code 332
 - threads 308, 333, 334
 - using to start an application 322
 - workflow 311
 - Debugger command preference 43
 - default browser, used in XSL Editor 257
 - deployment
 - production/full 97
 - rapid 96
 - types 96
 - using non-Workbench tools 97
 - Deployment Descriptor Editor
 - about 295, 296
 - setting preferences 18
 - deployment descriptors
 - about 295
 - associating with projects 297
 - creating 297
 - validating against archives 94
 - deployment documents
 - about 101
 - listing for different application servers 101
 - Deployment Plan Editor
 - about 301
 - setting preferences 18
 - deployment plans
 - about 301
 - associating with projects 304
 - creating 302
 - modifying 303
 - SilverStream eXtend Application Server 101
 - validating 304
 - deployment settings
 - creating 102
 - deploy-only projects
 - see projects
 - directories
 - adding to projects 68, 71
 - display preferences 15
 - driver classes
 - see databases
 - DTD catalog 252
 - DTDs (Document Type Definitions)
 - attaching to XML documents 235
 - converting to Schemas 238
- ## E
- EJB archives (EJB JARs)
 - see archives 9
 - Enable Todo preference 42
 - enterprise archives (EARs)
 - see archives 9
 - environment variables
 - using for project settings 80
- ## F
- files
 - see source files
 - files, specifying editor to use on 20
 - fonts
 - used by Workbench, specifying 49
 - used in native editors 16
 - full deployment 97
- ## G
- graphics, opening in Workbench 12

I

- IBM WebSphere server
 - deploying archives to 108
 - deployment documents 101
 - deployment settings 102
- Image Viewer 12
- inner classes, listing 76
- instance variables
 - see debugger
- internationalization support 49

J

- J2EE
 - archives 9
 - components 9
 - creating components 117
 - deployment descriptors 9, 295
 - META-INF directories 295
 - Web Services 9
 - WEB-INF directories 55, 295
- Jakarta Tomcat
 - deploying archives to 108
 - deployment documents 101
- Java archives (JARs)
 - see archives 9
- Java class files
 - opening in Workbench 12
- Java Editor 211
- Java expressions, code completion in editors 218
- JavaServer Pages
 - editing 211, 226
 - inserting custom tags 226
 - JSP Editor 211
- jBroker Web
 - compilers used by Web Service Wizard 188
- JSP Editor 211

L

- Launch Action command 42
- line numbers
 - displaying in editors 16
 - printing 17

- local variables
 - see debugger

M

- META-INF directories
 - see J2EE
- monitor
 - see debugger

N

- native editors, editing files with 224
- Navigation Pane
 - refreshing 71
- NetBeans-based editors
 - adding file types edited with 221
 - using 216

O

- OASIS XML catalog standard 252
- Oracle9iAS server
 - deploying archives to 108
 - deployment documents 101
 - deployment settings 102

P

- permissions
 - changing 4
- preferences
 - abbreviations 19
 - autosave 21
 - backup 21
 - build 15
 - deployment 18
 - display 15
 - file type 20
 - general 14
 - printing 17
 - setting 13
 - text editing 16

- version control 23
- XML Editor colors 24
- printing
 - specifying preferences 17
- production deployment 97
- profiles
 - creating 24
 - creating database profiles 27
 - creating registry profiles 282
 - creating server profiles 24
- project files
 - about 52
 - closing 7
 - opening 6, 77
 - saving 6
 - working with 6
- projects
 - about 51
 - adding multiple files at the same time 70
 - adding source files and directories 68, 71, 81
 - adding subprojects 71
 - adding to the project classpath 90
 - compiling, building, and archiving 87, 91
 - creating 9, 56
 - creating a project that includes existing source files 64
 - creating source files 65
 - defining deployment settings 102
 - deploying 98, 108
 - deployment plans 301
 - deploy-only projects 62
 - designing 53
 - displaying in the Navigation Pane 3, 74
 - maintaining 76
 - managing content 80, 81, 85
 - modifying project entries 81
 - opening 77
 - organizing 53
 - populating 65
 - Project menu 91
 - refreshing contents of 71
 - removing files 85
 - renaming 87
 - setting preferences 14
 - settings 78, 80, 81

- specifying the classpath 88
- subprojects 53, 56
- tracking tasks 37
- using relative directory paths 81
- using the project popup menu 86
- viewing 74
- working with existing source files 64
- working with project files 6, 53
- proxy servers, using with Workbench 5

Q

- quick deployment 96

R

- rapid deployment 96
 - see archives
- Rebuild command 91
- Refresh command 71
- Refresh Schema Handler 237
- registries
 - see Web Services
- rmi2soap compiler
 - in Web Service Wizard 188
- rmi2wsdl compiler
 - in Web Service Wizard 188

S

- Schema catalog 252
- Schema Guide 244
- Schemas
 - attaching to XML documents 235
 - creating from DTDs 238
- servers
 - creating deployment settings 102
 - creating profiles 24
 - using secure servers, SSL, and HTTPS protocol 27
- SilverDebugger.exe
 - see debugger

- SilverStream eXtend Application Server
 - deploying archives to 108
 - deployment plans 101
 - deployment settings 102
 - source control
 - see version control
 - source files 9
 - abbreviations in 19, 214
 - adding to projects 68, 71
 - bookmarks (NetBeans-based editors) 224
 - catalog entry files 252
 - changing case (native editors) 215
 - changing DOS and UNIX line endings (native editors) 224
 - changing read-only and write-only attributes (native editors) 225
 - changing spaces, tabs, and indentation (native editors) 215
 - clipboard support (native editors) 225
 - closing 8
 - code completion for Java expressions (NetBeans-based editors) 218
 - color coding (NetBeans-based editors) 217
 - compiling a Java file 91
 - creating 65
 - creating components 9, 117
 - debugging 307
 - defining how a file type is launched 20
 - deleting 8
 - directory structure considerations 53
 - displaying in the Edit Pane 3
 - editing 10, 211
 - graphics, opening in Workbench 12
 - inserting JSP tags (native editor) 226
 - opening 7
 - renaming 8
 - saving 7
 - searching 213
 - searching (NetBeans-based editors) 223
 - searching across multiple files 213
 - setting preferences 14, 20
 - setting text editing preferences 16
 - src directories 53
 - using NetBeans-based editors 216
 - using the native editors 224
 - using the native editors for Java, JSP, and HTML files 225
 - working with 7
 - Source Layout view 74
 - SPF files
 - see project files
 - src directories 53
 - status bar 4
 - subprojects
 - adding to projects 71
 - creating 56
 - displayed in Navigation Pane 74
 - parent project classpaths 89
 - Sun J2EE Reference Implementation server
 - deploying archives to 108
 - deployment documents 101
- ## T
- tabs
 - using in source editor 16
 - Text Editor 211
 - threads
 - see debugger
 - Todo lists 37
- ## U
- UDDI
 - see Web Services
 - undeploying archives 115
- ## V
- validation of archives 94
 - validation of XML documents 249
 - version control
 - accessing 36
 - setting up access 29
 - using 29
 - version information about Workbench 5

W

Web archives (WARs)

see archives 9

Web Service Wizard

about 187

compilers 188

panel details 191

panel sequence 189

Web Services

browsing registries 284

creating registry profiles 282

publishing to registries 293

registries 281

retrieving WSDL files from registries 292

tools provided in Workbench 13

UDDI 281

WSDL Editor 13, 267

WEB-INF directories

see J2EE

Workbench

about 1

basic operations 5

creating profiles 24

displaying version information about Workbench
components 5

exiting 5

extending tools and services 50

panes 3

printing 17

setting preferences 13, 21

specifying the Java compiler 88

specifying the project classpath 88

starting 5

WSDL (Web Services Description Language)

see Web Services

wsdl2java compiler

in Web Service Wizard 188

XML Editor 229

catalog, used by 252

code completion 239

context editing support 239

keyboard shortcuts 258

Schema Guide 244

setting colors 24

validating documents 249

XML Wizard 233

xsd2java compiler

in Web Service Wizard 188

XSL Editor 256

X

XML catalog 252

XML Catalog Editor 254

XML Catalog Wizard 253