Novell exteNd Director

5.2

www.novell.com

UTILITY TOOLS

Novell®

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.

www.novell.com

exteNd Director*Utility Tools*
June 2004

Online Documentation: To access the online documemntation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

ConsoleOne is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Composer is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc.
Novell eGuide is a trademark of Novell, Inc.

SilverStream Trademarks

SilverStream is a registered trademark of SilverStream Software, LLC.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Third-Party Software Legal Notices

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. 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 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 "Apache" 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.

JDOM.JAR

Copyright (C) 2000-2002 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).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos.

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 JDOM AUTHORS OR THE PROJECT 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

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.

Indiana University Extreme! Lab Software License

Version 1.1.1

Copyright (c) 2002 Extreme! Lab, Indiana University. 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 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 Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. 5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

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 AUTHORS, COPYRIGHT HOLDERS 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.

Phaos

This Software is derived in part from the SSLavaTM Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved. Customer is prohibited from accessing the functionality of the Phaos software.

W3C

W3C® SOFTWARE NOTICE AND LICENSE

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications: 1.The full text of this NOTICE in a location viewable to users of the redistributed or derivative work. 2.Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code. 3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Book

Purpose

This book explains how to use the exteNd Director™ utility tools. The utility tools are basic project and programming facilities provided to support your application development work in the Novell® exteNd Director development environment.

Audience

The information in this book applies to application developers and application deployers.

Prerequisites

This book assumes that you are familiar with the following standard technologies used in Web applications:

- HTML, XML, XSL, and CSS
- Java and J2EE

Learning materials on these topics are 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
Development Environment	Use the basic features and common facilities of the exteNd Director development environment
Projects and Archives	Work with the projects you create to generate application archives
Source Editors	Use the source editing capabilities that exteNd Director provides for standard file types (including Java, XML, HTML, text)
XML and CSS	Use the exteNd Director utility tools for working with XML and CSS files
Web Services	Use the exteNd Director utility tools for working with Web Services
J2EE	Use the exteNd Director utility tools for working with J2EE components
Deployment	Use the exteNd Director utility tools for deploying J2EE archives

1 Development Environment

This chapter introduces the **utility tools** and other **basic facilities** of the Novell exteNd Director development environment. You'll learn about the supporting features provided to assist you and your application development work in the higher-level exteNd Director tools. Topics include:

- About the utility tools
- Basic panes
- Basic operations
- Basic wizards
- Basic editors
- Basic viewers
- Basic tools for Web Services
- Debugging facilities
- Managing toolbars
- Setting preferences
- Setting up profiles
- Using version control
- Maintaining Todo lists
- Using Ant
- Internationalization support
- Extending the development environment

 For more information about performing basic project-level operations (such as creating a generic project, adding files to a project, building a project, and creating an archive) see Chapter 2, "Projects and Archives".

About the utility tools

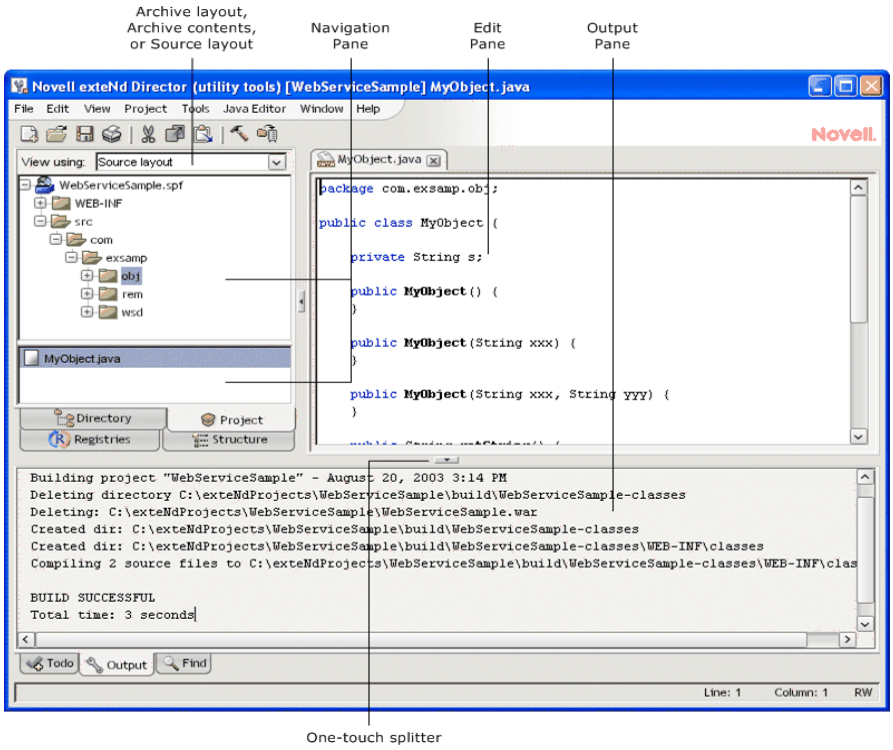
At its lowest level, the exteNd Director development environment is a file-system based toolset that includes these utility tools and facilities:

- **Graphical and text-based editors** for working on Java files, JSP files, XML files, XSL files, CSS files, WSDL files, HTML files, plain text files, and deployment descriptors
- **Wizards** that help create new files when needed and guide you through complex technologies (such as J2EE)
- **Web Service facilities** for developing, publishing, finding, and running Web Services
- **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, generating and validating J2EE archives, and deploying archives to supported J2EE servers
- **Version control integration** that provides access from the exteNd Director development environment to your version control system

Basic panes

The exteNd Director development environment includes three resizable panes:

- Navigation Pane
- Edit Pane
- Output Pane



You can use the **one-touch splitters** to collapse or restore panes with a single click.

Navigation Pane

The Navigation Pane lets you access various aspects of projects and registries. It displays the following tabs:

- 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
- The **Structure** tab lets you browse the class members of the Java source file currently selected in the Edit Pane

The Structure tab appears only when a Java source file is open. 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.

Using the Navigation Pane

Here's how to use the Navigation Pane:

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	Window menu	Use the Window 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 62.
For a Java source file, view and navigate the structure and see type and visibility of its members	Structure tab	Click the Structure tab after you open a Java source file in the Edit Pane. Use the drop down menu near the top to sort the classes, methods, and fields of the file in several ways. Double-click a class member to position the cursor in the Edit Pane to the source code for that member.

Displaying additional information

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. This is 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).

In the Structure tab, pointing at a class member shows its signature.

Structure tab icons

In the Structure tab, each item is identified with a double icon. The left icon indicates the visibility of the item. The right indicates its type. The table below defines the icons:

Icon category	Icon	Description
Visibility		(Lock) Protected
		(Blue cube) Package protected
		(Key) Private
	No icon	Public
Type		(Document) Class
		(Gray cube) Field
		(Plus symbol) Constructor
		(Function symbol) Method or function

Edit Pane

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

Output Pane

The Output Pane provides several informational tabs that you can access:

- The **Todo** tab displays and manages Todo list items
- The **Output** tab displays output from the build, validate, and deploy processes
- The **Find** tab displays search results
- The **Version Control** tab (if enabled) displays messages from your version control system

Other types of messages and indicators display in the **status bar**, which is located below the Output Pane.

Basic operations

This section tells you about:

- Starting the development environment
- Using proxy servers
- Opening, saving, and closing projects and files
- Getting product version information

Starting the development environment

This section describes how to start and exit the exteNd Director development environment.

➢ **To start the development environment:**

- Depending on your platform, do one of the following:

Platform	What to do
Windows	**1** Go to the list of all programs on the Windows **Start** menu. **2** Select **Novell exteNd *n*.*n*>Director>Director Designer**.
Windows, Linux	**1** Go to the Novell exteNd **tools/bin** directory. **2** Run the executable file named **xd**.

➢ **To exit the development environment:**

- Select **File>Exit** from the exteNd Director menu bar.

Using proxy servers

If you are using a proxy server, you need to specify the proxy host and its port in **xd.conf** (in the Novell exteNd™ tools\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 the exteNd Director development environment:

- 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**. exteNd Director project files have the extension **.SPF**.

➢ **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). exteNd Director 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 you open another (unrelated) project, exteNd Director 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** of the Navigation Pane.
2. Double-click the file (or right-click it and choose **Open** from 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 the exteNd Director development environment. 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). exteNd Director displays the file in the Edit Pane using the appropriate source file editor (Java, JSP, XML, Text, etc.).

Alternatively, you can:

1. Navigate to the file in the **Directory tab** of 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. Double-click the file (or right-click it and choose **Open** from 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 26.)

You can also make an open file the active file by using **Window>More Windows** to select it 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 on its tab in the Edit Pane.

  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, exteNd Director prompts you to save that file before closing it, closing its parent project, or exiting the development environment.

Performing file system operations

You can delete and rename files from the exteNd Director development environment.

➢ **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.
2. Select the files to delete. You can select multiple files using Shift+Click and Ctrl+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 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.

Getting product version information

Use the About dialog (**Help>About Director**) to get version information about the exteNd Director development environment and its tools (editors, wizards, viewers, and so on). This can be useful if you need to check which product components you have installed.

Basic wizards

To assist your development work on various supporting application files, you can use the basic wizards provided by the exteNd Director development environment:

Use these basic wizards	To do this
XML and CSS wizards	◆ Create XML files, XML Schema files, and XML catalog files 📖 See Chapter 4, "XML Editors". ◆ Convert DTD files to XML Schema files 📖 See Chapter 4, "XML Editors". ◆ Create XSL style sheet files 📖 See Chapter 5, "XSL Editor". ◆ Create CSS style sheet files 📖 See Chapter 6, "CSS Editor".
Web Service wizards	◆ Generate the Java classes you need to create and access standard Web Services 📖 See Chapter 10, "Web Service Wizard". ◆ Create WSDL (Web Services Description Language) documents 📖 See Chapter 11, "WSDL Editor".
Generic J2EE project wizards	Create projects for generic J2EE archives, including: ◆ 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 📖 See "Creating projects and subprojects" on page 50.
General J2EE file wizards	Create standard J2EE components, including: ◆ JavaServer Pages ◆ Servlets ◆ Java classes ◆ Enterprise JavaBeans ◆ JavaBeans ◆ Tag handlers 📖 See "Creating source files" on page 57 and Chapter 13, "J2EE Wizards".
Deployment wizards	Create deployment descriptors and exteNd deployment plans 📖 See Chapter 16, "Deployment Descriptor Editor" and Chapter 17, "Deployment Plan Editor".

Basic editors

When you open a source file in the exteNd Director development environment, the appropriate editor automatically displays in the Edit Pane. Here's a summary of the basic source editors you get:

If you open	The Edit Pane displays	For more information, see
Java file	Java Editor	Chapter 3, "Source Editors"
JSP file	JSP Editor	Chapter 3, "Source Editors"
HTML file	HTML Editor	Chapter 3, "Source Editors"
Text file	Text Editor	Chapter 3, "Source Editors"
XML file	XML Editor	Chapter 4, "XML Editors"
XML Schema file	Schema Editor	Chapter 4, "XML Editors"
XML catalog file	XML Catalog Editor	Chapter 4, "XML Editors"
XSL file	XSL Editor	Chapter 5, "XSL Editor"
CSS file	CSS Editor	Chapter 6, "CSS Editor"
WSDL file	WSDL Editor	Chapter 11, "WSDL Editor"
J2EE deployment descriptor file	Deployment Descriptor Editor	Chapter 16, "Deployment Descriptor Editor"
exteNd deployment plan file	Deployment Plan Editor	Chapter 17, "Deployment Plan Editor"
JAR, J2EE archive, or ZIP file	A read-only listing of the contents	—

 To learn about the core functionality provided in all exteNd Director editors, see Chapter 3, "Source Editors".

Basic viewers

The exteNd Director development environment provides the following tools for viewing (but not editing) some other file types:

- Image Viewer
- Class Viewer

Image Viewer

Opening a GIF, JPG, JPEG, or PNG file displays the Image Viewer in the Edit Pane. You can zoom the image:

If you want to	Do this
Zoom in	Left-click or press +
Zoom out	Ctrl+left-click or press -
Restore the image to its actual size	Shift+left-click or press =

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 association preferences" on page 30.

Class Viewer

If you open a .class file, information about the .class file displays 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

Basic tools for Web Services

The basic Web Service facilities of the exteNd Director development environment include:

- 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, and viewing WSDL files
- A **Registry Manager** for publishing and discovering Web Services

 For more information, see Chapter 10, "Web Service Wizard", Chapter 11, "WSDL Editor", and Chapter 12, "Registry Manager".

Debugging facilities

While working in the exteNd Director development environment, you can troubleshoot J2EE and other Java code by running the debugger of your choice.

➢ **To use a debugger:**

- Select **Tools>Launch Debugger**.

  It prompts for a command to launch your debugger, then executes it. (For guidelines, see "Specifying the debugger command" below.)

You can specify a default debugger command on the General tab of the Preferences dialog.

➢ **To set a preference for your default debugger command:**

1. Select **Tools>Preferences**.
2. In the **Debugger command** setting on the General tab, specify the command to launch your debugger. (For guidelines, see "Specifying the debugger command" below.)
3. Click **OK**.

Specifying the debugger command Specify the operating system command that exteNd Director should issue when you select **Tools>Launch Debugger**. See your debugger's documentation for information about how to invoke that debugger from the command line.

You can include environment variables in the command by using the syntax `%varname%` or `${varname}`. exteNd Director 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 in the exteNd Director development environment:

- You can use the same variables that the exteNd Director version control interface uses (see "Predefined environment variables" on page 37).

  The file-related environment variables (such as %_PATH%) refer to the file that is open and currently active in the exteNd Director development environment.
- 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

Managing toolbars

This section tells you how to control the use of toolbars in the exteNd Director development environment:

- Displaying toolbars
- Configuring toolbars

Displaying toolbars

The **Main Toolbar** (which appears below the menu bar) is the global toolbar for the development environment. Individual tools can have their own toolbars as well (in the Edit Pane). You can choose to show or hide any of these toolbars.

➢ **To show or hide a toolbar:**

- **Select View>Toolbars** to list the available toolbars, then choose one to toggle it on or off.

  OR

  Right-click a toolbar to display the popup menu, then choose one of the listed toolbars to toggle it on or off.

How tabs display Toolbars with one tab display as regular toolbars. Toolbars with more than one tab display as tabbed toolbars.

Configuring toolbars

You can configure any toolbar to control the buttons and tabs it displays.

➢ **To configure a toolbar:**

1 Right-click that toolbar and select **Configure**.

2 When the **Toolbar Configuration dialog** displays, use it to make one or more of these changes:

If you want to	Do this
Add buttons	**1** Browse the **categories** at the top of the dialog to find a predefined button you want. **2** If the toolbar has multiple tabs, make the appropriate tab current in the **toolbar canvas** at the bottom of the dialog. **3** **Drag your button** to the toolbar canvas and drop it at the position you want. OR **Select your button** and click the Add Button control to place it at the end of the current tab. **TIP:** The Miscellaneous category includes a **separator** (vertical bar) that you can insert to visually organize toolbar buttons into groups.
Move buttons	**Drag and drop a button** within the current tab in the toolbar canvas. OR **Select a button** in the toolbar canvas and click the Move Button Left or the Move Button Right control.
Delete buttons	**Drag a button** from the toolbar canvas and drop it anywhere outside. OR **Select a button** in the toolbar canvas and click the Remove Button control. OR **Right-click a button** in the toolbar canvas and select Delete.
Add tabs	Click the **Add Tab** button, then specify the tab name and tool tip text.
Rename tabs	**1** Make the appropriate tab current in the toolbar canvas. **2** Click the **Modify Tab** button, then change the tab name and/or tool tip text.
Move tabs	**1** Make the appropriate tab current in the toolbar canvas. **2** Click the **Move tab - left** button or the **Move tab - right** button.
Delete tabs	**1** Make the appropriate tab current in the toolbar canvas. **2** Click the **Delete Tab** button.

3 Click **OK**.

Setting preferences

You can configure your exteNd Director development environment by setting:

- General preferences
- Build preferences
- Display preferences
- Editing preferences
  - Abbreviation preferences
  - Backup preferences

- Editor association preferences
- Code completion preferences
- XML color preferences
- File association preferences
- Printing preferences
- Deployment preferences
- Version control preferences

➢ **To specify preferences:**

1. Select **Tools>Preferences**.

   The Preferences dialog displays.
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 File menu. Default is 10.
Number of recent projects	Specifies how many recently open projects appear in the File menu. Default is 5.
Reload open projects	When starting, automatically reloads the projects that were open when you last exited the exteNd Director development environment. Default is No.
Reload open files	When starting, automatically reloads the files that were open when you last exited the exteNd Director development environment. Default is No.
Enable Todo	Specifies whether the Todo feature is enabled. When selected, exteNd Director 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 39.
Web browser	Specifies the Web browser to use when displaying exteNd product help. Choose a browser by typing the path or clicking the button.
Debugger command	Specifies the command that exteNd Director invokes when you select **Tools>Launch Debugger** to run a debugger of your choice. See "Debugging facilities" on page 22.
Help documentation location	Specifies where the development environment will look for exteNd product help when you request it. This can be either a path (typically on your local disk) or an URL (typically on the Novell exteNd documentation Web site). The installation program populates this setting with a default path or URL (depending on whether you choose to install the help locally).
Reset "Don't show me this message again" dialogs	Pressing the **Reset** button causes all dialogs that have a "Don't show me this message again" checkbox to resume displaying.

Build preferences

Specify build preferences as follows:

Setting	Description
Always save modified files before compiling	When set (the default), automatically saves all modified files before compiling, building, or rebuilding. When this property is not set, exteNd Director prompts you about saving unsaved files. 📖 For more save option preferences, see "Backup preferences" on page 28.
Compiler	Specifies the compiler. Default is **Javac 1.3 ("Modern")**. If you choose **Jikes** for this setting, note that the Jikes compiler is not provided when you install the exteNd Director development environment. You must obtain the Jikes compiler yourself and add it to your system PATH.
Generated class version	Specifies which JRE version the compiler is to target. When you compile for 1.4 (the default), your code can include 1.3 or 1.4 classes. When you compile for 1.3, your code can include only 1.3 classes.
Compiler options	Allows you to enable common command-line options for the compiler.

Display preferences

Specify display preferences as follows:

Setting	Description
Directory tab	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.
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.
For icons on navigation panes	Specifies whether you want to show large or small icons, with or without text, in the Navigation Pane.
In edit pane	Specifies the following: ◆ Whether the Edit Pane displays tabs for each open file ◆ Where the tabs are displayed relative to the editor (top or bottom)
Use native look and feel	Check this setting if you want the GUI characteristics (such as color scheme) of your native operating system to apply in the exteNd Director development environment. By default, the development environment uses its own look and feel. (If you change this setting, you must restart the development environment to see the result.)

Editing preferences

Specify editing 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 Printing tab; see "Printing preferences" on page 31.
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.
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 scrollbars	Choices are: only as needed (the default) or always.

For additional text options, see Chapter 3, "Source Editors".

Abbreviation 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 press **Ctrl+U** (or right-click and select **Text Tools>Complete Abbreviation**) 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 **Tools>Preferences**, click the **Editing** tab, and expand the **Abbreviations** section.
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 **Tools>Preferences**, click the **Editing** tab, and expand the **Abbreviations** section.
2. Select the abbreviation in the Abbreviations text box.
3. Click **Delete**.
4. Click **OK**.

➢ **To edit an abbreviation:**

1. Select **Tools>Preferences**, click the **Editing** tab, and expand the **Abbreviations** section.
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. Press **Ctrl+U** (or right-click and select **Text Tools>Complete Abbreviation**). 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.

Backup preferences

You can set 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 the exteNd Director development environment, 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 files saved (default is every five minutes)
Backup enabled	When you save a source file in the exteNd Director development environment, 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 **Tools>Preferences**, click the **Editing** tab, and expand the **Backup** section.

2 Choose **Auto save enabled** and/or **Backup enabled**.

3 Specify autosave and/or backup file parameters as described above.

4 Click **OK**.

Editor association preferences

These preferences specify which types of files will be edited using the NetBeans-based editors in the exteNd Director development environment.

📖 See "Adding files types edited by NetBeans-based editors" on page 87 and "Using the native Java, JSP, or HTML editor" on page 89.

Code completion preferences

📖 See "Creating parser database files" on page 86.

XML 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 **Tools>Preferences**, click the **Editing** tab, and expand the **XML colors** section.

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.

File association preferences

exteNd Director 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 the exteNd Director development environment. Use the File Associations tab to associate file extensions with external editors. For each file type, you can choose whether to open the file in:

- The exteNd Director development environment
- The system default editor for that file type (if you're using Windows)
- An external program you specify

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

1 Select **Tools>Preferences** and click the **File Associations** 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 Director	(The default) Opens files using the exteNd Director source editor.
Open using the default Windows program	(Windows only) 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 **exteNd** **deployment plans**, you will not be able to take advantage of the default setup that the exteNd Director editor provides and your project will not be associated with a deployment plan.*

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 Editing tab; see "Editing preferences" on page 27.
Print line numbers	Sets whether or not to print line numbers. Default is not to 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, exteNd Director 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 exteNd Director 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	Builds the project automatically when the Deployment Descriptor Editor or Deployment Plan Editor is opened. This setting ensures that the editors always have access to all of the latest classes.
Never automatically build my project	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. 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.
Prompt me to build my project	Prompts you to build the project each time you open one of the deployment editors. Use this setting when you want to specify when a project build should occur.

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	Specifies the server version initially selected when you create a new exteNd Application Server deployment plan. You can specify a server version for J2EE 1.2 projects and a server version for J2EE 1.3 projects. 📖 For more information, see Chapter 17, "Deployment Plan Editor".

Version control preferences

📖 See "Using version control" on page 35.

Setting up profiles

You can define the following types of profiles for use in the exteNd Director development environment:

- Server profile
- Database profile
- Registry profile

Server profile

A **server profile** stores information about a J2EE server, including the server's host name and port. When selected at deployment time, the server profile tells exteNd Director which 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 J2EE server, see the product documentation for that server.

➢ **To create a server profile:**

1 Select **Tools>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 meaningful name to identify the profile. The name cannot contain the period (.) character. **TIP:** Define a naming scheme based on your own development environment. For example, you might want to include project names, server names, server types, database types, and so on.
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.

Deployment tools directory	Specify the directory that contains the server's deployment tools—typically, a path such as:
	`C:\Program Files\Novell\exteNdn\AppServer\bin`
	or:
	`C:\bea\wlserver6.1spn\bin`
	If the server is located on another machine, you need either network access for running the tools or a copy of the tools in a local directory. For some servers, exteNd Director doesn't support remote deployment.

Setting	Description
Rapid deployment directory	For rapid deployment only. Enter the directory where you want exteNd Director to write the files for rapid deployment. Some servers require that files be written to a specific directory for rapid deploys. Make sure you specify the appropriate location for your server's configuration. See "Setting rapid deployment directories" on page 33 for the directory listings. **TIP:** Rapid deployment handles J2EE code, such as JSP pages and servlets. To quickly deploy resources such as components, images, and rules, use the exteNd Director Hot Deploy feature, described in Dynamic loading of resources and classes. For more information on rapid deployment, see "Rapid deployment" on page 276.
Server name	Set the server name using the following formats. For servers running http: *servername* http://*servername*[:*port*] For servers running https: https://*servername*[:*port*] Specify the port number if the server is not listening on the default port.
Database name	For exteNd application servers only. Specify the name of the database on the server where you want to deploy the archive. Typically you will deploy the archive to the SilverMaster database, so that the URL for the application won't need to include the database name. **NOTE:** You do not have to deploy the archive to the database you created for exteNd Director application data (described in the section on deploying an exteNd Director project in *Developing exteNd Director Applications*).
Target servers	For BEA WebLogic servers only. Enter the names of the target servers.

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
Novell exteNd Application Server SilverStream® eXtend Application Server	%INSTALL_DIR%\webapps
Apache Tomcat	%INSTALL_DIR%\webapps
BEA WebLogic	%INSTALL_DIR%\config*targetname*\applications

Connecting to secure servers The exteNd Director development environment connects to the target J2EE server at deployment time using the server profile. If the server profile indicates a secure server, exteNd Director will make the SSL connection automatically. It uses the set of commercial Certificate Authority certificates listed in **agrootca.jar** (located in the Novell exteNd Common\lib directory). If the server you're trying to deploy to uses a certificate issued by a CA certificate not listed in agrootca.jar, exteNd Director will not successfully connect to the server. You can add the CA certificate to agroootca.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 to use tools that require database access in the exteNd Director development environment. For example:

In this tool	You need a database profile
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 exteNd Director 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 **Tools>Profiles**.

2 On the **Databases** tab of the Profiles dialog, click **New**.

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	Enter the class name of the JDBC driver. You can specify any JDBC 2.0-compliant driver.
	To use the MySQL **Connector/J driver** (included with the exteNd Director development environment), specify:
	`com.mysql.jdbc.Driver`
	To use the Sun **JDBC-ODBC bridge driver** (in the JRE included with the exteNd Director development environment), specify:
	`sun.jdbc.odbc.JdbcOdbcDriver`
	If you specify **any other JDBC driver**, make sure that driver class can be loaded by the exteNd Director development environment; see "To make the driver class available:" below. (For the MySQL Connector/J driver and the Sun JDBC-ODBC bridge driver, the classes are set up automatically.)

JDBC URL	Enter an URL that specifies the database you want. For example:
	`jdbc:mysql://localhost:63306/SalesDB?profileSql=false&maxRows=0`
	or:
	`jdbc:odbc:TestDB`
	NOTE: The text you enter after the first colon is driver specific.

Setting	Description
Connection Catalog	(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. 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).

Datasource Name	Specify the name of the datasource to associate with this database profile. You can enter either the datasource name, such as:
	`SilverBooks`
	or the full JNDI specification, such as:
	`java:pm/JDBC/SilverBooks`

3 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.

4 In the test dialog, enter your database user name and password, then click **OK** to verify access.

5 Click **OK** to close the Create a New Database Profile dialog.

6 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 Novell exteNd **tools\lib\ext** directory.
- Edit the development environment configuration file (**xd.conf** in the Novell exteNd tools\bin directory) to point to the driver archive by including the line `addcp path/mydriver.jar`. For example:

```
addcp c:/sybase/SybJConnect.jar
```

3 Start the exteNd Director development environment.

Registry profile

The exteNd Director development environment provides a facility to define profiles for Web Service registries. These profiles supply the information that allows you to search registries and publish Web Services.

📖 For more information on registry profiles, see "Defining registry profiles" on page 204.

Using version control

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

- Setting up access to version control
- Accessing version control

Setting up access to version control

Before you can perform version control operations in the exteNd Director development environment, 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 **Tools>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 the development environment.
3. Select one of the available **Version Control Systems**.

   In this property, you're actually choosing a version control system **definition** that tells the development environment which version control commands to support. exteNd Director comes with definitions for several popular version control systems (ClearCase, CS-RCS, CVS, 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 development environment 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 the development environment 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.

If you want to	Do this
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 `%varname%` or `${varname}`. exteNd Director 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.
Reload when done	Tells exteNd Director 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 exteNd Director 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 exteNd Director 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
%_PATH%	Full path and name of the target file. For example: `x:/com/myco/myfile.java`
%_DIR%	Directory of the target file. For example: `x:/com/myco`
%_NAME%	Name of the target file (without directory). For example: `myfile.java`
%_BNAME%	Base name of the target file (without directory and extension). For example: `myfile`
%_EXT%	Extension of the target file. For example: `java`

Variable	Description
%_PROMPT *prompt-text*%	Prompts the user for a value by displaying a dialog. The dialog includes any *prompt-text* 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 the exteNd Director development environment doesn't provide a definition for your version control system, you can create one yourself.

➢ **To create a definition:**

1. From the **Version Control** tab of the Preferences dialog, click the **Add** button.
2. 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.

   exteNd Director 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). exteNd Director saves your definition XML file in the Novell exteNd **tools\Resources\version_control_config** directory (along with the definition XML files it provides).
3. 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 the exteNd Director development environment is installed.

➢ **To distribute a definition:**

1. Find the XML file for your version control system definition in the Novell exteNd **tools\Resources\version_control_config** directory.
2. Copy that file to the corresponding directory on each target computer.

   When the development environment 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:**

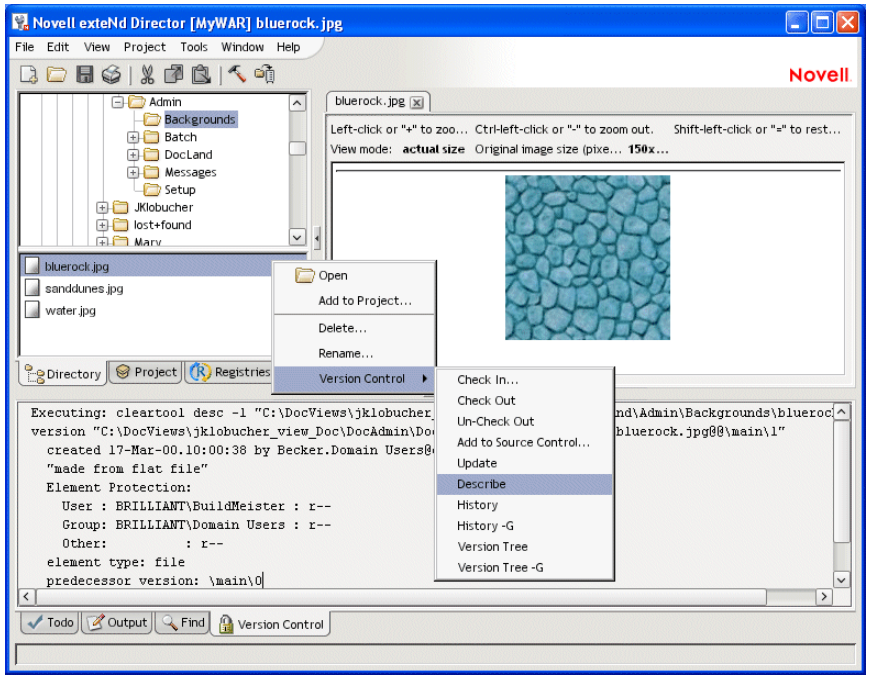
1. From the Version Control tab of the Preferences dialog, select a definition from the **Version Control Systems** dropdown list.
2. Click the **Remove** button.

   exteNd Director prompts you to confirm, then deletes that definition from the list. The definition's XML file is deleted from the Novell exteNd **tools\Resources\version_control_config** directory.

Accessing version control

When you use the exteNd Director development environment 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:

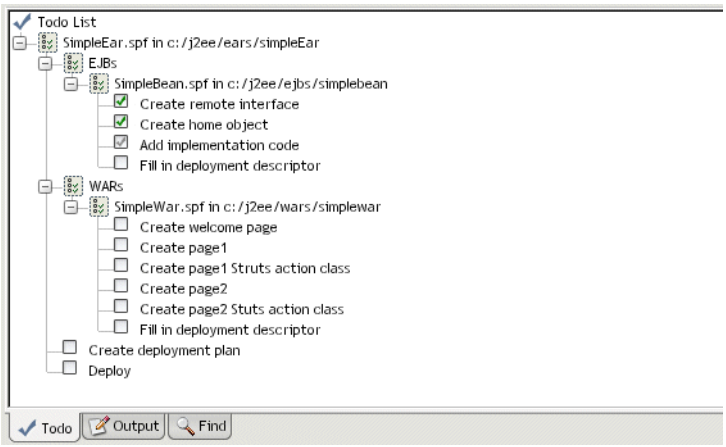
- Any file name on the Directory tab or Project tab of the Navigation Pane
- An open file in the Edit Pane



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

Maintaining Todo lists

The exteNd Director development environment enables you to maintain a Todo list that organizes and tracks your application development tasks. You maintain your Todo list in the Todo tab of the Output Pane.



You can:

- Create Todo items
- Associate items with 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 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:

Property	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	If you want to associate this item with an open project, select the project from the list.
	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:
	`projectFile in pathToProject`
	For example, if a project file is in c:\myProjects\myEAR\MyEAR.spf, the project folder will be named:
	`MyEAR.spf in c:/myProjects/myEAR`

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.

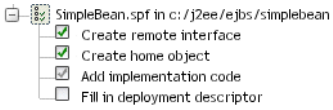
2 Right-click and select **Edit Item**.

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**.

An empty checkbox indicates that the task has not begun. A light check indicates partial completion. A dark check indicates completion.



TIP: You can 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. You can also do this simply by clicking the item's checkbox.

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 item 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.)

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:

Key(s)	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
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+/	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 (**Tools>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 reenable Todo.

Working with generated items

Various wizards and tools in the exteNd Director development environment 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, exteNd Director opens the servlet's source file and positions the insertion point appropriately.

Using Ant

Internally, the exteNd Director development environment uses Apache Ant when you build a project by selecting one of the Build commands on the Project menu (see "Compiling, building, and archiving" on page 71). You don't need to know anything about Ant if you only want to do builds that way. But direct access to Ant is also provided 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 Apache subproject. To learn more about Ant, including details on defining your own tasks and creating buildfiles, see ant.apache.org.

Using the exteNd Ant tools

You can use a couple of exteNd tools to invoke Ant from the command line. These Ant-based executables are in the Novell exteNd tools\bin directory:

- **xdbuild** allows you to build an exteNd Director project
- **xdant** 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 xdbuild takes an exteNd Director project file as input, and xdant takes an Ant buildfile. You invoke these tools from the command line.

xdbuild syntax Here is the command syntax for xdbuild:

```
xdbuild projectFile target options
```

where:

Argument	Description
projectFile	Path to the 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.
target	Specify one of these project buildfile targets: ◆ **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 development environment)
options	See below for information on the options.

xdant syntax Here is the command syntax for xdant:

```
xdant 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 xdbuild and xdant:

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	(xdant only) Prints logging information without adornments.
-logfile *file*	Sends output to *file* instead of to the screen. This option creates *file* if it doesn't exist and overwrites *file* if it does exist.
-logger *class*	Specifies the class to do the logging. The default logger is org.apache.tools.ant.DefaultLogger. You can also specify another built-in logging class (look in ant.jar in the Novell exteNd Common\lib directory for provided classes) or specify a logging class you wrote yourself. 📖 See the Ant documentation at ant.apache.org for details.

Option	Description
-listener *class*	Adds *class* as a listener. A listener is notified when one of the following events occur: ◆ 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 There is no default listener. You can specify a built-in listener class (look in ant.jar in the Novell exteNd Common\lib directory for provided classes) or specify a listener class you wrote yourself. 📖 See the Ant documentation at ant.apache.org for details.
-D*property=value*	Overrides property value set in the buildfile. Properties are defined as <property> elements in the buildfile.
-buildfile *file*	(xdant only) Specifies the buildfile to use. If this option is not specified, Ant uses build.xml in the current directory. (This option applies only to xdant, because xdbuild always uses the project buildfile that the exteNd Director development environment creates for you automatically.)
-find *file*	(xdant only) Searches for buildfile *file* 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 *file*. If *file* is not specified, it searches for build.xml.

Examples

xdbuild examples The following command builds and creates the archive(s) for the exteNd Director project myApp (if changes have been made since the last time the project was built and archived):

```
xdbuild myApp.spf build
```

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

```
xdbuild myApp.spf rebuild
```

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

```
xdbuild myApp.spf clean
```

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

```
xdant
```

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

```
xdant 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:

```
xdant purge -find test.xml
```

Internationalization support

This section describes support for internationalization in the exteNd Director development environment.

Specifying fonts

If some international characters are not displaying correctly (for example, they are displaying as boxes) or if the font mapping on your system is poor, you can specify different fonts for the exteNd Director development environment 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 the development environment:**

1. Select **Tools>Preferences** and click the **Display** tab.
2. Turn on the setting **Use native look and feel**.
3. Exit the exteNd Director development environment.
4. Specify alternate font names (and optionally sizes and colors) in the following lines of **ide.props**, which is in the Novell exteNd tools\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 gray background.

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

Extending the development environment

The exteNd Director development environment can be extended (via an advanced **extensibility API**) to add custom tools and facilities. To learn about obtaining and using the extensibility API for the development environment, contact your Novell exteNd representative.

2 Projects and Archives

Your work in the Novell exteNd Director development environment is organized into **projects** for creating the J2EE and other **archives** that make up an exteNd Director application. 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 the basics:

About projects and archives

A **project** is a collection of source files that you work with in the exteNd Director development environment 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 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 (nonbuildable) archive

You aren't limited 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) in the exteNd Director development environment.

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, XHTML or HTML pages, XML files, 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 **project file** (SPF file) that defines it. This file is automatically created to store settings that you specify in the exteNd Director development environment. 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 the development environment. If you manually change the file incorrectly, you may compromise your ability to open the project.*

Organizing projects

When you create a project, you must specify which 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

Design decisions affect how you create the projects, subprojects, and components that make up your application.

The exteNd Director development environment supports almost any method for creating projects and components, including **bottom-up** (creating components first and then 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 50. For information about creating a project that contains existing source files and components, see "Working with existing source files" on page 56.

📖 For project considerations specific to exteNd Director applications, see the part on working with projects in *Developing exteNd Director Applications*.

Project directory structure considerations

The exteNd Director development environment 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 projects using a single or a combined source tree.

For more information on project settings, see "Managing project content settings" on page 66.

Directory structure of an archive The internal directory structure of a 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 J2EE 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.

📖 To learn more about specifying archive directory structures and packaging archives for J2EE, see the Sun J2EE Blueprints document.

📖 For information about how you can specify source and archive directory structures in the exteNd Director development environment, see "Managing project content settings" on page 66.

📖 For project considerations specific to exteNd Director applications, see the part on working with projects in *Developing exteNd Director Applications*.

Creating projects and subprojects

The projects (and subprojects) you can create fit into these categories:

- **exteNd Director projects**, which you typically use for the bulk of your application development work. These projects provide a wealth of functionality built in and ready to tailor to your needs.

  📖 To learn about creating exteNd Director projects, see the part on working with projects in *Developing exteNd Director Applications*.

- **Generic projects**, which you typically use when necessary to develop supporting modules. A generic project can be an EAR, EJB JAR, WAR, RAR, JAR, deploy-only archive, or application client.

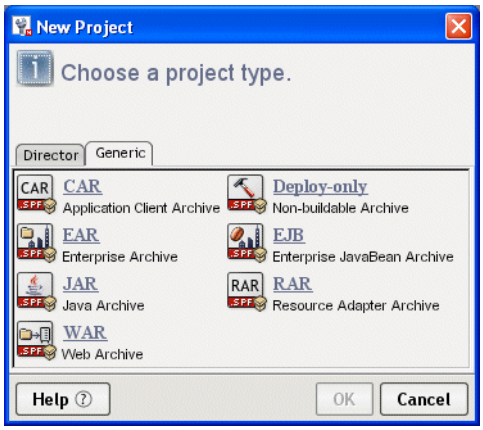
  As you create a generic project, you define a project name and a location for your source files. exteNd Director maps all of these source files to names and locations that you define for the archive. The following steps (using the New Project Wizard) apply to each type of generic project, with exceptions noted.

  📖 For information on organizing the development workspace before beginning a project, see "Project directory structure considerations" on page 48.

➢ **To create a generic project:**

NOTE: If you are creating a subproject, you must open the parent project in the exteNd Director development environment before starting this procedure.

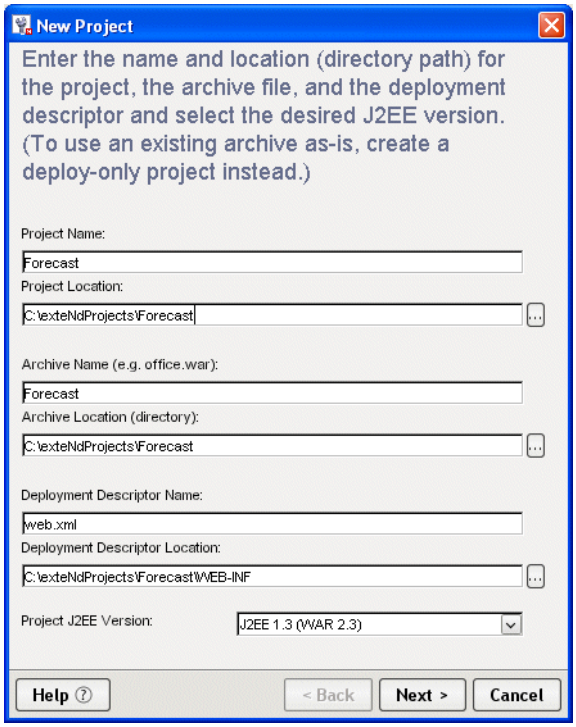
1 Select **File>New>Project**. The New Project dialog displays:



2 On the **Generic tab**, 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 55.
- 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 221.
- 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 56.

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



3 Specify project information as follows:

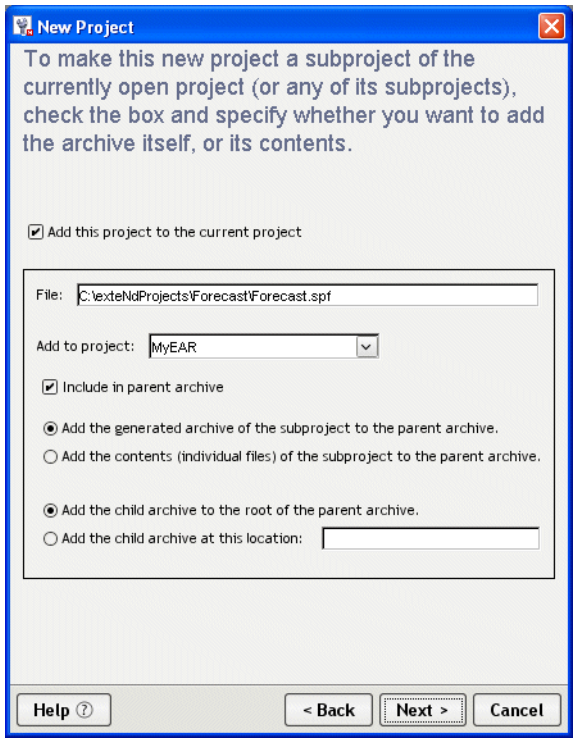
New project setting	What you do
Project Name	Specify the name you want to use for the project (the .SPF extension is automatically appended). This name appears in the source layout. As you enter a project name, the archive name is filled in automatically. You can keep the same name for your archive or enter another one.
Project Location	Specify the directory where you want the project (and other source files) to be located. exteNd Director creates a project file (with an .SPF extension) in the project location. As you enter a project location, the rest of the new project settings are filled in automatically. You can change these settings. You can click the ellipsis beside the Project Location field to select a location, or type the project directory. If you specify a project location directory that does not exist, the wizard prompts you to create it. If you do not specify an absolute path, the wizard locates the project under the Novell exteNd **tools\bin** directory.
Archive Name	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. 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. 📖 For information about creating projects based on existing archives, see “Working with existing source files” on page 56. For details about deploy-only archives, see “Creating a deploy-only project” on page 55.
Archive Location	Enter the location of the project archive or accept the default (the project root directory). The archive location appears in the archive layout of the Navigation Pane after the project has been created.

New project setting	What you do
Deployment Descriptor Name	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 source file. exteNd Director creates the default deployment descriptor name and location (based on archive type) when you build and archive the project. In most cases you should accept the default name and location. If you are converting an existing archive project (by creating a new project file), enter the name of the deployment descriptor file on disk. If you want to have multiple J2EE subprojects of the same type sharing the same deployment descriptor directory location, see Deployment Descriptor Location (below). The deployment descriptor name you enter here affects only the **source** file name—not the file name that is used in the JAR. When exteNd Director builds the archive, it includes this deployment descriptor file in the archive using the standard name and location defined by the J2EE specification for the archive type. 📖 For more information on deployment descriptor names, see Chapter 15, "Archive Deployment" and Chapter 16, "Deployment Descriptor Editor".
Deployment Descriptor Location	Enter the location of the deployment descriptor or accept the default. Each archive type uses a required J2EE default directory location. If you are converting an archive project, enter the location of its deployment descriptor. 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. If you specify (or if exteNd Director 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. 📖 For more information on deployment descriptor default names and locations, see Chapter 15, "Archive Deployment" and Chapter 16, "Deployment Descriptor Editor".
Project J2EE Version	Specify the version of J2EE for this project. 📖 For information on targeting your application at an appropriate version of J2EE, see Chapter 14, "How to Handle J2EE Versions".

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, 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, 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 varies, 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 of 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 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 58.

Creating a deploy-only project

You can 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 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 56.

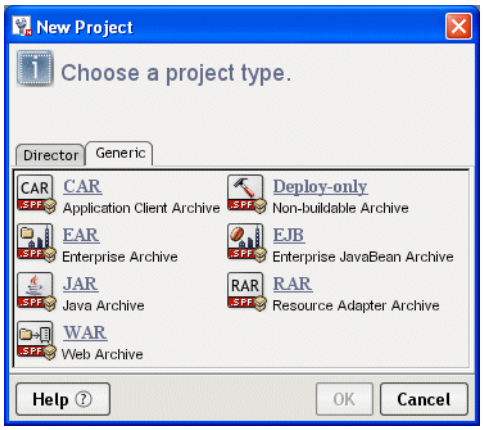
How you tell that a project is deploy-only When you open a deploy-only project:

- 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:

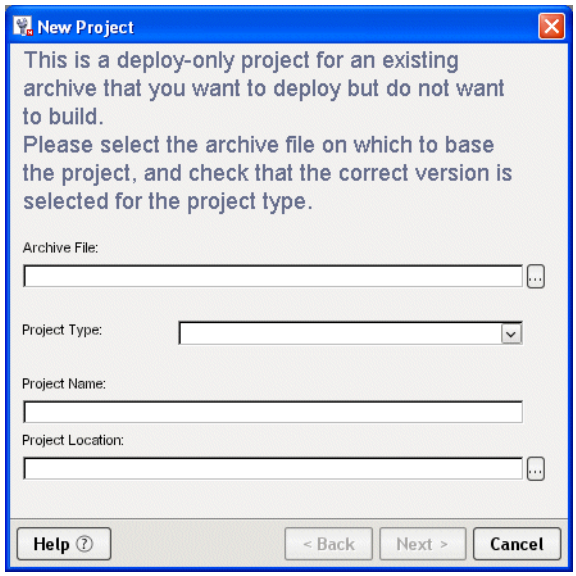
```
The archive is deploy only. Its contents cannot be modified or examined.
```

➢ **To create a deploy-only project:**

1 Select **File>New>Project**. The New Project dialog displays:



2 On the Generic tab, select **Deploy-only** and click **OK**.



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

Deploy-only project setting	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. exteNd Director creates a 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 of 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 the exteNd Director development environment:

- If you want to create a nonbuildable archive that you validate and deploy in the exteNd Director development environment, 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 55.

- If you have source code files and want to build an editable archive, you should create a regular 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 add source files to this type of archive later.

➢ **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 project, as described under "Creating projects and subprojects" on page 50.
3. Add the source directory you created in Step 1 to the project, as described under "Adding to projects" on page 58.

   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 populate it by doing either (or both) of the following:

- Creating new files in the project
- Adding existing files to the project

Creating source files

The exteNd Director development environment provides a wide selection of wizards that you can use to create new source files for your projects. These wizards prompt for information about what you want to generate and where (project and/or directory).

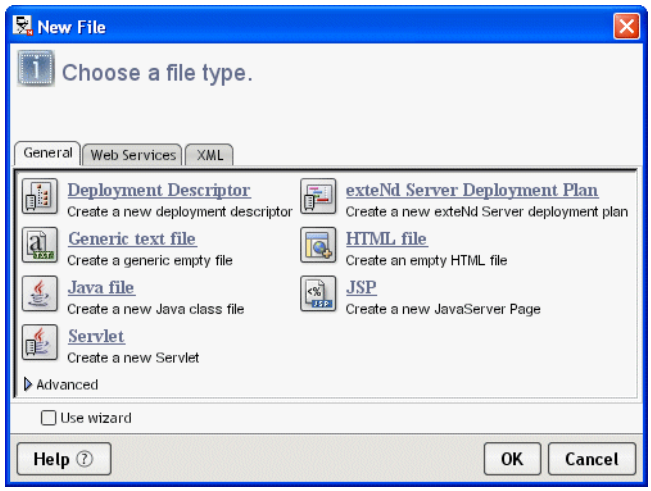
Source files include:

- **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, XHTML or HTML pages, XML files, images, and so on

➢ **To create a source file:**

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

The New File dialog displays.



3 Go to the **tab** that matches your need (note that the tabs you see depend on your exteNd product configuration).

4 Choose a file type and click **OK**. The wizard for that file type starts.

TIP: To sidestep the wizard and immediately open a new blank file in the appropriate source editor, deselect the **Use wizard** checkbox (when supported).

5 Follow the prompts onscreen. For more information:

To learn about	See
Wizards for basic XML and CSS files	Chapter 4, “XML Editors” Chapter 5, “XSL Editor” Chapter 6, “CSS Editor”
Wizards for basic Web Service files	Chapter 10, “Web Service Wizard” Chapter 11, “WSDL Editor”
Wizards for basic J2EE and Java files	Chapter 13, “J2EE Wizards” Chapter 16, “Deployment Descriptor Editor” Chapter 17, “Deployment Plan Editor”
Wizards for exteNd Director features	The appropriate subsystem guide in the exteNd Director help

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

For information about the source editors (text, Java, etc.), see Chapter 3, “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**.

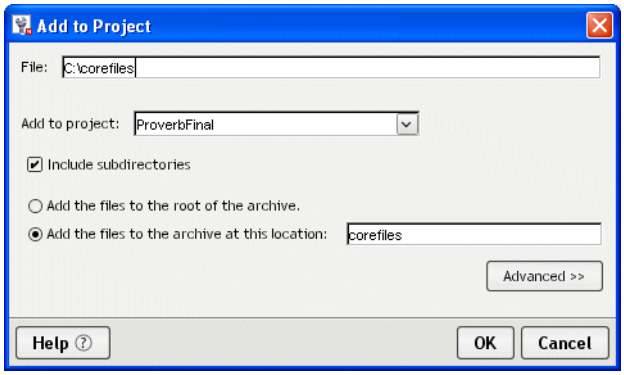
📖 For other methods, see "Other ways to add files and directories to a project" on page 60.

3 Choose whether you want to add a file or a directory.

Typically, you add directories rather than individual files to your project.

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	Select to add the specified item to a specified location other than the archive root. 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 66 and "Using relative paths" on page 67.

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 66.

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 When you make changes in the exteNd Director development environment, the Directory and Project tabs of the Navigation Pane automatically update. If you make changes outside of the development environment, 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 58), 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 automatically When you add the entire directory to a project, exteNd Director 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.

Choosing other files to exclude After you add a directory to a project, you can manually exclude individual files that you don't want to participate in the project.

📖 For more information, see "Excluding individual files from a project directory" on page 69.

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 50.

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

1 Open the project you want to add to.

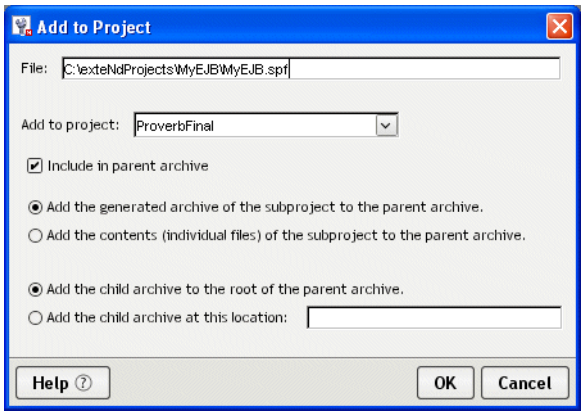
2 Select **Project>Add to Project>Subproject**.

📖 For other methods, see "Other ways to add files and directories to a project" on page 60.

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 varies, 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	Select (and enter a location) to add the specified archive to a location other than the root directory of the parent archive. 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 66 and "Using relative paths" on page 67.

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.
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 66 and "Using relative paths" on page 67.

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 67.

Viewing projects

You use the **Project tab** of 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

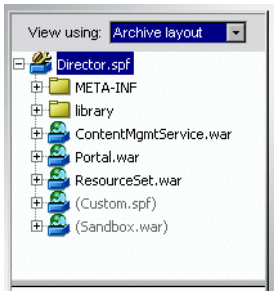
Source layout view The source layout view reflects the organization of the project's files and directories on your hard disk. Subprojects are listed at the top level as folders.

- 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 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

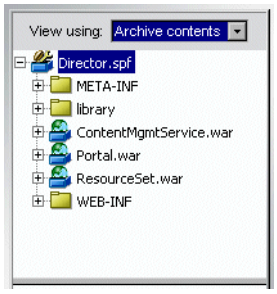
  In the following screen, the ResourceSet subproject has been added as an archive, so it displays with its archive name. The Custom subproject has been added to the project as contents, so it is grayed out. The Sandbox subproject has been excluded from the parent archive, so it too is grayed out.



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. These 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\MyEAR\web.xml).

Maintaining projects

Your open project may be your top-level project or it may be a subproject. The exteNd Director development environment 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
- Excluding individual files from a project directory
- Removing files, directories, and subprojects from projects
- Renaming a project

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 an application client provided they are all part of the same top-level EAR.

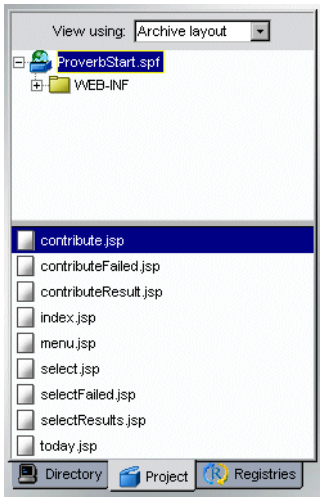
NOTE: Whenever you add a component or subproject, the project file is automatically saved. The only time you need to explicitly do a save is when you make changes to a source file using one of the editors.

➢ **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

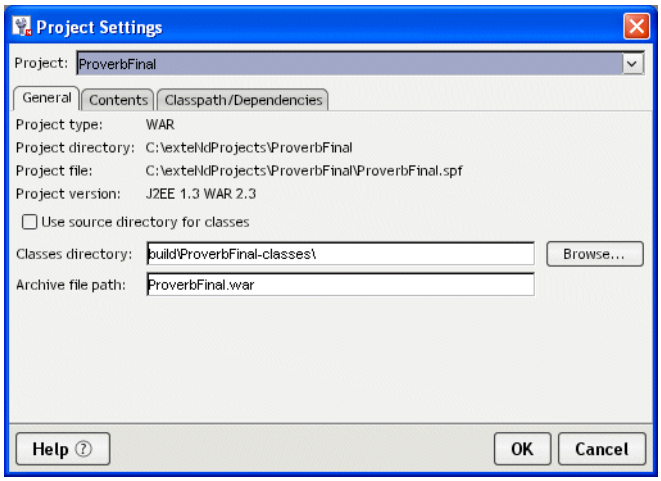
The General tab on the Project Settings dialog lets you view information about the open project and change the location of the source directory that stores the project class files.

➢ **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:

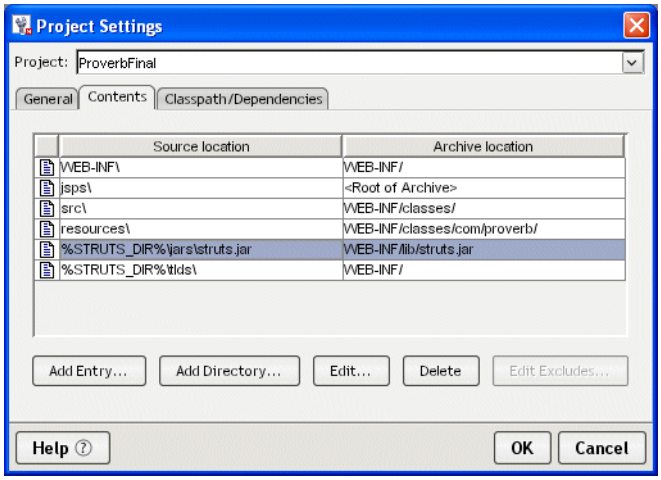


Setting	Description or action
Project	Lists the project currently open.
Project type	Lists the type of project you created.
Project directory	Lists the open project's root directory.
Project file	Lists the file name and location of the open project.
Project version	Lists the J2EE version.

Setting	Description or action
Use source directory for classes	Specifies whether you want to compile Java files into the same directory as their corresponding source files. 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). If you select the check box, all project classes are generated into the source directory along with with their source files.
Classes directory	Lists the root of the build directory where the project's compiled class files will be located. exteNd Director writes the generated classes to this directory when it builds the archive. 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.
Archive file path	Lists the name and location of the archive. exteNd Director writes the generated archive to this location, which is relative to the project root. You defined this location when you created the project.

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

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 for locating files that are not under the project's root directory.

You set environment variables in your operating system. You reference the variables in the exteNd Director development environment by using the following syntax: **%varname%** or **${varname}**. You can use 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 the 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 the development environment before the value of an environment variable (set in your operating system) 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 the exteNd Director development environment 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, use environment variables.

Modifying project entries

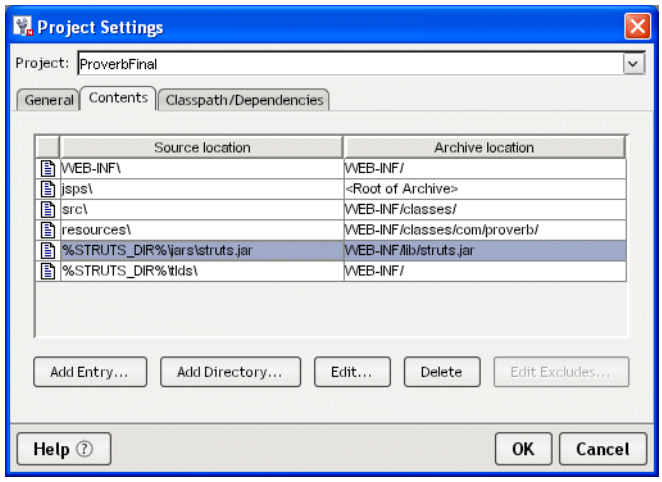
➢ **To modify project entries:**

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 the exteNd Director development environment. See "Renaming a project" on page 71.
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.



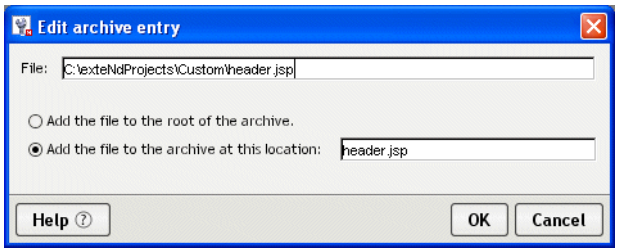
Setting	Description or action
Project	The project to modify.
Source location	The source location of the selected entry. A full path is listed whenever the source of the entry is not relative to the project root directory. Any files you later add to the source directory will also get included in the project archive. 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 66 and "Using relative paths" on page 67.
Archive location	The archive location of the contents of the selected entry. The archive location can be the same as or different from the source location. All archive locations are relative to the archive root directory. 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. An asterisk (*) indicates that you want to include all files in the specified directory, but not any nested subdirectories.
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.
Edit Excludes	See "Excluding individual files from a project directory" on page 69.

📖 For information on adding an entry or directory, see "Adding source files to a project" on page 58. For information on removing entries, see "Removing files, directories, and subprojects from projects" on page 70.

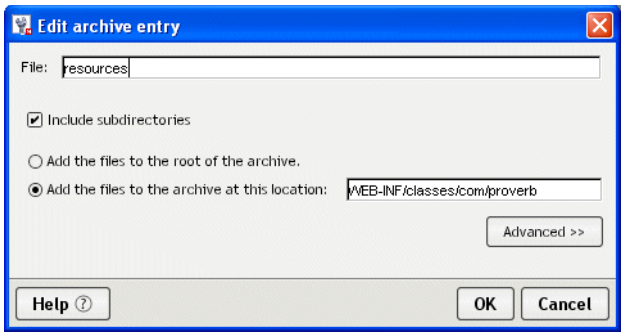
5 Select the project entry you want to modify by either double-clicking the entry or selecting the entry and clicking **Edit**.

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

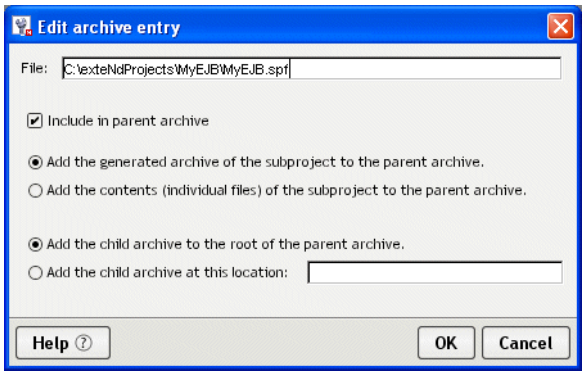
- The following dialog appears if you selected a **file**. The settings on this dialog are the same as on the Add to Project dialog. For more information, see "Adding to projects" on page 58.



- 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 58.



- 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 61.



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

Excluding individual files from a project directory

Although it's common practice to add entire directories to a project, you may sometimes have a few files in such directories that don't belong in the project. To handle this situation, you can exclude individual files from any directory you've added to a project. You can also include a file once again if you need it.

➢ **To exclude/include files:**

1 On the **Project tab** of the Navigation Pane, open a **directory** you've previously added.

2 Right-click a **file** you want to exclude, then select **Exclude from Project** from the popup menu.

Once a file is excluded, it is no longer used in the project. It appears gray (and in parentheses) on the Project tab.

To bring an excluded file back into the project, you can right-click it and select **Include in Project** from the popup menu.

Alternatively, you can exclude/include files by editing the **exclusions list** for a directory in the Project Settings dialog:

➢ **To edit the exclusions list:**

1 Choose **Project>Project Settings**.

2 Select the **Contents** tab.

3 Select the **directory** entry whose exclusions list you want to edit, then click the **Edit Excludes** button.

4 When the Edit Exclude Definitions dialog displays, you can do the following:

If you want to	Do this
Add a file to the list (to exclude it from the project)	Click **Add**, then choose that file in the target directory.
Remove a file from the list (to include it in the project once again)	Select that file from the list, then click **Delete**.

5 Click **OK** to close the Edit Exclude Definitions dialog.

6 Click **OK** to close the Project Settings dialog.

Removing files, directories, and subprojects from projects

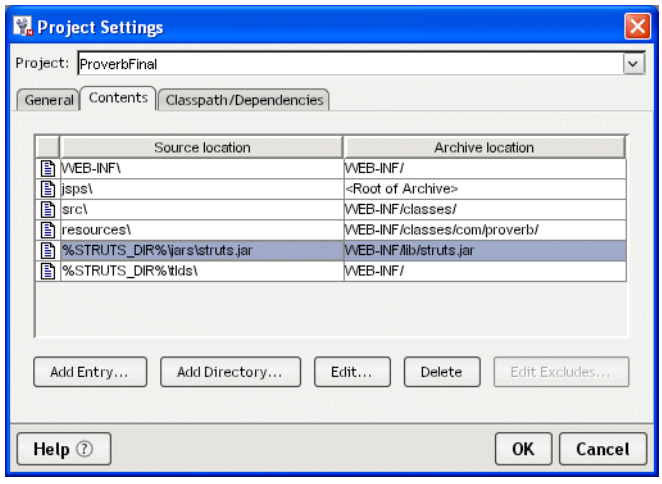
There are two ways to remove items from a project: using the **Project Settings** dialog or using the **Remove From Project** popup menu in the Project tab. Removing a project's source files or directories in the exteNd Director development environment does not delete them from your hard disk. It just 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**, exteNd Director removes the entry or entries so that 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), exteNd Director simply removes the entry so that it is no longer referred to in the project.
- When you remove a directory that was added to a project as part of nested subdirectories, exteNd Director prompts you to confirm that you want to remove the whole tree from the project.
- You can remove a subproject by selecting it in the top part of the Navigation Pane (one subproject at a time).

You get a list of the directory trees to be removed. For example, if you select to remove **src\a\b\c**, you are prompted that this will cause the **src** directory tree entry to be removed; if you confirm that this is OK, exteNd Director 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 in general you never directly edit a project file, you must do so in this situation.

➢ **To rename a project:**

1 Using your operating system, 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.

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

Compiling, building, and archiving

The exteNd Director development environment 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:

- Specifying build settings
- Using the commands

Specifying build settings

Specifying settings for your compiles and builds includes:

- Defining the Java compiler
- Defining the project classpath

Defining the Java compiler

By default, the exteNd Director development environment 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 26.

Defining the project classpath

The project classpath defines where to 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 66.

exteNd Director constructs the project classpath using these values:

Item	Description
Defaults	By default, the exteNd Director development environment uses: • The standard JDK default classpath (for all projects) • The JAR file that provides the J2EE API packages needed for compiling J2EE projects. For J2EE 1.2 projects, the file is j2ee_api_1_2.jar; for J2EE 1.3 projects, the file is j2ee_api_1_3.jar. If the J2EE API JAR file is accidentally removed from the classpath, you can find it in the Novell exteNd **tools\compilelib** directory.
Project contents	The Contents tab of the Project Settings dialog lists the components that you've added to a project. exteNd Director adds these items to the project's classpath in the order you added them to the project.
Classpath	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. 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: • If you put the project file on the classpath, exteNd Director 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, exteNd Director 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, exteNd Director 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, exteNd Director builds the parent project using the following:

2a The parent project's contents

2b The parent project's classpath

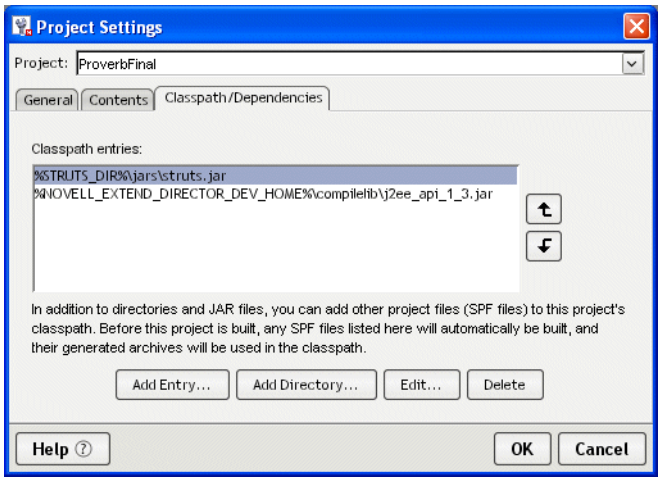
2c The referenced project's classpaths (which are constructed following the same rules—the contents, the classpath, and any referenced projects)

Suppose you have an EAR project, and the EAR contains a WAR (a subproject), and the WAR contains a utility JAR. exteNd Director 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 66).

If adding a directory, type the directory (specifying environment variables if desired) or click **Browse** and select the directory.

5 Click **OK**.

6 Repeat Step 3 and Step 4 for any other required items.

7 (Optional) To edit a single classpath entry:

7a Select the entry.

7b Click **Edit** and modify the entry in the dialog.

7c 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 **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	Compiles the currently open Java file. (Does not perform checking for interdependencies between the currently open file and other files in the project and its subprojects.) **NOTE:** **Compile** is not available in the popup menu that appears when you right-click a project file in the Navigation Pane.
Build	**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 24. **3** Writes the generated class files to the locations specified in the Project Settings dialog.
Rebuild All	**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 24. **3** Writes the generated class files to the locations specified in the Project Settings dialog.
Build and Archive	**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	**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**.
  exteNd Director compiles the Java file and writes the compile messages to the Output tab of the Output Pane.

➢ **To build a project:**

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

➢ **To create an archive:**

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

Building from the command line

You can use a command-line tool (xdbuild) to build projects outside of the exteNd Director development environment.

➢ **To build a project from the command line:**

1 Open a command window.

2 Make current the Novell exteNd **tools\bin** directory (it contains xdbuild).

3 Issue the following command:

```
xdbuild projectFile operation
```

where:

Argument	Description
projectFile	Path to the project (.SPF) file for the project you want to build
operation	One of the following: ◆ **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 development environment)

NOTE: xdbuild displays messages while it processes the project.

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

```
xdbuild c:\myProjects\myApp\myApp.spf build
```

NOTE: The development environment and xdbuild use Apache Ant to do the build processing. For more information on using Ant—including additional command-line options you can provide with xdbuild and how to use Ant to do your own customized processing—see "Using Ant" on page 43.

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, exteNd Director:

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 output messages.

Validation output Validate Archive writes information to:

- The Output tab of the Output Pane
- The *projectname*-chk.txt file (located in the project root directory)

In the Output tab of the Output Pane, you can double-click the line containing the string ***projectname*-chk.txt** to open that file in the Text Editor. The *projectname*-chk.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 **Output tab** of the Output Pane.
3. If there are validation errors, double-click the following text in the Output Pane:

   ```
   Look in file "projectdir\projectname-chk.txt" for detailed results on test
   assertions.
   ```

   The *projectname*-chk.txt file opens.

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

3 Source Editors

The Novell exteNd Director development environment provides two sets of editors, one set based on open-source NetBeans editors and the other set native to the development environment:

In this set	The editors are
NetBeans-based editors	Java Editor
	JSP Editor
	HTML Editor
	XML-related editors and CSS Editor
Native editors	Text Editor
	Text views of the Deployment Descriptor Editor and Deployment Plan Editor
	Nondefault 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 4, “XML Editors”
- Chapter 5, “XSL Editor”
- Chapter 6, “CSS Editor”
- Chapter 11, “WSDL Editor”
- Chapter 16, “Deployment Descriptor Editor”
- Chapter 17, “Deployment Plan Editor”

Common features

This section describes features common to all of the editors:

- Standard editing features
- Editor preferences
- Using text abbreviations
- Changing case
- Changing spaces, tabs, and indentation
- Searching across multiple files

- Regular expressions for text searches

Standard editing features

All of the editors provide these text-editing features:

To perform this function	Use this menu item
Cut, copy, and paste	Under **Edit**: • Cut (or Ctrl+X) • Copy (or Ctrl+C) • Paste (or Ctrl+V)
Undo and redo	Under **Edit**: • Undo (or Ctrl+Z) • Redo (or Ctrl+Y)
Select all text	Under **Edit**: • Select All (or Ctrl+A)
Toggle the display of line numbers	Under **View**: • Line Numbers (or Ctrl+L)
Find and replace	Under **Edit**: • Find (or Ctrl+F) • Find Next (or F3) • Replace (or Ctrl+R) **NOTE:** You can also search several files at once. See "Searching across multiple files" on page 79. **NOTE:** The native editors provide support for regular expression search. See "Regular expressions for text searches" on page 80.
Move cursor to a line	Under **Edit**: • Go To Line (or Ctrl+G)

Editor preferences

Much of the editor display and behavior can be configured in the Editing tab of the Preferences dialog, which you can access using **Tools>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 "Editing preferences" on page 27.

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. 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 the exteNd Director development environment appear on the Editing tab of the Preferences dialog (**Tools>Preferences**). From this dialog you can define new abbreviations and change or delete existing abbreviations. For details, see "Abbreviation preferences" on page 27.

Once you have defined an abbreviation, you can replace its name with the associated expanded text by pressing **Ctrl+U** (or by right-clicking and selecting **Text Tools>Complete Abbreviation**).

Changing case

You can easily change the case of text.

To perform this function	Do this
Change case of word containing insertion point	Right-click and select: ◆ Text Tools>**To Uppercase** ◆ Text Tools>**To Lowercase**

Changing spaces, tabs, and indentation

You can manipulate spaces, tabs, and indentation in text.

To perform this function	Do this
Change spaces to tabs or tabs to spaces	Right-click and select: ◆ Text Tools>**Spaces to Tabs** ◆ Text Tools>**Tabs to Spaces** If you select text before choosing these menu items, only that text is affected; if nothing is selected, the entire file is affected
Remove trailing whitespace	Right-click and select: ◆ Text Tools>**Remove Trailing Whitespace** If nothing is selected, this action works on the current line; otherwise, it acts on all selected lines
Change the indentation level	Right-click and select: ◆ Text Tools>**Shift Right** ◆ Text Tools>**Shift Left** You must select text on at least one line before you can select either of these menu items

Searching across multiple files

You can search across multiple files at once. You can search through:

- All files open in the exteNd Director development environment
- Files in all open projects
- Files in a specified open project
- Specified files on the file system

➢ **To search across multiple files:**

1 Select **Tools>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 "Regular expressions for text searches" below.

3 Click **OK**.

exteNd Director 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 **Tools>Next Occurrence** (or press **F4**)
- Select **Tools>Previous Occurrence** (or press **Shift+F4**)

Regular expressions for text searches

Sometimes searching for a literal string is too limiting. For example, you may want to search for a word starting at the beginning of a line or two words separated by any number of spaces. The text-based editors in the exteNd Director development environment support the use of regular expressions—patterns for describing string matching—to augment the usual search capabilities. Regular-expression search is available in the **Tools>Find in Files** dialog and in the **Find** dialog invoked from a native editor.

This section provides the following information:

- Using regular expressions in search operations
- Regular-expression reference

Using regular expressions in search operations

In addition to allowing you to type regular-expression syntax directly into the **Search for** text box, the Find dialog has a **regular expression** helper menu you can use in constructing regular-expression searches.

Selecting a helper menu item appends to the expression one or more characters that make up a syntactical building block. For most regular-expression searches, you will need to use several of these syntactical building blocks in combination with text you type directly into the **Search for** text box.

📖 For more information on the regular expression helper menu items, see the "Regular-expression reference" on page 81.

NOTE: When doing regular-expression searches, you will not be able to use all the Find dialog search options. For example, the Match whole word option becomes meaningless, since this choice is made within the regular expression itself.

➢ **To use regular expressions in a search operation:**

1 Select **Tools>Find in Files** or select **Edit>Find** in a native editor.

The Find dialog displays.

2 Select the **Regular Expression** check box.

OR

Click the right arrow to the right of the **Search for** text box and make a selection from the regular expression helper menu.

3 Type a regular expression in the **Search for** text box, or use a combination of literal text and selections from the regular expression helper menu to construct your regular expression.

For example:

To match	Enter
getText or **setText**	**[gs]etText**
void followed by **main**, with any amount of white space between the two words	**void\s+main**

4 Click **OK** to begin the search.

Text matching the search criteria appears highlighted in the editor.

Regular-expression reference

The tables in this section explain the syntactical building blocks of regular expressions supported in the exteNd Director development environment. Many of these building blocks are available on the regular expression helper menu.

There are several categories of building blocks for regular expressions:

- Characters
- Character classes
- Standard POSIX character classes
- Nonstandard POSIX-style character classes
- Predefined classes
- Boundary matchers
- Closure operators
- Backreferences

Characters

Syntax	Description
unicodeChar	Matches any identical unicode character
\	Used to quote a metacharacter (like *)
\\	Matches a single slash (\) character
\0nnn	Matches a given octal character
\xhh	Matches a given 8-bit hexadecimal character
\\uhhhh	Matches a given 16-bit hexadecimal character
\t	Matches an ASCII tab character

Syntax	Description
\n	Matches an ASCII newline character
\r	Matches an ASCII return character
\f	Matches an ASCII formfeed character

Character classes

Syntax	Description
[abc]	Simple character class
[a-zA-Z]	Character class with ranges
[^abc]	Negated character class

Standard POSIX character classes

Syntax	Description
[:alnum:]	Alphanumeric characters
[:alpha:]	Alphabetic characters
[:cntrl:]	Control characters
[:digit:]	Numeric characters
[:graph:]	Characters that are both printable and visible (for example, a space is printable but not visible, but an **a** is both printable and visible)
[:lower:]	Lowercase alphabetic characters
[:print:]	Printable characters (characters that are not control characters)
[:punct:]	Punctuation characters (characters that are not letters, digits, control characters, or space characters)
[:space:]	Space characters (such as space, tab, and formfeed)
[:upper:]	Uppercase alphabetic characters

Nonstandard POSIX-style character classes

Syntax	Description
[:javastart:]	Start of a Java identifier
[:javapart:]	Part of a Java identifier

Predefined classes

Syntax	Description
.	Matches any character other than newline

Syntax	Description
\w	Matches a word character (alphanumeric plus "_")
\W	Matches a nonword character
\s	Matches a whitespace character
\S	Matches a nonwhitespace character
\d	Matches a digit character
\D	Matches a nondigit character

Boundary matchers

Syntax	Description
^	Matches only at the beginning of a line
$	Matches only at the end of a line
\b	Matches only at a word boundary
\B	Matches only at a nonword boundary

Closure operators

All closure operators (+, *, ?, {m,n}) are by default **greedy**, meaning that they match as many elements of the string as possible without causing the overall match to fail.

Syntax	Description
A*	Matches A zero or more times
A+	Matches A one or more times
A?	Matches A zero or one time
A{n}	Matches A exactly n times
A{n,}	Matches A at least n times
A{n,m}	Matches A at least n but not more than m times

Backreferences

You can refer to the contents of a parenthesized expression within a regular expression itself using a **backreference**. The first backreference in a regular expression is denoted by \1, the second by \2, and so on. For example, the expression:

```
([0-9]+)=\1
```

will match any string of the form n=n (like 0=0 or 2=2).

The NetBeans-based editors

The core editors in the exteNd Director development environment are based on NetBeans, an open-source Java-based framework and set of editors. These editors are:

- Java Editor
- JSP Editor
- HTML Editor
- XML-related editors and CSS Editor (these editors have some different features than the other NetBeans-based editors; see Chapter 4, “XML Editors”, Chapter 5, “XSL Editor”, Chapter 6, “CSS Editor”, and Chapter 11, “WSDL Editor”)

NOTE: Previous releases of the exteNd Director development environment used native versions of the Java, JSP, and HTML editors. The native versions are still provided and you can configure the development environment to use them instead of the NetBeans versions. See “Using the native Java, JSP, or HTML editor” on page 89.

The following sections describe the NetBeans-based editors:

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 code Java in the Java Editor (or on a JSP page in the JSP Editor), you can use the **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 **parser database files**. The exteNd Director development environment ships with predefined parser files that include the following classes:

- JDK 1.4
- Ant 1.5
- Novell exteNd Web Services SDK (com.sssw.jbroker.web packages)

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 86.

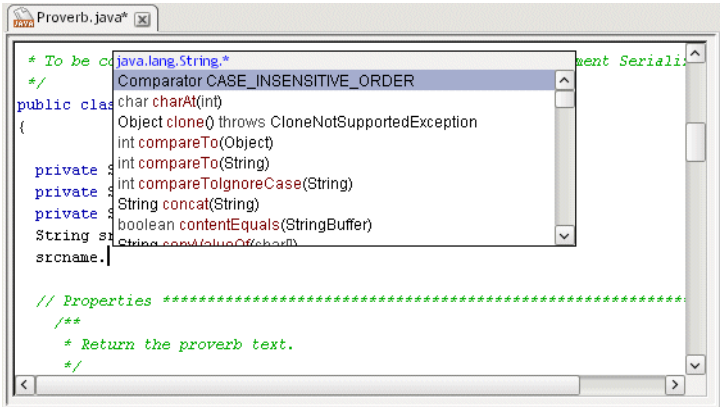
➢ **To complete a Java expression:**

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, 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
- 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 up to the point of 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.

The code completion box displays the data type of the entered arguments in its title; 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 **parser database files**. You can have the exteNd Director development environment 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 **Tools>Preferences**.

4 Select the **Editing** tab and expand the **Code Completion** section.

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. exteNd Director will create two files: *prefix*.jcb and *prefix*.jcs.

Setting	Description
Java source file directory	The root of the directory containing your project's **source** files, such as c:\myProject\src
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, the exteNd Director development environment 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 **Tools>Preferences**.

2 Select the **Editing** tab and expand the **Editor Associations** section.

This lists the NetBeans kits that are installed and allows you to specify which you want to use.

The list at the bottom 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 you get the corresponding native editor). For details, see "Using the native Java, JSP, or HTML editor" on page 89.

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 editors in the exteNd Director development environment are based on NetBeans and have the features described above. The following editors are native to the development environment and have a different feature set:

- Text Editor
- Text views of the Deployment Descriptor Editor and Deployment Plan 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 right-click in the editor and select:

- **Text Tools>Convert to UNIX Line Endings** to convert all DOS-style line ending characters to UNIX-style line ending characters
- **Text Tools>Convert to DOS Line Endings** to convert all UNIX-style line ending characters to DOS-style line ending characters

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, the bottom-right corner of the exteNd Director development environment displays 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, the exteNd Director development environment uses NetBeans-based Java, JSP, and HTML editors (see "The NetBeans-based editors" on page 84). 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 90).

➢ **To use the native Java, JSP, or HTML editors:**

1. Select **Tools>Preferences**.
2. Select the **Editing** tab and expand the **Editor Associations** section.

   This lists the NetBeans kits that are installed and allows you to specify which you want to use.

   The list at the bottom 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 want to 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

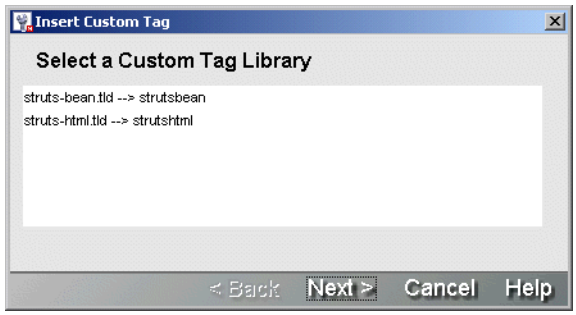
The native JSP Editor provides a Custom Tag Wizard that enables you to easily insert custom tags into a JSP page.

➢ **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 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**.

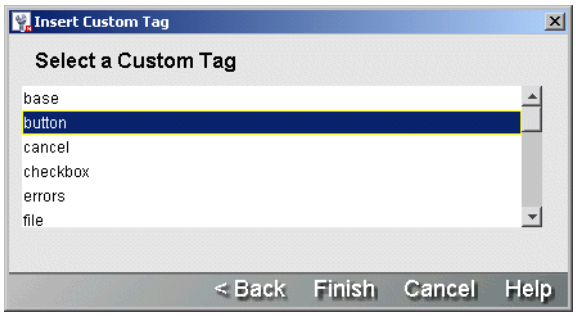
   NOTE: You must be using the native JSP Editor to access this wizard. See "Using the native Java, JSP, or HTML editor" on page 89.

   If the page has more than one **taglib** directive, a list of all tag libraries specified on the page displays. For example:

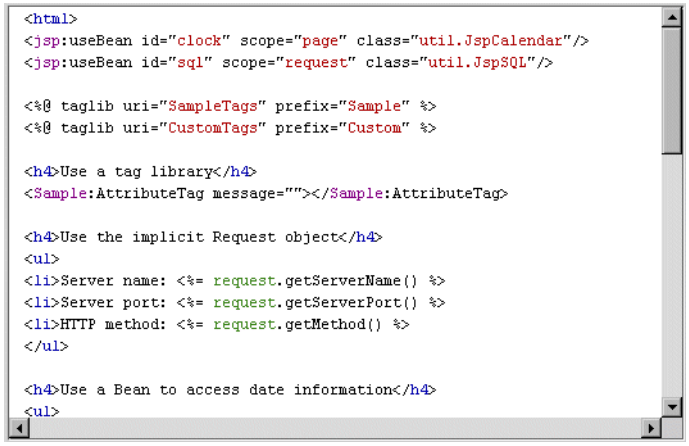


   Select the tag library you want to use and click **Next**.
6. If the wizard cannot find the tag library specified, it prompts you to locate that tag library on your file system.

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:



```
<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.

XML and CSS

These chapters present the exteNd Director utility tools for working with XML and CSS files:

- Chapter 4, “XML Editors”
- Chapter 5, “XSL Editor”
- Chapter 6, “CSS Editor”

4 XML Editors

This chapter describes the basic facilities that the Novell exteNd Director development environment provides to work with XML and XML-related files. It contains the following topics:

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 (J2EE deployment descriptors are XML documents).

The definition of an XML document is specified by either a **DTD** (Document Type Definition) 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 www.w3.org/XML.

TIP: If you are new to XML, you might want to read the XML FAQ at www.ucc.ie/xml. Among other topics, it describes the differences between Schemas and DTDs.

XML in the development environment

The exteNd Director development environment provides broad support for working with XML files, including:

- XML-related wizards
  - **XML File Wizard** to create an XML file
  - **XML Schema File Wizard** to create a Schema file
  - **DTD to Schema Wizard** to convert a DTD to a Schema
  - **XML Catalog File Wizard** to create a catalog entry file
  - **XSL File Wizard** to create an XSL file
- XML-related editors
  - **XML Editor**
  - **XML Schema Editor**
  - **XML Catalog Editor**
  - **XSL Editor**

You'll learn about these XML facilities in this chapter and in the "XSL Editor" chapter.

Using the XML Editor

The XML Editor enables you to:

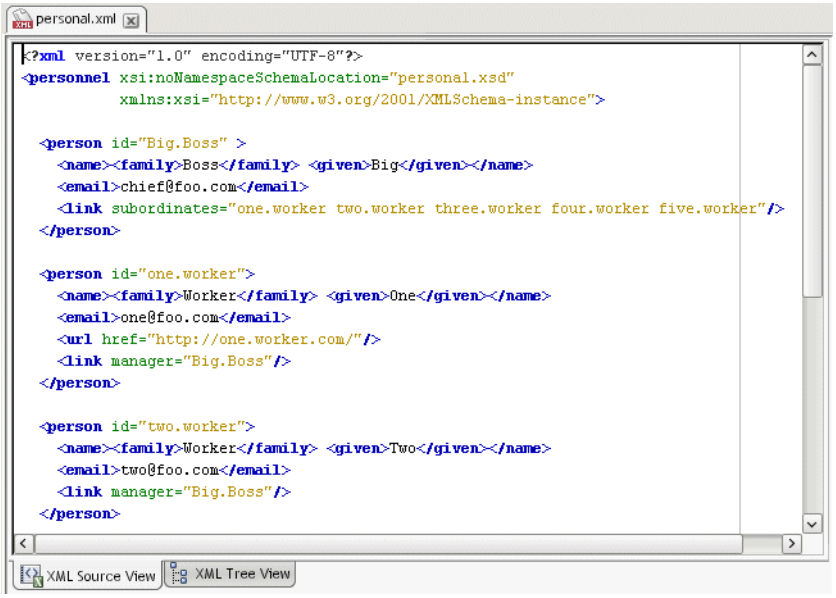
- **View and edit XML documents** in a syntax-colored Source View or a Tree View
- **Create and modify document elements** easily through the editor's context-based code completion and the Schema Guide window
- **Attach a Schema or DTD** to an XML document
- **Detach a Schema or DTD**
- **Validate an XML document** against a Schema or DTD, and check for well-formedness
- **Convert a DTD** to a Schema
- **Attach and edit a style sheet** for an XML document by using the CSS Style Manager

Using the Source View

The Source View is 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 106)
- Formatting of XML elements (see "Modifying text" on page 119 and "Changing indentation" on page 120)
- Navigating by XML elements (see "Moving the insertion point" on page 117)
- Finding matching tags (see "Moving the insertion point" on page 117)
- Bookmarks (see "Bookmarks" on page 120)
- Specifying colors to display different types of information (see "XML color preferences" on page 30)

The XML Editor displays the current XML document in Source View if you click the **XML Source View** tab.

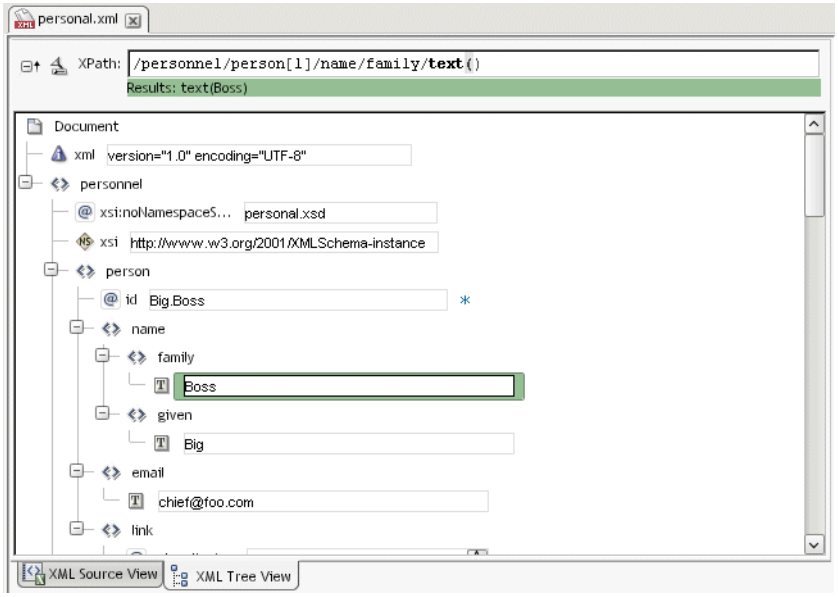


Using the Tree View

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 106)
- Cut-and-paste editing (see “Editing nodes” on page 110)
- Drag and drop (see “Editing nodes” on page 110)
- Searching by name, value, XPath, or text (see “Searching an XML document” on page 112)
- Finding matching elements (see “Navigation and display” on page 116)

The XML Editor displays the current XML document in Tree View if you click the **XML Tree View** tab.



Tree View controls

You can use these controls to manipulate and navigate the tree:

Control	Description
	Expand all nodes. You can also do this from the context menu; right-click a node and select **View>Expand All**.
	Collapse all nodes. You can also do this from the context menu; right-click a node and select **View>Collapse All**.
	Display the **Tree Filters** dialog to show or hide particular types of nodes in the tree. The default is to show all, but you can selectively show or hide any of the following: ◆ Attributes ◆ CDATA ◆ Comments ◆ Namespaces ◆ Processing Instructions ◆ Text You can also do this from the context menu; right-click a node and select **View>Filter**.
XPath	You can do either of the following: ◆ **Enter an XPath expression** in this control to navigate to a corresponding node in the tree ◆ **Highlight a node in the tree** to see its XPath expression in this control The XPath control in the XML Editor's Tree View is a subset of the **XPath Navigator** described in the chapter on working with scoped paths and XPaths in *Developing exteNd Director Applications*. See that chapter to learn about XPath expressions and how to use them.
	Expand node. You can also do this from the context menu; right-click a node and select **View>Expand Node**.
	Collapse node. You can also do this from the context menu; right-click a node and select **View>Collapse Node**.

Tree View icons

Here's an explanation of the icons that display in the tree:

Icon	Description
	Document
	Element
	Attribute
	CDATA
	Comment

Icon	Description
	Namespace declaration
	Processing instruction
	Text value of an element; for example, the text value of `<myTag>some text</myTag>` is **some text**
	Search result
*	Required attribute
	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>File**.

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 File Wizard, select **Use Wizard** and click **OK**. The XML File Wizard displays. Go through the wizard as follows.

4 Specify the **name** 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 XML 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 XML catalog; the corresponding file name is displayed in the File Name field
- Select a Schema or DTD directly from the file system by clicking the **browse** (...) button and selecting the file

📖 For more information about the XML catalog, see "Maintaining the XML catalog" on page 113.

6 If you want the wizard to generate some skeletal contents for your XML file based on the selected Schema or DTD, check the **Create XML from XML Schema or DTD** option; when you click **Next**, the wizard will prompt you to choose a root element for the document.

7 Click **Next**.

8 Specify the **location** of the XML file and click **Finish**.

The XML Editor displays the Schema Guide.

9 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 103.

➢ **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 or .TLD (for a tag library descriptor file). The selected file opens in the XML Editor and the XML Editor menu appears on the menu bar.

NOTE: Other kinds of XML files may open in specialized XML editors, such as the XML Catalog Editor or Deployment Descriptor Editor.

Finding Schemas and DTDs If the XML document specifies a Schema or DTD, the editor searches for it when opening the document. If the editor finds the Schema or DTD, it **attaches** it to the XML document. If the reference is unqualified, the editor first looks in the XML catalog; if the editor 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 a Schema or DTD.

📖 For more information, see "Associating Schemas and DTDs with XML documents" on page 100.

NOTE: The window title for an XML document specifies whether the document is attached to a Schema or DTD.

Working with Schemas and DTDs

To work with the Schemas and DTDs for your XML documents, you need to know about these topics:

- Associating Schemas and DTDs with XML documents
- Converting a DTD to a Schema
- Creating and editing Schemas
- Using the Schema Guide

Associating Schemas and DTDs with XML documents

In order to use context-sensitive code completion and to validate an XML document, an XML Schema (.XSD file) or a DTD (.DTD file) **must** be attached to the document. If the editor doesn't attach a Schema or DTD when opening your XML document, you can use the XML Editor menu to attach one, or you can manually edit the document to specify a Schema or DTD (and then refresh).

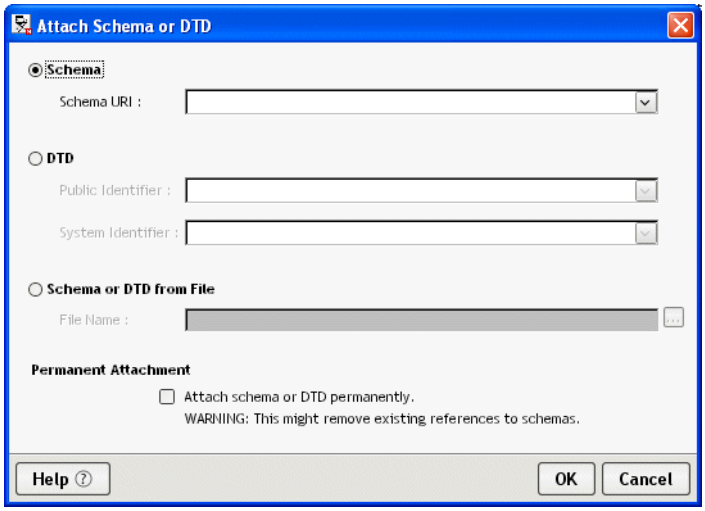
Attaching a Schema or DTD to a document

You can attach a Schema or DTD that is in the XML catalog or elsewhere on the file system to an open XML document. You can choose to attach the specified Schema or DTD either temporarily or permanently.

📖 For more information about the XML catalog, see "Maintaining the XML catalog" on page 113.

➢ **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 XML 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 XML catalog; the corresponding file name is displayed in the File Name field
- Select a Schema or DTD directly from the file system by clicking the **browse** (...) button and selecting the file

3 Set the **Permanent Attachment** property to make this association either temporary or permanent:

- If you **uncheck** Permanent Attachment, the Schema or DTD association is only for the purpose of context editing and validation in the current XML Editor session. The XML document is not modified.
- If you **check** Permanent Attachment, the XML document is modified to reflect the Schema or DTD association. Note that this may remove existing Schema references.

4 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.

Errors Any errors that occur when attaching a Schema or DTD are reported in the Output Pane.

Manually specifying a Schema or DTD in a document

Another way to permanently associate your document with a Schema or DTD is to manually edit the XML and then make the XML Editor aware of the association.

➢ **To make a manual association and update the editor:**

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 the editor to use the association by selecting **XML Editor>Refresh Schema Handler**.

The XML Editor 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 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 the editor 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 editors.

The exteNd Director development environment provides support for converting DTDs to Schemas. 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>File**.
2. On the **XML** tab, select **DTD to Schema** (in the Advanced section) and click **OK**.
3. Specify the **DTD** to convert. You can click the ellipsis 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 ellipsis button to browse the file system.
6. Specify whether you want the Schema **opened** in the Schema Editor after it is created.
7. Click **Finish**.

   This converts the DTD to a Schema, stores the Schema in the specified location, and displays the Schema in the Schema 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 Schema Editor and attach it to an XML document for context editing and validation.

Creating and editing Schemas

The exteNd Director development environment provides a wizard you can use when you need to create a new Schema. When you need to modify a Schema, you can use the **Schema Editor** (which is simply a more specialized version of the XML Editor).

➢ **To create a new Schema:**

1. Select **File>New>File**.
2. On the **XML** tab, select **XML Schema file**.
3. To create a new Schema using the wizard, make sure **Use Wizard** is selected then click **OK**.

   (Alternatively, you can deselect **Use Wizard** to skip the wizard and immediately get an empty, untitled Schema in the Schema Editor.)
4. When the **XML Schema File Wizard** displays, specify a name and location for the new Schema file then click **Finish**.

   The wizard creates that file and opens it in the Schema Editor (which includes a Schema Editor menu on the menu bar). You can now start developing your new, empty Schema by using either the Tree View or the Source View.

➢ **To edit a Schema:**

1. Select **File>Open**.
2. In the Open dialog, select the Schema (.XSD) file to edit and click **Open**.

   The file opens in the Schema Editor (which includes a Schema Editor menu on the menu bar). You can now manipulate this Schema as needed by using either the Tree View or the Source View.

Using the Schema Guide

The XML editors provide context editing functionality to help you work on XML documents. But context editing doesn't always supply all the information you might want. For example:

- It doesn't show exactly how a Schema (or DTD) is put together and **which 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 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 performed a full validation of the document.
- **If you're 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 had 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.

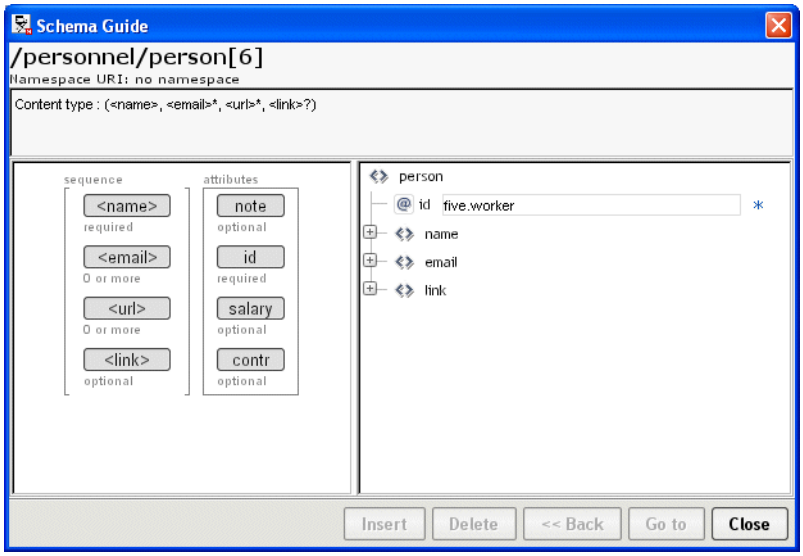
The Schema Guide addresses these situations.

➢ **To invoke the Schema Guide:**

1. Go to the **Tree View** in an XML editor.
2. Do one of the following:
   - Right-click an element whose contents you want to edit and select **Edit>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 the chapter on working with scoped paths and XPaths in *Developing exteNd Director Applications*.
- **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 sixth person element (/personnel/person[6]) 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 a horizontal bracket above and below
- **Sequence groups** are shown with a vertical bracket on the left and right
- **Attributes and All groups** are displayed in boxes

The Schema Guide displays how many instances of each subelement and attribute are allowed (such as **0 or more**, **required**, or **optional**). Positioning the mouse pointer over an element displays a tool tip describing that element (if there is documentation for it in the Schema or DTD).

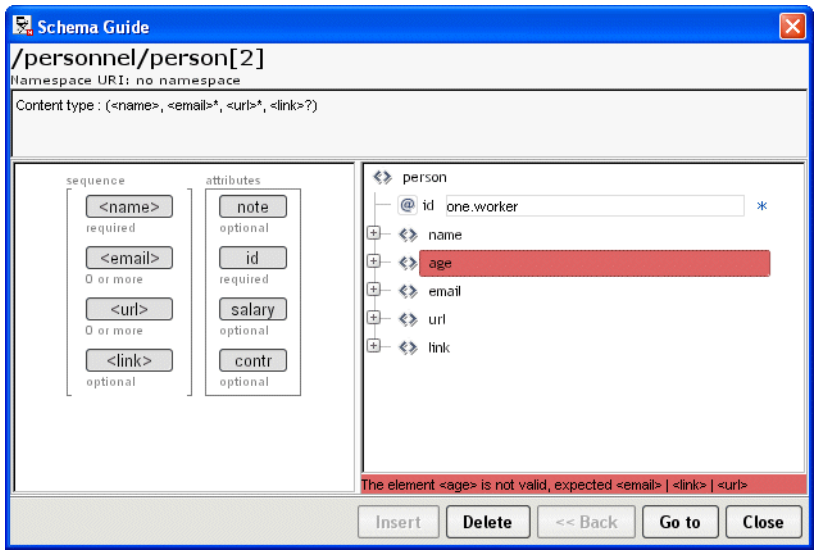
The Schema Guide is invoked automatically when you use the XML File Wizard to create an XML document. You can also invoke it when the document is empty and has a Schema attached. In this situation, the Schema Guide lists in the left pane possible **root elements**. If you are using a DTD, the description in the header will show the suggested root elements (that is, those elements not in the content model of other elements).

About the right pane

The right pane displays the standard **Tree View** 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 red. Clicking a colored element displays a similarly colored region of text along the bottom of the tree. The text describes the issue in more detail.

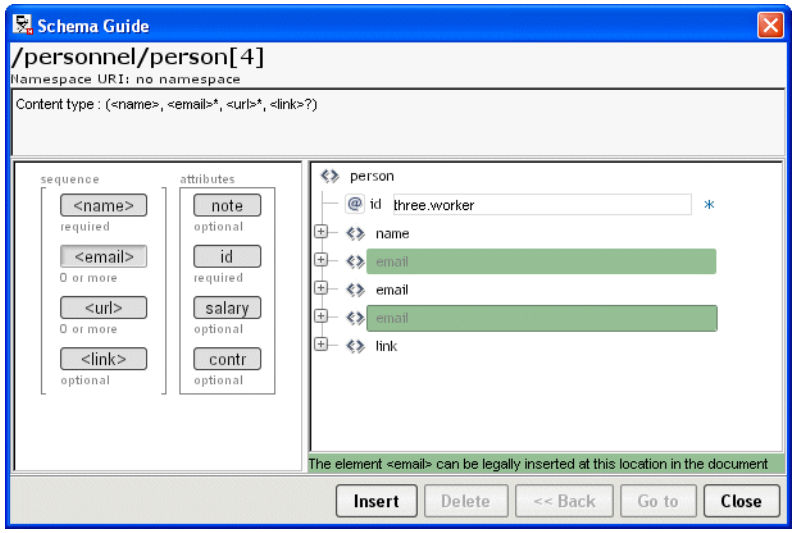
In many cases, the Schema Guide can **fix validation errors**, either by removing illegal elements or attributes, or by moving an element from a wrong namespace into a correct one. In the following example, the Schema Guide is indicating that the age element is invalid in the person element. You can 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 validation but whose namespace is incorrect is shown with a yellow background. You can edit the element to specify the correct namespace.

Adding elements and attributes

Elements When you select an element in the left pane, the right pane shows where that 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 nodes 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.

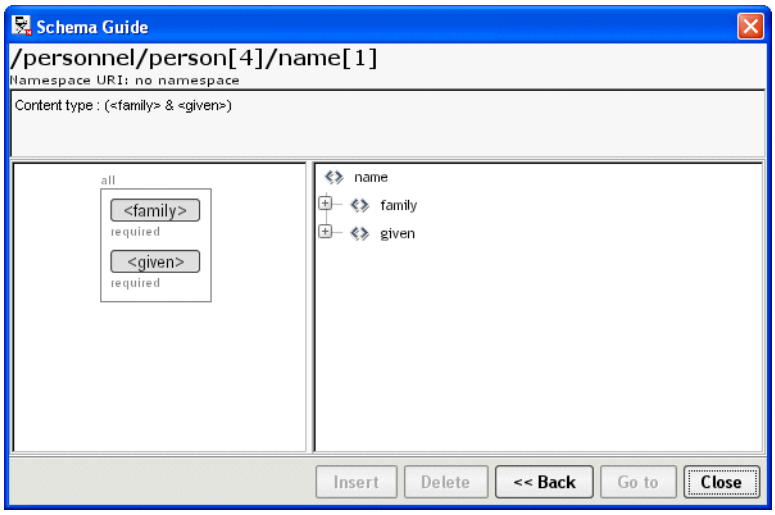
If you click an element in the left pane that cannot be legally inserted, you will not see any green nodes in the right pane.

Attributes To add an attribute, select it in the left pane. If it is legal to add, you will see a green node in the right pane. Click **Insert** and specify the attribute's value.

Looking at different elements

You can navigate the element hierarchy by selecting a subelement in the right pane and clicking **Go to**. That subelement becomes the selected element and its definition displays in the left pane; the tree structure for the selected instance displays 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 you select the person element's name subelement and click **Go to**.



Click **Back** to return to working with the parent element.

Editing an XML document

You can edit an XML document in either Tree View or Source View. This section describes the editing features you can use:

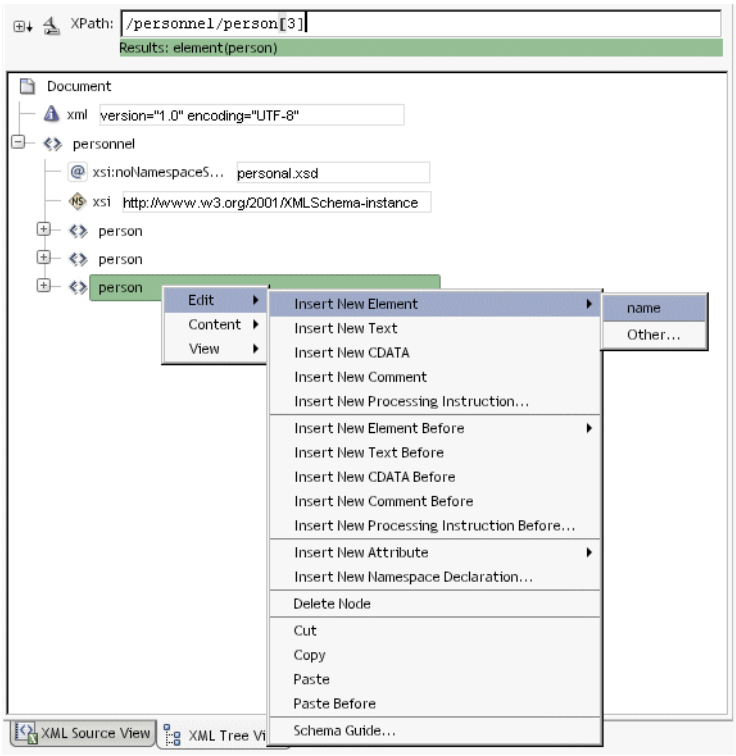
- About context support
- Adding elements
- Adding attributes
- Adding other nodes
- Editing nodes

About context support

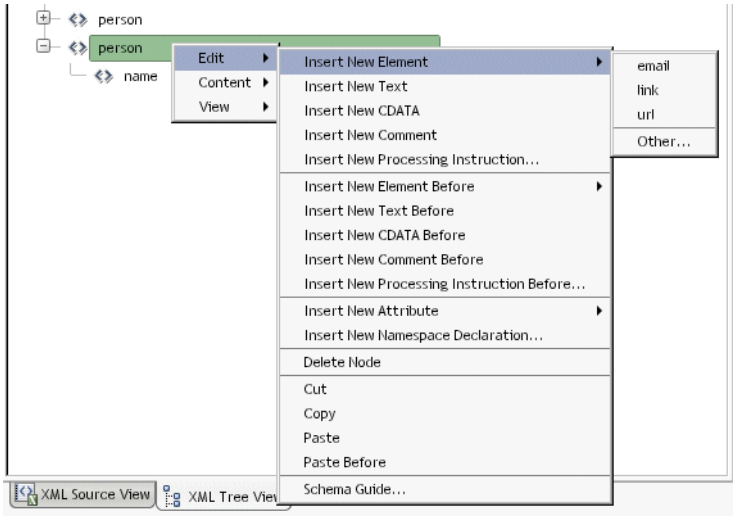
If you have attached a Schema or DTD, you can use the XML Editor's **context support**. The editor provides context support in both Tree View and Source View to help you edit your XML documents.

Context support in Tree View

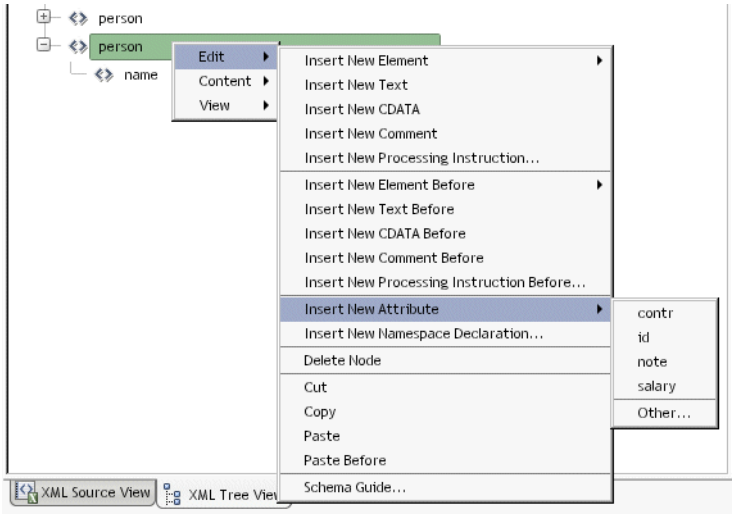
In Tree View, **right-click** at the appropriate location in the document. In the following illustration, a new person is being added to the document, and the XML Editor detects from the Schema that the next valid element is **name**.



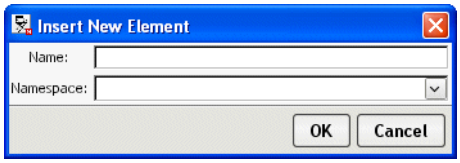
Once the name has been added, the XML Editor presents the new list of **valid elements**, according to the Schema.



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. It displays a dialog to fill in:



Using the Schema Guide In addition to using the Tree View's 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 103.

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). You can select the one you want to insert it. For example:

In the following example, a space is typed in an **url** element, which displays the valid attribute, **href**:

Adding elements

➢ **To add an element in Tree View:**

1. Select the location where you want to insert the element.
2. **Right-click** and select one of the following:
   - **Edit>Insert New Element** to insert an element inside the current element
   - **Edit>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; select the one you want. Alternatively, you can add an element yourself by choosing **Other**.

➢ **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; select the one you want. Alternatively, you can add an element yourself 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 **Edit>Insert New Attribute**.

   If valid attributes can be inferred from the definition of the document, they will be listed; select the one you want. Alternatively, you can add an attribute yourself by choosing **Other**.
3. Specify a **value** for the attribute, as appropriate.

➢ **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; select the one you want. Alternatively, you can add an attribute yourself by typing it.
3. Specify a **value** for the attribute, as appropriate.

Adding other nodes

➢ **To add other nodes in Tree View:**

1 Select the location where you want to insert the node.

2 **Right-click** and select one of the following from the **Edit** menu:

To add this	Select one of these
Text	**Insert New Text** **Insert New Text Before**
CDATA	**Insert New CDATA** **Insert New CDATA Before**
Comment	**Insert New Comment** **Insert New Comment Before**
Processing instruction	**Insert New Processing Instruction** **Insert New Processing Instruction Before**
Namespace declaration	**Insert New Namespace Declaration**

3 Specify the details for the node, as appropriate.

➢ **To add other code in Source View:**

- For entities other than elements and attributes, you must type the code yourself.

Editing nodes

➢ **To copy, move, or delete nodes in Tree View:**

- **Drag and drop** to move nodes
  OR
- **Right-click** and select one of the following from the **Edit** menu:
  - **Delete Node** to remove a selected node and its contents
  - **Cut** or **Copy** to place a node on the clipboard, then **Paste** to insert it as the last child of a selected node or **Paste Before** to insert it before a selected node

    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.

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 code 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** (**Ctrl+Z**) or redone by selecting **Edit>Redo** (**Ctrl+Y**).

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 a Schema or DTD) and check it for well-formedness.

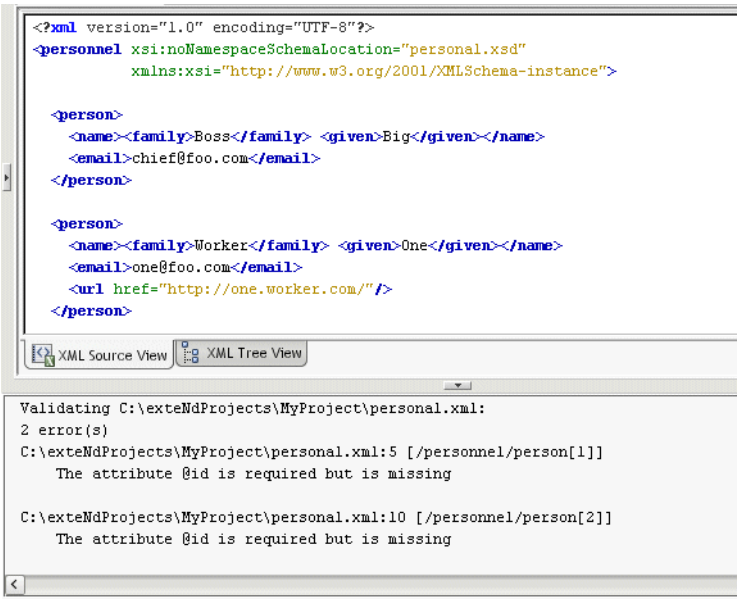
➢ **To validate an XML document and check for well-formedness:**

1 Open the XML document.

2 Select **XML Editor>Validate**.

NOTE: This 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. It also checks the XML document for well-formedness.



Results The report identifying any validation errors or malformed statements displays in the Output Pane. References to errors are reported as **XPaths**.

In the example above, the **id** attribute is reported as missing from the first two person elements. 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.

📖 For more information about XPaths, see the chapter on working with scoped paths and XPaths in *Developing exteNd Director Applications.*

TIP: You can search for specific XPaths in Tree View. See "Searching an XML document" on page 112.

➢ **To check only for well-formedness:**

1 Open the XML document.

2 Select **XML Editor>Well-formedness Check**.

Searching an XML document

You can search your document in either Source View or Tree View.

➢ **To search an XML document:**

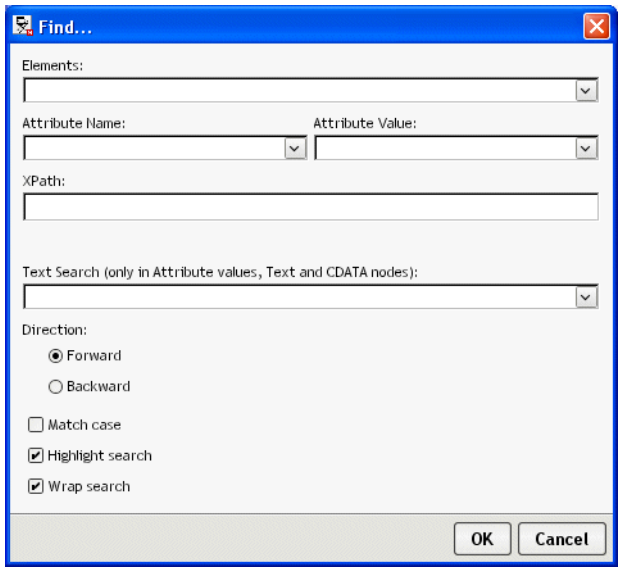
1 In either Source View or Tree View, select **Edit>Find** or press **Ctrl+F**.

The Find dialog displays.

2 In **Source View**, you can perform standard text searches.

In **Tree View**, you can specify one of the following:

- Element name
- Attribute name and/or value
- XPath (to learn about XPaths, see the chapter on working with scoped paths and XPaths in *Developing exteNd Director Applications*)
- Text (to search for a string in all attribute values, text nodes, and CDATA nodes)



3 Click **OK** to search.

If there is a match, the first is selected and all occurrences are indicated:

- In **Source View**, matches are highlighted
- In **Tree View**, matches are indicated with the search result icon

4 To go to the next occurrence, press **F3**.

Styling an XML document

You can style an XML document via **CSS** (Cascading Style Sheets). The exteNd Director development environment provides a **CSS Style Manager** you can use to attach and edit a style sheet for your XML document.

➢ **To style an XML document:**

1 Open the XML document.

2 Select **XML Editor>CSS Styling**.

The CSS Style Manager displays.

For details on attaching and editing style sheets with the CSS Style Manager, see Chapter 6, “CSS Editor”.

Maintaining the XML catalog

The exteNd Director development environment provides a built-in **XML catalog** of widely used Schemas and DTDs. For example, the XML catalog includes the Schemas for XSL, WSDL, and XML Schemas; the Sun J2EE DTDs; and the Novell exteNd Application Server deployment plan DTDs.

When you open an XML document that references a Schema or DTD, if the Schema or DTD is in the XML catalog, the editor associates it with the XML document and enables context editing and validation.

Catalog standard The XML catalog is based on the OASIS XML catalog standard. This 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 www.oasis-open.org/committees/entity/spec.html.

Catalog directories The built-in XML catalog consists of three directories in the Novell exteNd **Common\Resources** directory:

- **SchemaCatalog**, which contains a set of Schemas
- **DTDCatalog**, which contains a set of DTD files
- **CatalogFiles**, which contains catalog entry files

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 **../DTDCatalog**). Similarly, the two Schema-related catalog entry files both point to Schemas in the SchemaCatalog directory (so their base directory is **../SchemaCatalog**).

An 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 catalog entry file schemacatalog.xml lists this URI and maps it to XMLSchema.xsd in the SchemaCatalog directory, when you open personal.xsd the editor 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 XML 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 XML catalog is by using the existing catalog directory structure.

➢ **To add to the XML catalog by 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 CatalogFiles directory.
   - **user-dtdcatalog.xml**, whose base directory is DTDCatalog
   - **user-schemacatalog.xml**, whose base directory is SchemaCatalog
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 115.

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 XML 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 by using the **XML Catalog File Wizard**:

   2a Select **File>New>File**.

   2b On the **XML** tab, select **XML Catalog file** (in the Advanced section) and click **OK**.

   The XML Catalog File Wizard displays.

   2c Specify the **name** of the catalog entry file.

   2d Specify its **location**. In order to have exteNd Director read the catalog entry file, locate this file in the CatalogFiles directory.

   2e Specify the **base URI**—that is, the path to the directory containing the DTD or Schema files. It is through this base URI that exteNd Director is able to find the DTDs or Schemas listed in the catalog entry file.

   2f Click **Finish**.

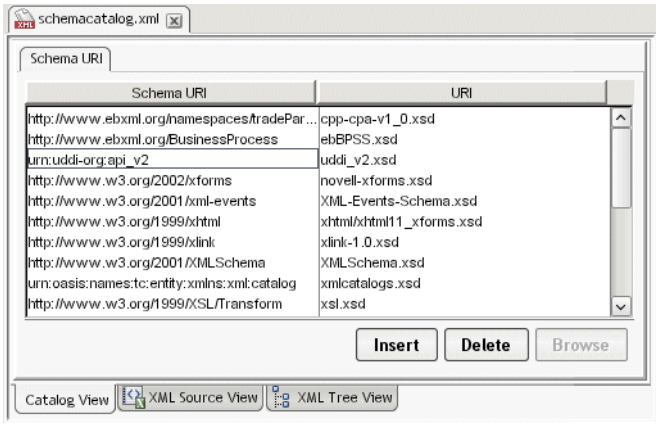
   The catalog entry file is opened in the XML Catalog Editor.

   2g Add entries as described in "Using the XML Catalog Editor" below.

Using the XML Catalog Editor

When you open a catalog entry file, it displays 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 DTDCatalog** has two tabs: Public Identifier and System Identifier
- A catalog entry file whose **base directory is 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 a DTD, select the appropriate tab and click **Insert**.
  - **For a Schema**, specify the Schema URI and the resolved URI
  - **For a DTD**, specify 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; the exteNd Director development environment may require them.*

➢ **To edit an entry:**

- **Double-click** the entry and make the 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).

Keyboard shortcuts

Here are the keyboard shortcuts provided in the XML editors.

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
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
Alt+Shift+H	Toggles display of search result icon
Ctrl+Alt+Shift+H	Clears the search

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

Keys	Description
Ctrl+Shift+V	Pastes the contents of the clipboard as the last child of the selected element
Delete	(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+D	Inserts new CDATA as last child
Ctrl+Shift+L	Inserts new element before selected node
Ctrl+Shift+D	Inserts new CDATA before selected node
Ctrl+K	Inserts new attribute
Ctrl+M	Inserts new 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

Keys	Description
Alt+Right Arrow	Moves the insertion point to first child
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
Shift+F11	Reformats the entire document

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: **1** Text preceding insertion point on same line **2** Indentation on same line **3** Line break **4** 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

In Catalog View (XML Catalog Editor)

Modifying text

Keys	Description
Ctrl+B	Displays dialog for changing the base URI (the path to the directory containing the DTD or Schema files for the catalog entry file)

5 XSL Editor

This chapter describes the facilities that the Novell exteNd Director development environment provides to work with XSL files. It contains the following topics:

About XSL

XSL (eXtensible Stylesheet Language) is a standard language for expressing style sheets. You can develop and use **XSL style sheets** to control how the contents of XML documents are displayed.

XSL includes the following features:

- **XSLT**, a language for transforming XML documents
- **XPath**, a language for specifying parts of an XML document
- **XSL Formatting Objects**, a vocabulary for formatting XML documents

The complete XSL standard can be found at www.w3.org/Style/XSL.

The power of XSLT Document transformation is the most important feature of XSL. The XSLT language was originally provided to perform complex styling operations, but you are now more likely to use it as a general purpose XML processing language. In particular, XSLT is useful for transforming an XML document into a different XML, XHTML, HTML, or other document.

XSLT works by transforming the XML input document (represented as a **source tree**) into the output document (represented as a **result tree**). You define **templates** that:

- **Select (via XPaths)** which parts of the source to process
- **Specify the transformations** to perform on those parts

XSL in the development environment

The exteNd Director development environment provides the following tools for working with XSL:

- **XSL File Wizard** for creating an XSL (style sheet) file
- **XSL Editor** for developing and testing an XSL file, including the transformations (XSLT processing) you want it to perform

Required files With either tool, you'll need the following files before you begin (unless you plan to write your XSLT code manually):

File	Purpose
XML source document	Provides the XML to be transformed
XML result document	Provides a sample of the XML to be produced; you'll map the source document's XML tree to the result document's XML tree in order to generate the XSLT code you need

Creating and opening XSL files

You can create new XSL files or work with existing ones.

➢ **To create a new XSL file:**

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

2 On the **XML** tab, select **XSL file**.

3 To create a blank XSL file, deselect **Use Wizard** and click **OK**. An empty XSL file is created and displayed in the XSL Editor.

To use the XSL File Wizard, select **Use Wizard** and click **OK**. The XSL File Wizard displays. Go through the wizard as follows.

4 Specify the **name** of the XSL file and click **Next**.

5 Specify the **location** of the XSL file and click **Next**.

6 Specify the **XML source document** and click **Next**.

7 Specify the **XML result document** and click **Finish**.

The XSL Editor displays with your new XSL file open.

➢ **To open an XSL file:**

1 Select **File>Open**.

2 In the Open dialog, select the XSL file and click **Open**.

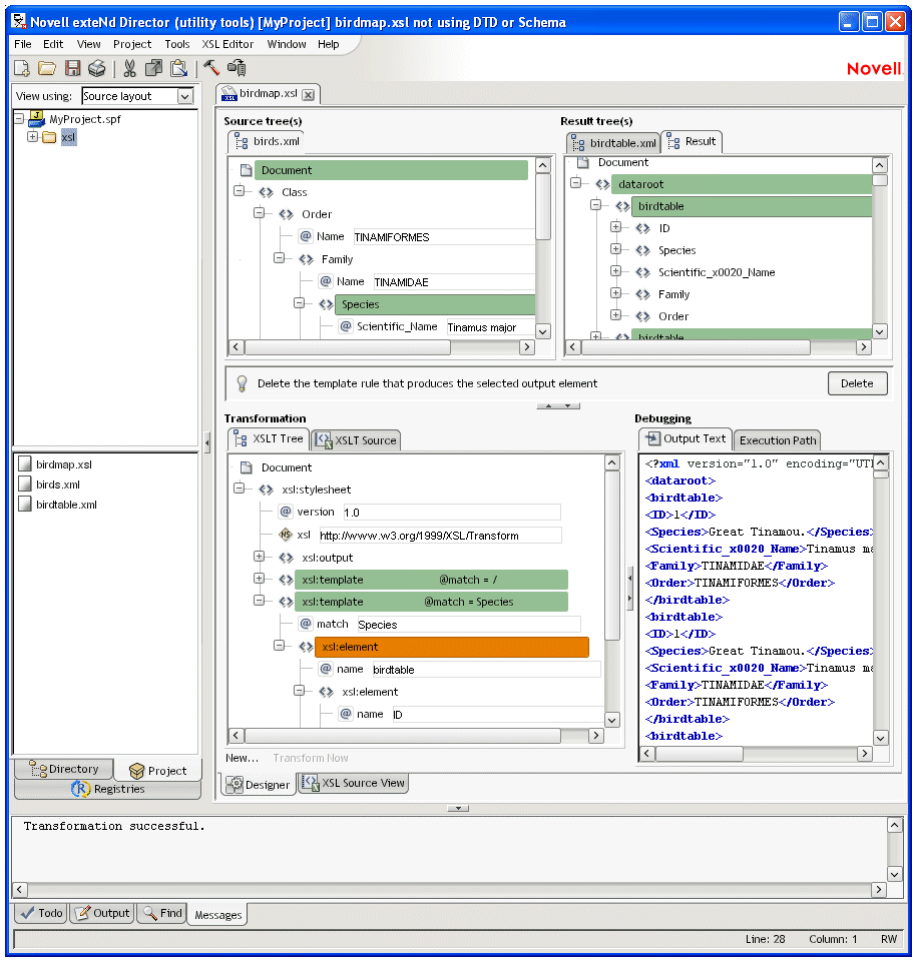
The file extension must be .XSL. The selected file opens in the XSL Editor and the XSL Editor menu appears on the menu bar.

Using the XSL Editor

The XSL Editor provides two views for working with an XSL file:

- The **Designer** tab displays a set of panes that work together to help you develop and test your XSL file and its XSLT processing in a graphical way:
  - You map nodes from the **Source tree** pane to nodes in the **Result tree** pane to create template rules
  - The template rules you create display in the **Transformation** pane
  - When you test your transformations, the **Debugging** pane displays the output
- The **XSL Source View** tab displays a source editor that you can use to examine and edit your XSLT code directly. The XSL Source View offers the same standard text editing features that are available in the XML Editor's Source View (for details, see Chapter 4, "XML Editors").

Here's an example of using the XSL Editor's Designer to develop a style sheet that transforms one XML file (containing information about birds) into another XML file with a different format:


Novell exteNd Director (utility tools) [MyProject] birdmap.xsl not using DTD or Schema
File Edit View Project Tools XSL Editor Window Help
Novell
View using: Source layout
MyProject.spf
xsl
birdmap.xsl
birds.xml
birdtable.xml
Directory
Project
Registries
Source tree(s)
Result tree(s)
Document
Class
Order
Name TINAMIFORMES
Family
Name TINAMIDAE
Species
Scientific_Name Tinamus major
dataroot
birdtable
ID
Species
Scientific_x0020_Name
Family
Order
Delete the template rule that produces the selected output element
Delete
Transformation
XSLT Tree
XSLT Source
xsl:stylesheet
version 1.0
xsl http://www.w3.org/1999/XSL/Transform
xsl:output
xsl:template @match = /
xsl:template @match = Species
match Species
xsl:element
name birdtable
xsl:element
name ID
New... Transform Now
Debugging
Output Text
Execution Path
<?xml version="1.0" encoding="UT
<dataroot>
<birdtable>
<ID>1</ID>
<Species>Great Tinamou.</Species>
<Scientific_x0020_Name>Tinamus m
<Family>TINAMIDAE</Family>
<Order>TINAMIFORMES</Order>
</birdtable>
<birdtable>
<ID>1</ID>
<Species>Great Tinamou.</Species>
<Scientific_x0020_Name>Tinamus m
<Family>TINAMIDAE</Family>
<Order>TINAMIFORMES</Order>
</birdtable>
<birdtable>
Designer
XSL Source View
Transformation successful.
Todo
Output
Find
Messages
Line: 28 Column: 1 RW

6 CSS Editor

This chapter describes the facilities that the Novell exteNd Director development environment provides to work with CSS files. It contains the following topics:

About CSS

CSS (Cascading Style Sheets) is a mechanism for applying styles (such as fonts, colors, and spacing) to Web documents (including HTML, XHTML, and XML files). You define a **style sheet** to specify the styling you want for a document. That style sheet can either be embedded in the document, or stored externally in a separate **CSS file** and attached to the document. The advantage of CSS files is that they enable you to reuse a style sheet with multiple documents.

The complete CSS standard can be found at www.w3.org/Style/CSS.

Inside a style sheet A style sheet consists of **rules**, which identify specific parts of a document and the styling to apply to them. Here's a typical rule:

```
.glossaryitem {font-weight: bold; color: slategray;}
```

In this rule:

- `.glossaryitem` is a **selector** that specifies a class of entities in the document; this rule will apply only to those entities. CSS provides several different kinds of selectors to give you flexibility when targeting your styling.
- `font-weight` is a style **property** to set. CSS provides many different properties to give you control over each aspect of style.
- `bold` is the **value** to set for the property.

Multiple rules in a style sheet can apply to a particular entity in a document. When this happens, the styling defined by those rules **cascades** to affect multiple properties of the entity.

Sample style sheet The following style sheet defines several different rules; these rules use various kinds of selectors and properties to specify what to style and how.

```
body {margin-top: 0pt;}

h1
  {
  font-family: arial, verdana, helvetica, sans-serif;
  font-size: 12pt;
  text-align: left;
  margin-top: 0pt;
```

```
  margin-bottom: 9pt;
  padding-top: 0pt;
  padding-bottom: 0pt;
  }

p, ol, ul, dl, dt, dd, table, td, th, select, input
  {
  font-size: 8pt;
  font-family: verdana, arial, helvetica, sans-serif;
  }

p.tablepara
  {
  padding-top: 0pt;
  padding-bottom: 0pt;
  margin-top: 1pt;
  margin-bottom: 1pt;
  }

.glossaryitem {font-weight: bold; color: slategray;}

#title
  {
  position: absolute;
  top: 25px;
  left: 4px;
  width: 208px;
  visibility: visible
  }
```

CSS in the development environment

The exteNd Director development environment provides the following tools for working with CSS:

- **CSS File Wizard** for creating a CSS (style sheet) file
- **CSS Editor** for examining and editing a CSS file
- **CSS Style Manager**, a dialog that you can access from other exteNd Director tools when CSS styling is called for

Creating and opening CSS files

You can create new CSS files or work with existing ones.

➢ **To create a new CSS file:**

1. Select **File>New>File**.
2. On the **Portlet** tab, select **CSS file**.
3. To create a blank CSS file, deselect **Use Wizard** and click **OK**. An empty CSS file is created and displayed in the CSS Editor.

   To use the CSS File Wizard, select **Use Wizard** and click **OK**. The CSS File Wizard displays. Go through the wizard as follows.
4. Specify the **name** of the CSS file.
5. Select which **wizard pages** you want to see to specify style sheet features:
   - Template
   - Vocabulary

- Basic appearance
- Button appearance
- Link appearance
- Headings appearance

This enables you to focus on just those style sheet features you want to specify right now. (You can always specify others later in the CSS Editor.)

6 Click **Next**.

7 Specify the **location** of the CSS file and click **Next**.

8 The wizard now displays the pages you selected, in order. Complete each page and click **Next**.

9 When you complete the last page, click **Finish**.

The CSS Editor displays with your new CSS file open.

➢ **To open a CSS file:**

1 Select **File>Open**.

2 In the Open dialog, select the CSS file and click **Open**.

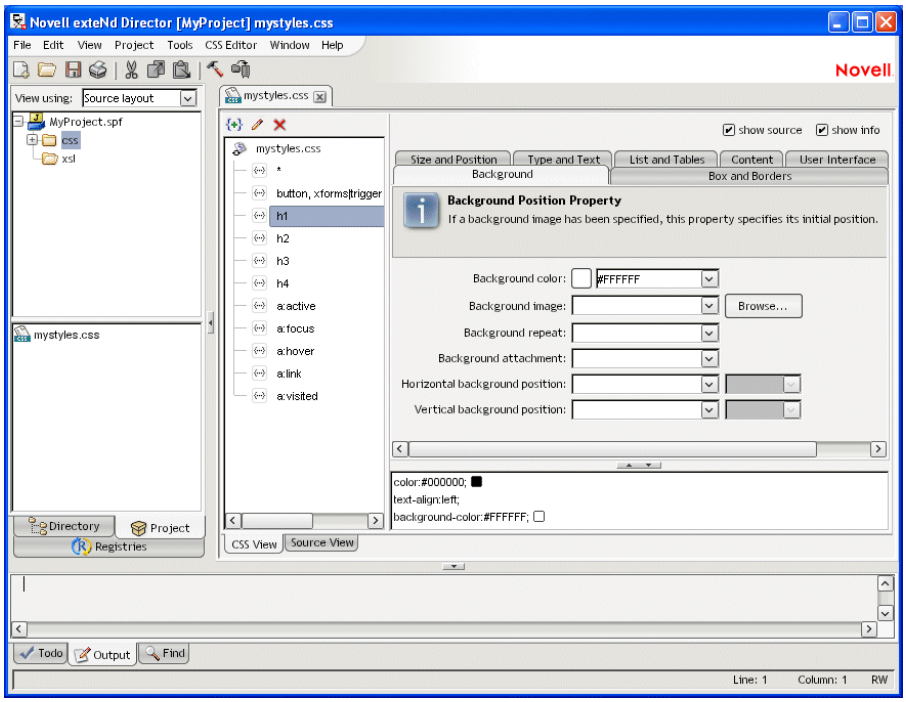
The file extension must be .CSS. The selected file opens in the CSS Editor and the CSS Editor menu appears on the menu bar.

Using the CSS Editor

The CSS Editor provides two views for working with a CSS file:

- The **CSS View** tab enables you to develop and examine your style sheet in a graphical way. You can:
  - Manipulate the list of **rules** for the style sheet
  - Specify the **selectors** for any rule
  - Set the **properties** for any rule
- The **Source View** tab displays a source editor that you can use to examine and edit your CSS code directly. The Source View offers the same standard text editing features that are available in the XML Editor's Source View (for details, see Chapter 4, "XML Editors").

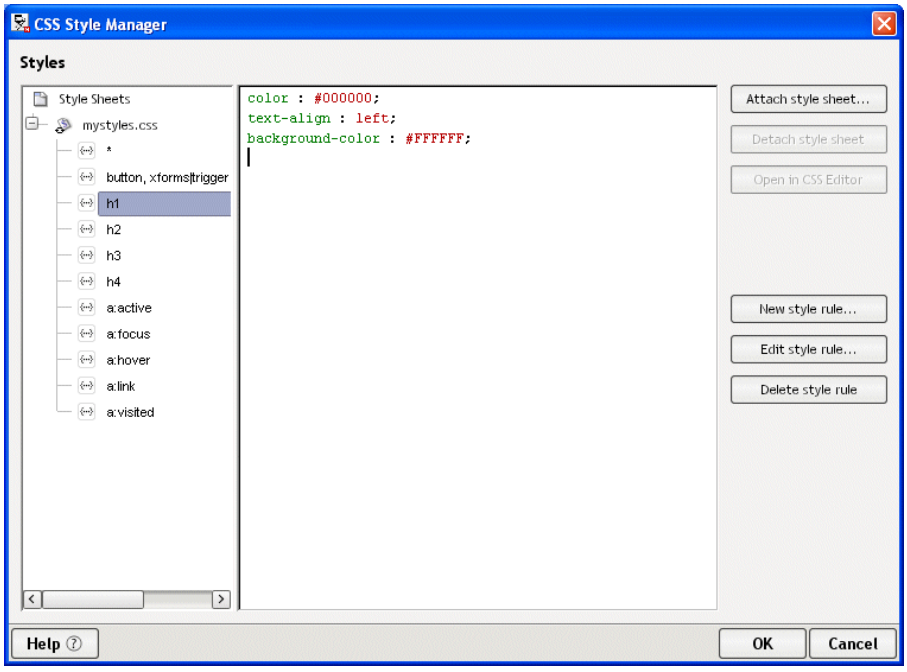
Here's an example of using the CSS Editor's CSS View to develop a simple style sheet:



Using the CSS Style Manager dialog

The CSS Style Manager is a dialog you can access from other exteNd Director tools when CSS styling is called for (such as while you're editing an XML file or XHTML file). This dialog enables you to:

- **Attach** a new or existing style sheet
- **Detach** a style sheet
- **Edit** a style sheet (either in the dialog or by opening the CSS Editor)



II Web Services

These chapters present the exteNd Director utility tools for working with Web Services:

- Chapter 7, “Web Service Basics”
- Chapter 8, “Generating Web Services”
- Chapter 9, “Generating Web Service Consumers”
- Chapter 10, “Web Service Wizard”
- Chapter 11, “WSDL Editor”
- Chapter 12, “Registry Manager”

Related link:

- Novell exteNd Web Services SDK

7 Web Service Basics

Web Services enable businesses to share application functionality regardless of the source language, operating system, or hardware used to create that functionality. Web Services overcome implementation incompatibilities by using **standard Internet protocols** and **XML-based messaging** to provide intercomponent communication.

This chapter gives an overview of Web Service technologies and how the Novell exteNd Director development environment supports the creation and use of Web Services. Topics include:

- About Web Services
- Web Service providers, consumers, and registries
- Providing Web Services
- Using Web Services
- Using Web Service registries
- Learning more about Web Services
- Popular Web Service implementations
- Web Service development tools

About Web Services

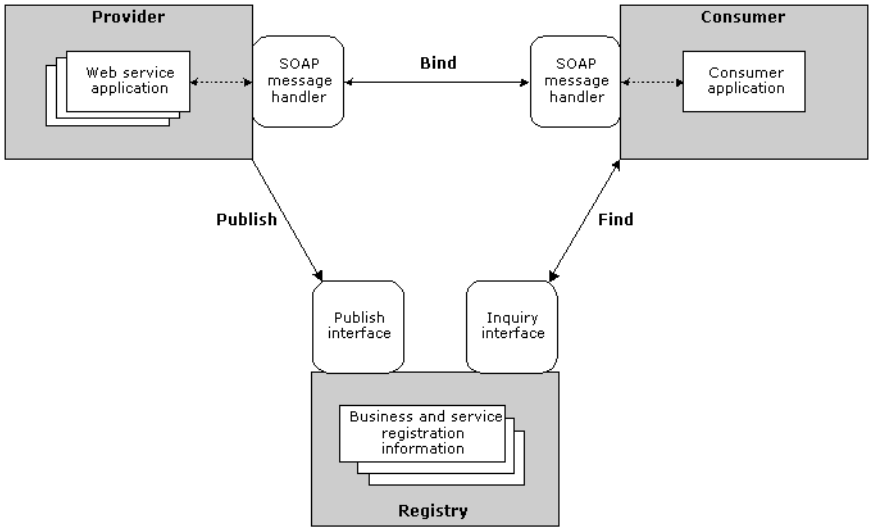
Web Services are modular software components whose application functionality is accessible over the Web using **Simple Object Access Protocol** (SOAP), a standardized XML-based messaging protocol.

Applications invoke Web Services like remote procedure calls, except that the procedure call and response are handled using **SOAP messages** embedded in **HTTP requests and responses**. An application calls a Web Service by sending a SOAP message embedded in an HTTP request to a Web location associated with that service. The Web Service performs the application logic for that message and then returns any application output in the form of another SOAP message embedded in an HTTP response.

To learn more about SOAP messages, see www.w3.org/TR/SOAP.

Web Service providers, consumers, and registries

The Web Service architecture typically consists of Web Service providers, consumers, and registries:



A Web Service **provider** is an organization that creates and hosts Web Services. Typically, a provider publishes information about their organization and the services they offer in a Web Service registry that can be queried by members of the organization or possibly by other businesses.

A Web Service **consumer** finds a Web Service (typically by querying a Web Service registry) then runs the service by establishing a connection to the provider. This is called **binding** to a Web Service.

A Web Service **registry** is a collection of business and service information that is readily accessible to providers and consumers, through programmatic publishing and querying interfaces.

Providing Web Services

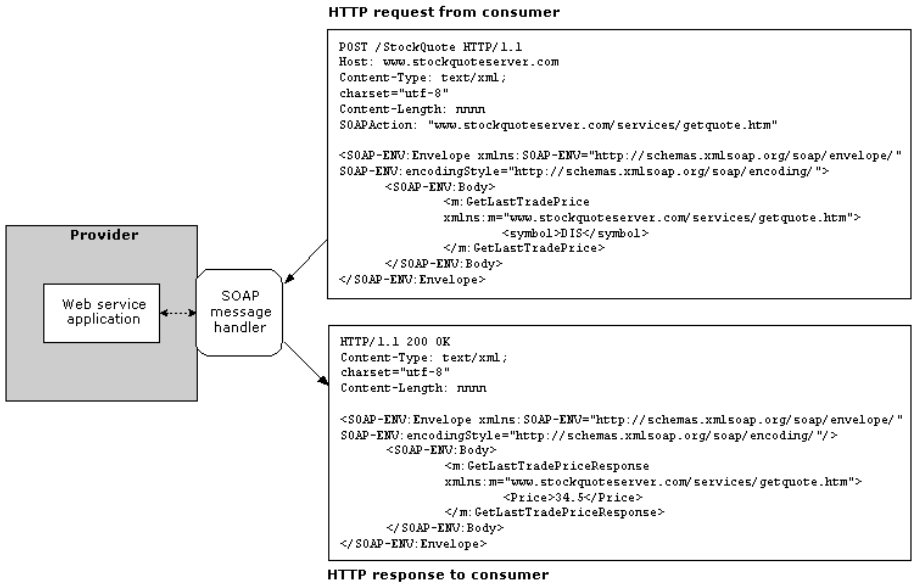
A Web Service provider:

1. Creates and deploys Web Service components
2. Creates a WSDL file to describe the Web Service
3. Publishes information about the Web Service so prospective consumers can discover and use it

Creating Web Service components

A provider creates the application logic components and deploys them to a network-accessible location, typically using a Web application server. To make these logic components into a Web Service, the provider creates and deploys a **SOAP message-handling interface** that enables HTTP requests containing well-defined SOAP messages to invoke the appropriate Web Service functionality.

When a consumer application accesses the service by sending a SOAP message embedded in an HTTP request, the provider runs the application logic and returns any application output in another SOAP message embedded in an HTTP response. For example:



Creating a WSDL file

To specify information about a Web Service in a standard form, the provider creates a **Web Services Description Language** (WSDL) document describing its characteristics. WSDL is an XML-based format that describes a Web Service by using these elements:

Element	Contains definitions of
Type	Data types specified in message content
Message	Data formats of messages
Port type	Endpoint types and the operations they support
Binding	Message formats and protocol details for a particular port type
Port	A network address for each endpoint
Service	Groups of related endpoints

In WSDL, an **endpoint** specifies a network address as well as the protocol and data format of messages exchanged with that address.

Given the flexibility of the WSDL specification, the information in a WSDL document can become complicated. For easier understanding, think of a WSDL document as essentially specifying the interface and port location of a Web Service.


To learn more about WSDL, see www.w3.org/TR/wsdl.

Publishing Web Service information

Once a Web Service has been created and deployed, the provider can publish information about the service and the provider organization in one or more registries. This enables prospective consumers to discover that the service is available and learn how to use it.


For details, see “Using Web Service registries” on page 134.

Another way to publish Web Service information is to provide the information directly to specific consumers by using Web pages, e-mail, personal communications, and so on. This is called **direct publishing**.

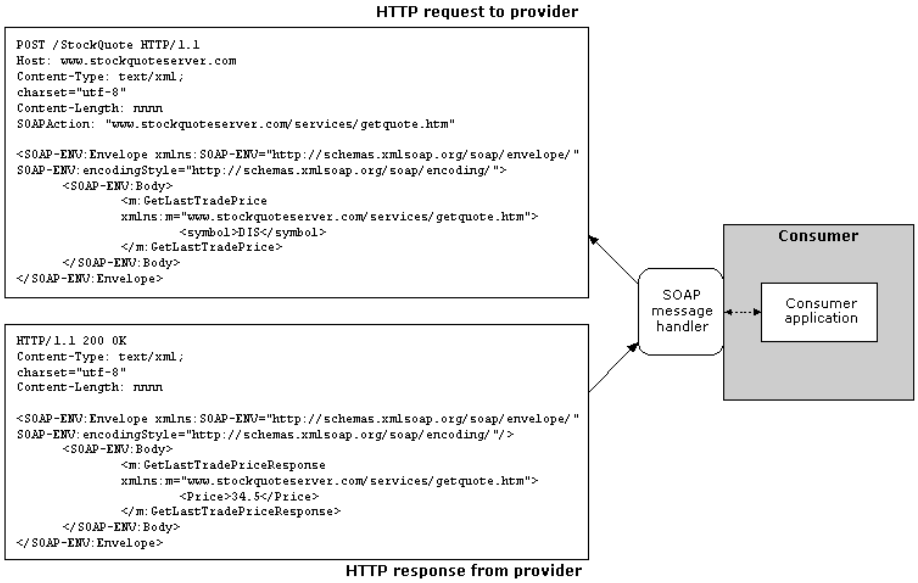
Using Web Services

A Web Service consumer creates applications that use Web Services. Typically, a consumer finds an appropriate Web Service by querying a Web Service registry (see "Using Web Service registries" on page 134).

From the WSDL information provided, the consumer can create the **SOAP message-handling code** needed to use the Web Service. When the consumer application calls the Web Service, the SOAP message-handling code **binds** to that service, as follows:

1. Establishes an HTTP connection to the provider
2. Creates and sends a SOAP message embedded in an HTTP request, instructing the provider to invoke the appropriate Web Service application logic
3. If the HTTP response contains a SOAP message, converts that message (into a data format understandable to the consumer application) then returns the data to the application

To the consumer application, this is similar to calling a remote method. However, the interaction between the application code and the Web Service uses SOAP messaging embedded in a standard HTTP request and response. For example:



Using Web Service registries

A Web Service registry is a repository of Web Service information that can be accessed programmatically over a network. Both providers and consumers can use Web Service registries:

- **Providers can publish** information about their organization and services to registries, making them visible to prospective consumers.
- **Consumers can query** registries to find the services and businesses that fit their needs and to retrieve provider-supplied information about those services (such as where and how to access them, the WSDL representation, and so on).

About registries

A registry can contain these kinds of information:

Category	Includes
Business information	Name, industry or product category, geographic location, and business identification numbers (such as NAICS or DUNS numbers)
Web Service information	General description, business process or category, and technical information (about connecting to and communicating with the Web Services for a given business)
Business service information	Corporate home page URL, sales and technical support contact information, business services not hosted on the Web, and so forth
Specification pointers	URL addresses of WSDL for services and other technical documents

Registry data formats

Registries store their business and service information in a standard XML-based format such as **Universal Description, Discovery and Integration** (UDDI) or **Electronic Business XML** (ebXML). Businesses hosting registries typically provide Web page, GUI, or programmatic interfaces for publishing to and querying the registry (so providers and consumers don't need to know details about the internal registry implementation).

To learn more about UDDI, see www.uddi.org. To learn more about ebXML, see www.ebxml.org.

Public and local registries

Businesses may use public or local registries:

- A **public registry** is typically visible to anyone on the Web and contains information about numerous companies and services. It may implement varying degrees of authentication and authorization security for publishing and querying.
- A **local registry** might be limited to local network access, enabling a business to share Web Services internally without exposing them to consumers outside the organization.

Learning more about Web Services

Here's a summary of Web sites you can visit to find out more about specific Web Service technologies:

Topic	Site
SOAP	www.w3.org/TR/SOAP
HTTP	www.w3.org/Protocols
WSDL	www.w3.org/TR/wsdl
UDDI	www.uddi.org
ebXML	www.ebxml.org

Popular Web Service implementations

While it's important to know about the underlying Web Service technologies (SOAP, WSDL, UDDI, ebXML, and so on), it's usually not efficient to develop applications at that level. As a result, higher-level implementations have emerged to make those technologies more accessible by wrapping them in familiar constructs. These implementations include:

- **J2EE** Java 2 Enterprise Edition provides Web Service support through its JAX-RPC (Java API for XML-based RPC) specification.
- **.NET** Microsoft provides Web Service support through its .NET platform.

For example, a programmer familiar with J2EE can more easily use a JAX-RPC implementation to develop and access Web Services. There's no need to become a SOAP expert or process SOAP messages manually.

When properly designed and built, Web Services should be **interoperable** across different implementations. For instance, a JAX-RPC client should be able to access a .NET Web Service and a .NET client should be able to access a JAX-RPC Web Service.

Web Service development tools

The exteNd Director development environment provides tools for creating, deploying, and maintaining Web Services based on the **JAX-RPC** standard. That means Web Services are packaged in J2EE **Web archives** (WARs) that can be deployed to a J2EE server. You can also develop Java-based Web Service consumers that comply with JAX-RPC.

To help you implement Web Services and Web Service consumers, the development environment includes the following core facilities:

Facility	Description
Web Services SDK	Core technologies for exteNd Web Service support, including compilers and SOAP runtime based on JAX-RPC
Web Service Wizard	Tool that helps you invoke the Web Services SDK compilers to generate Java classes and WSDL files for Web Services and Web Service consumers
Registry Manager	Tool for querying and publishing to Web Service registries
WSDL Wizard and Editor	Tools for creating and editing WSDL files

On top of those features, exteNd Director adds higher-level facilities that make it easy to take full advantage of Web Services in your applications.

 To learn about these other Web Service facilities of exteNd Director, see the Pageflow and Form Guide.

Web Services SDK

The Novell exteNd Web Services SDK is a JAX-RPC implementation that includes **compilers** and a **runtime environment** for developing and executing Web Service provider and consumer applications.

The Web Service Wizard uses the Web Services SDK compilers to create Web Service components (skeletons, ties, stubs) and WSDL files. Developers can also invoke these compilers separately from the command line.

Both provider and consumer deploy **wssdk.jar** (and some supporting JARs) with their applications to provide the necessary runtime environment. This includes the SOAP engine that runs when stub and skeleton components pass SOAP messages between consumer and provider applications.

📖 For more information, see the Web Services SDK help.

Web Service Wizard

The Web Service Wizard enables you to create Web Service components from Java classes or WSDL files. It generates the Java **remote interface** for accessing an object as well as **skeleton**, **tie**, and **stub** Java classes that handle SOAP message communication between a consumer application and a Web Service. The generated code is based on JAX-RPC.

The provider deploys a Web Service as a Web archive (WAR) in which the skeleton and tie classes implement a **servlet** that processes incoming SOAP messages. A consumer application accesses Web Service functionality by calling methods in the stub class, which sends SOAP messages to the server.

📖 For more information, see:

- Chapter 8, “Generating Web Services”
- Chapter 9, “Generating Web Service Consumers”
- Chapter 10, “Web Service Wizard”

Registry Manager

The Registry Manager helps providers publish to Web Service registries. It helps consumers query Web Service registries.

📖 For more information, see Chapter 12, “Registry Manager”.

WSDL Wizard and Editor

The WSDL Wizard helps providers create new WSDL documents. The WSDL Editor helps providers edit and use existing WSDL documents.

📖 For more information, see Chapter 11, “WSDL Editor”.

8 Generating Web Services

This chapter walks you through the basic steps and typical scenarios for using the **Web Service Wizard** to generate Web Services from a variety of sources. Topics include:

- Basics
- Steps
- Choosing an implementation model
- Scenario: starting with a Java class

 To learn about the steps and scenarios for using the wizard when you want a program to access Web Services, see Chapter 9, “Generating Web Service Consumers”.

Basics

You can use the Web Service Wizard of the Novell exteNd Director development environment to develop **standard (SOAP-based) Web Services** that are implemented as **Java remote objects** (using RMI). The wizard generates Java source files based on JAX-RPC (Java API for XML-based RPC) and the Novell exteNd Web Services SDK (the JAX-RPC implementation included with Novell exteNd). JAX-RPC is the J2EE specification that provides Web Service support.

The generated files include a **servlet** to handle access to your Web Service and its methods from HTTP SOAP requests. 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.

 For an introduction to Web Service concepts, standards, and technologies, see Chapter 7, “Web Service Basics”.

 For detailed documentation on the wizard, see Chapter 10, “Web Service Wizard”.

Steps

The complete development process involves:

1. Preparing to generate
2. Generating Web Service files
3. Examining the generated files
4. Editing the generated files
5. Using the generated files

Preparing to generate

To prepare for using the Web Service Wizard, you:

1 Set up a **WAR project** in the exteNd Director development environment.

For each Web Service you generate, the wizard creates a **servlet** 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.

A possible variation is to set up a **JAR subproject** in your WAR and use that JAR to contain the servlet and other classes for a Web Service. In any case, the **servlet mapping** will be in the WAR's deployment descriptor (web.xml).

(Note that the approach of using a JAR subproject is not currently supported by the Web Service Wizard when you generate a Web Service from a WSDL file. In this situation, it only supports a WAR project.)

2 **Add these files** to the project:

Files	Details
Source files, classes, or archives from which your Web Services are to be generated	You can generate a Web Service from any one of the following: ◆ A **JavaBean** or other **Java class** ◆ An **EJB session bean** ◆ A **Java remote interface** ◆ A **WSDL file** No matter which one you provide, it should (at minimum) declare the methods you want your generated Web Service to expose. **Compile your Java files** If you provide any Java files, make sure you compile them in your project before starting the Web Service Wizard (because the wizard works from compiled classes). **No overloaded methods** Overloaded method names are not allowed in Web Service interfaces (as of WSDL 1.2). If you're generating from: ◆ A JavaBean or other Java class, the wizard lets you pick no more than one method with a given name ◆ An EJB, remote interface, or earlier WSDL file, you must remove any overloaded method names defined in that file before you start the wizard

Files	Details
Archives required by the Web Services SDK: ◆ **wssdk.jar**, which contains the Web Services SDK API classes needed at runtime ◆ jakarta-regexp-1.2.jar (when using a pre-1.4 JDK and XML Schema with patterns) ◆ xercesImpl.jar (when using SilverStream eXtend Application Server 4.x) ◆ xmlParserAPIs.jar (when using SilverStream eXtend Application Server 4.x) ◆ xmlsec.jar (when using XML Signature)	You'll find these JARs in the Novell exteNd **tools\compilelib** directory. Depending on your J2EE server configuration, you should do one of the following: ◆ Add them to the **WEB-INF/lib** directory of your WAR project ◆ Add them to the **server classpath** of your J2EE server 📖 For more information, see "Deploying Web Services" on page 283.

3 Edit the **classpath** of your project so you can compile your Web Service classes once they're generated and edited. You'll need to include:

- wssdk.jar and the supporting JARs listed in Step 2 (as appropriate)
- j2ee_api_1_*n*.jar (automatically added when you create a WAR project)
- Any application-specific entries (such as an EJB-client JAR file you've provided for a session bean Web Service)

Generating Web Service files

Once you've set up your WAR project, you're ready to use the Web Service Wizard. The wizard produces one Web Service at a time, so you'll need to use it multiple times if you have several to develop.

Each time you launch the wizard, it takes input from you about the kind of Web Service to produce. It then generates a set of source files that together make up the Web Service. Here's a summary of the process:

1 Select **File>New>File** to display the New File dialog and go to the **Web Services** tab.

2 Launch the Web Service Wizard by doing one of the following:

To generate a Web Service from	Select
One of these: ◆ A JavaBean or other Java class ◆ An EJB session bean ◆ A Java remote interface	**New Web Service**
A WSDL file	**Existing Web Service** As its name suggests, this item is mainly used to generate **Web Service consumers** that access deployed Web Services (based on their WSDL files). But it can also be used to read WSDL files as blueprints and generate the matching **Web Services** themselves.

3 When the wizard prompts you for **project location** information, specify:

- The **WAR or JAR project** you set up to contain the generated Web Service files (if you're generating from a WSDL file, the wizard currently requires you to specify a WAR project here)
- The target **directory and package** in that project (if you're generating from a Java class, you won't have to fill in some of these settings; the wizard will automatically handle them for you)

If you specify a JAR project to contain the generated Web Service files, the wizard will also ask you for a WAR project to map the Web Service's servlet.

4 When the wizard prompts you, select the **class or WSDL file** to generate the Web Service from.

The wizard then asks for additional information based on your selection:

If you select	The wizard prompts you to specify
A JavaBean or other Java class	◆ Which methods to expose in the generated Web Service (in contrast, when you generate from an EJB, remote interface, or WDSL file, all methods are automatically exposed) ◆ Binding style (document or RPC) ◆ Schema information (when appropriate) ◆ Class-generation and SOAP options
The home interface of an EJB session bean	◆ Lookup information for the EJB ◆ Binding style (document or RPC) ◆ Schema information (when appropriate) ◆ Class-generation and SOAP options
The remote interface of an EJB session bean or the SessionBean class itself	◆ The home interface of the EJB session bean ◆ Lookup information for the EJB ◆ Binding style (document or RPC) ◆ Schema information (when appropriate) ◆ Class-generation and SOAP options
A Java remote interface	◆ Binding style (document or RPC) ◆ Schema information (when appropriate) ◆ Class-generation and SOAP options
A WSDL file	◆ Namespace-to-package mappings (when there are multiple) ◆ Web Service type mappings ◆ Class-generation and SOAP options

5 When the wizard prompts you for **class-generation and SOAP options**, you need to choose and configure the set of source files to generate for your Web Service.

The most important choice is whether to generate **skeletons** to be **tie-based** or not. The answer depends on the architectural model you want the implementation of your Web Service to follow. See "Choosing an implementation model" on page 149.

You can choose to generate **stubs** (which come with a simple client application) for testing your Web Service. When generating from a Java class, you can also request a **WSDL file** (for publishing the Web Service to a registry) as well as specify the **service address** (URL) for the Web Service. When generating from a WSDL file, you can override the default service address if necessary.

NOTE: Support for jBroker™ Web 1.*x* applications is available via a **backward-compatibility** option. For more information, see "If you choose jBroker Web 1.x compatibility" on page 145.

6 Click **Finish** when you're done specifying options for the Web Service.

Examining the generated files

When you finish the wizard, it generates everything you've specified for your Web Service and updates other parts of your project with supporting changes:

What the wizard generates	Details
Java source file for remote interface	***xxx*WS.java** This file is automatically generated whenever your input to the wizard is not a remote interface (such as when you start from a JavaBean, Java class, EJB session bean, or WSDL file). That's because a remote interface (which extends java.rmi.Remote and declares the methods to expose) is required to construct your Web Service. When you start from a WSDL file, the name of the generated remote interface is simply *xxx*.java.
Java source file for skeletons	***xxx*_ServiceSkeleton.java** Abstract servlet class that handles access to the Web Service (from HTTP SOAP requests). In the tie model, *xxx*_ServiceTieSkeleton extends this class. In the skeleton model, you extend it yourself (with an implementation of your remote interface).
Java source files for tie-based skeletons	***xxx*_ServiceTieSkeleton.java** Abstract servlet class that extends *xxx*_ServiceSkeleton. ***xxx*Tie.java** Servlet that's used in the tie model as the front end for the Web Service. It extends *xxx*_ServiceTieSkeleton to handle access to the Web Service (from HTTP SOAP requests). It delegates to one of the following to process method calls for the Web Service: ◆ If you start with a JavaBean, Java class, or EJB session bean, *xxx*Tie instantiates *xxx*Delegate and delegates to it. ◆ If you start with a Java remote interface or WSDL file, you must edit the *xxx*Tie.java file to specify a class of your own to instantiate and delegate to. ***xxx*Delegate.java** This file is generated if you start with a JavaBean, Java class, or EJB session bean that implements the methods for your Web Service. *xxx*Delegate instantiates that implementation class and calls those methods on it. With an EJB session bean, *xxx*Delegate does a lookup and create to get the remote interface object. Then it uses that object to make the method calls.

What the wizard generates	Details
Java source files for stubs	***xxx*Service.java** Service interface used by JAX-RPC clients to obtain the stub for the target Web Service. ***xxx*ServiceImpl.java** Service implementation class that handles instantiation of the stub (*xxx*_Stub). It also supports alternative ways of accessing the target Web Service, including dynamic (stubless) calls. (Note that when you start from a WSDL file, the names generated for the service interface and implementation class depend on your WSDL and may omit the text **Service**.) ***xxx*_Stub.java** Facilitates method calls from a Java-based consumer to the target Web Service. *xxx*_Stub implements the remote interface corresponding to the Web Service by sending an appropriate HTTP SOAP request for each method call. ***xxx*Client.java** Simple client application that works as a consumer of the target Web Service. It obtains the stub (via the Service object), then uses the stub to call Web Service methods. You can run *xxx*Client from the exteNd Director development environment (select **Tools>Run Web Service Client Class**) or from a command line.
WSDL file	***xxx*.wsdl** For use when publishing your Web Service to a registry. It describes the Web Service in a standard format.
Updates to deployment descriptor	In the tie model (when you generate tie-based skeletons), the wizard updates your WAR project's **web.xml** file to declare *xxx*Tie as the servlet to handle HTTP SOAP requests for your Web Service. In the skeleton model, you must edit web.xml yourself to declare the servlet to use (your class that extends *xxx*_ServiceSkeleton).
Updates to project contents	The wizard updates your project to add generated files (and other application-specific files) to it.
Updates to project classpath	The wizard updates your project classpath to include application-specific files as needed.

About generated file names

When generating file names, the Web Service Wizard follows the naming rules specified by JAX-RPC. If you start with a Java class, the resulting file names are based on the name of that class. If you start with WSDL, the resulting file names are based on the definitions in that WSDL.

For simplicity, this documentation uses *xxx* to represent the portion of a generated Web Service file name that's derived from a class name or WSDL definition.

Additional details of generation

Under the covers, the Web Service Wizard uses the **Web Services SDK compilers** when generating the Web Service files listed above. In some cases, these compilers may generate additional code or files to support requirements specific to your application, such as:

- Type mapping
- Faults
- Multiple portType definitions

 For more information, see the Web Services SDK help.

If you choose jBroker Web 1.*x* compatibility

The current version of the Web Services SDK provides a high degree of backward compatibility with earlier versions. However, some changes introduced to support the JAX-RPC standard may require you to modify code when upgrading an application that originated in jBroker Web 1.*x*. These changes involve the conventions used for:

- **File names** JAX-RPC specifies rules for naming certain Web Service files. In order to follow these rules while keeping all generated names simple and consistent, new name patterns were adopted (for details, see Generated 1.x-compatible files below).
- **Stub access in client code** With JAX-RPC, clients use a Service object to instantiate the stub instead of looking up the stub directly via JNDI.

Although it's recommended that you upgrade to the current Web Services SDK and JAX-RPC conventions, it's not required. By using the **jBroker Web 1.*x* compatibility** option in the Web Service Wizard, you can generate Web Service files according to the original jBroker Web conventions for file names and stub access. This enables you to take advantage of all the other improvements in the latest version of the Web Services SDK without altering your existing 1.*x* applications.

Generated 1.*x*-compatible files The following table describes the files generated when you use the jBroker Web 1.*x* compatibility option:

With 1.*x* compatibility on, you get	With 1.*x* compatibility off, this is named	Details
*xxx*_REMOTE.java Example: `MyObject_REMOTE.java`	*xxx*WS.java Example: `MyObjectWS.java`	Generated remote interface
_*xxx*_ServiceSkeleton.java Example: `_MyObject_REMOTE_ServiceSkeleton.java`	*xxx*_ServiceSkeleton.java Example: `MyObjectWS_ServiceSkeleton.java`	Abstract servlet class
_*xxx*_ServiceTieSkeleton.java Example: `_MyObject_REMOTE_ServiceTieSkeleton.java`	*xxx*_ServiceTieSkeleton.java Example: `MyObjectWS_ServiceTieSkeleton.java`	Abstract tie servlet class
*xxx*_TIE.java Example: `MyObject_TIE.java`	*xxx*Tie.java Example: `MyObjectWSTie.java`	Servlet for the Web Service (in the tie model)
*xxx*_SERVICE.java Example: `MyObject_SERVICE.java`	*xxx*Delegate.java Example: `MyObjectWSDelegate.java`	Delegate class for the tie servlet
*xxx*Service.java Example: `MyObjectREMOTEService.java`	*xxx*Service.java Example: `MyObjectWSService.java`	Service interface for the stub This class is not used in 1.*x*-style stub access. It is generated in case you want to upgrade your client code to the JAX-RPC approach.

With 1.*x* compatibility on, you get	With 1.*x* compatibility off, this is named	Details
*xxx*ServiceImpl.java Example: `MyObjectREMOTEServiceImpl.java`	*xxx*ServiceImpl.java Example: `MyObjectWSServiceImpl.java`	Service implementation class for the stub This class is not used in 1.*x*-style stub access. It is generated in case you want to upgrade your client code to the JAX-RPC approach.
_*xxx*_ServiceStub.java Example: `_MyObject_REMOTE_ServiceStub.java`	*xxx*_Stub.java Example: `MyObjectWSBinding_Stub.java`	Stub for the Web Service
*xxx*_CLIENT.java Example: `MyObject_CLIENT.java`	*xxx*Client.java Example: `MyObjectWSClient.java`	Client application for consuming the Web Service The 1.*x*-compatible client obtains the stub directly via a JNDI lookup. In contrast, the JAX-RPC client obtains the stub indirectly via the Service object.
xxx.wsdl Example: `MyObject_REMOTE.wsdl`	*xxx*.wsdl Example: `MyObjectWS.wsdl`	WSDL file for the Web Service

Editing the generated files

Follow these guidelines when editing the files generated by the Web Service Wizard:

Guideline	Details
File you **may** need to edit	◆ *xxx*Tie.java See “Editing the xxxTie.java file” on page 146.
File you **must** edit	◆ *xxx*Client.java See “Editing the xxxClient.java file” on page 147.
Files you should **not** edit	◆ *xxx*_ServiceSkeleton.java ◆ *xxx*_ServiceTieSkeleton.java ◆ *xxx*Service.java ◆ *xxx*ServiceImpl.java ◆ *xxx*_Stub.java

It’s OK to edit any of the other generated files, but not typically required.

In some cases, completing the implementation of your Web Service may require you to add one or more manually coded files to work with the generated ones. See “Creating additional files” on page 147.

Editing the *xxx*Tie.java file

The generated *xxx*Tie.java file includes an **init() method** you may need to edit.

If you start with a JavaBean or Java class, init() is generated to call the setTarget() method of *xxx*_ServiceTieSkeleton and pass an instance of *xxx*Delegate (to delegate to it). If *xxx*Delegate provides an empty constructor, the generated code uses that constructor to do the instantiation.

But if no implicit or explicit empty constructor is available, you must modify the code to indicate which one to use. You may also want to modify it to use a constructor that expects an argument.

The wizard automatically generates calls to setTarget() for every public constructor of *xxx*Delegate. Each line is commented out—except the one that uses the empty constructor (if available). Uncomment the line with the constructor you want and make any related changes:

```
//super.setTarget( new MyObjectWSDelegate(  java.lang.String arg0) );
//super.setTarget( new MyObjectWSDelegate(  java.lang.String arg0, java.lang.String arg1) );
super.setTarget( new MyObjectWSDelegate( ) );
```

If you start with a Java remote interface or WSDL file, init() is always generated with the setTarget() call commented out. In this case, you must provide a class of your own to instantiate and delegate to:

```
//super.setTarget(new CONSTRUCT_YOUR_SERVICE_OBJECT_HERE);
```

If you start with an EJB session bean, you shouldn't need to edit the generated init() method.

Editing the *xxx*Client.java file

Before you can test your Web Service with *xxx*Client, you must edit the generated *xxx*Client.java file to call one or more methods of the Web Service. Look for the **process() method** in this file and you'll find comments listing all of the possible method calls:

```
// System.out.println("Test Result = " + remote.getString());
// System.out.println("Test Result = " + remote.setString(java.lang.String));
// System.out.println("Test Result = " + remote.sayHello());
```

Uncomment the method call(s) you want to test and supply appropriate argument values, as needed:

```
// System.out.println("Test Result = " + remote.getString());
System.out.println("Test Result = " + remote.setString(args[0]));
System.out.println("Test Result = " + remote.sayHello());
```

 For additional changes you may want to make to the generated *xxx*Client.java file, see Chapter 9, "Generating Web Service Consumers".

Creating additional files

In many scenarios, once the wizard finishes generating, you'll have all of the Java source files you need for your Web Service. But there are cases where you must code additional classes yourself:

In this case	You must add
When using the skeleton model	A class that extends the generated servlet *xxx*_ServiceSkeleton and implements the remote interface for your Web Service. You'll use this manually coded class as the servlet for the Web Service.
When using the tie model and starting with a Java remote interface or WSDL file	A class that implements the remote interface for your Web Service. You must edit the generated *xxx*Tie.java file to instantiate this manually coded class and delegate to it.

Using the generated files

To use the Web Service files generated by the wizard, you:

1 **Update the deployment descriptor**, if necessary.

When you use the tie model, the wizard automatically updates the WAR project's web.xml file with the appropriate servlet mapping for your Web Service. But with the skeleton model, you must edit web.xml yourself to supply this information.

In the following example, MyService is the servlet class that the developer has coded for the Web Service MyRemote:

```
<servlet>
   <servlet-name>MyService</servlet-name>
   <servlet-class>com.exsamp.rem.MyService</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>MyService</servlet-name>
   <url-pattern>/MyRemote</url-pattern>
</servlet-mapping>
```

Another reason to edit web.xml is to enable or disable the browser-based test environment for your Web Service.

📖 For details, see "Testing your Web Service from a browser" on page 149.

2 **Update the project**, if necessary.

As the wizard works, it automatically adds files to your project classpath and contents, as needed. But you should also check yourself to make sure the project has everything it requires to compile and run.

For instance, if your Web Service accesses an EJB session bean, the EJB-client JAR file should be on your project's classpath.

📖 For details on setting up the required classpath and contents for your project (including what the Web Services SDK needs), see "Preparing to generate" on page 140.

3 **Build and archive the project**.

When you complete this step, you'll have a WAR file containing the Web Service(s) you've generated.

4 **Set up for deployment** to your J2EE server.

Prepare the server-specific deployment information required to deploy the WAR to your J2EE server. For example, if you're going to deploy to the Novell exteNd Application Server, create an exteNd deployment plan file.

If you're going to deploy from the exteNd Director development environment, you should also set up a server profile for your J2EE server.

5 **Deploy the WAR** to your J2EE server.

When you complete this step, each Web Service in the WAR will be accessible as a servlet that can respond to standard HTTP SOAP requests for your exposed methods.

6 **Test your Web Service(s)** running on the J2EE server.

If you've generated, edited, and compiled the *xxx*Client class for a Web Service, you can use it for a quick test of your method calls. To run *xxx*Client from the exteNd Director development environment, select **Tools**>**Run Web Service Client Class**. The Web Service Wizard Client Runner displays, offering you a list of client classes from the current project to choose from.

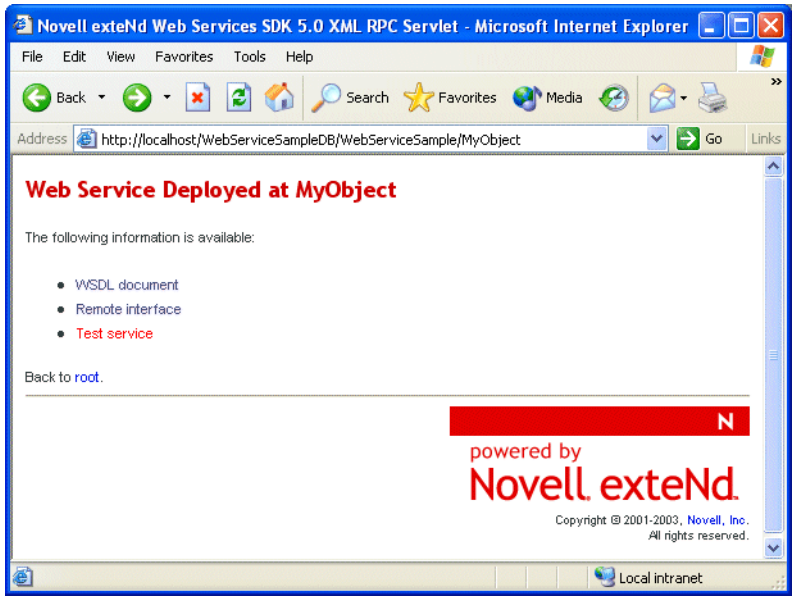
You can also run *xxx*Client from a command line (providing that you include the appropriate directories and archives on your system classpath).

📖 For further details on running *xxx*Client, see Chapter 9, "Generating Web Service Consumers".

Testing your Web Service from a browser

The Web Services SDK provides a feature to help you test your Web Services from a Web browser. Once you develop and deploy a Web Service, you can **browse to the URL** for that service to:

- **View the WSDL file** that describes the Web Service
- **View the remote interface** for the Web Service
- **Run test forms** for each available method; from there you can enter any required parameters, invoke the Web Service, and see the result



Runtime availability of this testing feature is controlled via an environment entry named **wssdk.test.disable** in your WAR's deployment descriptor (web.xml). When you create a generic WAR project in the exteNd Director development environment, the New Project Wizard automatically includes wssdk.test.disable in web.xml and sets it to **true**. To enable the feature, you must set wssdk.test.disable to **false** before you deploy.

```
<env-entry>
   <env-entry-name>wssdk.test.disable</env-entry-name>
   <env-entry-value>false</env-entry-value>
   <env-entry-type>java.lang.Boolean</env-entry-type>
</env-entry>
```

Choosing an implementation model

There are two basic implementation models you can choose from when developing with the Web Service Wizard. This section explores these choices to help you select the one that's most appropriate for the Web Services you generate:

- Tie model
- Skeleton model

Tie model

Here's an overview of the tie model and when to use it:

Topic	Details
Typical use	The tie model is typically used when you have an **implementation class** to provide as input to the Web Service Wizard. That might be a JavaBean, Java class, or EJB session bean that already implements the methods you want to expose as a Web Service.
How it works	The tie model uses a **delegation** approach to hand off method calls from the generated Web Service classes (which handle the HTTP SOAP processing for your Web Service) to your implementation class (which handles the method processing).
Advantages	The tie model enables you to keep your implementation class (business logic) separate from the generated infrastructure classes that support your Web Service. A related benefit is that you can reuse existing implementation classes currently accessible via other protocols.
How to generate it	When you specify class-generation and SOAP options in the Web Service Wizard, check both of these items: ◆ Generate skeletons ◆ Tie-based
Files generated	If you start with a JavaBean, Java class, or EJB session bean, the wizard generates: ◆ *xxx*WS.java (remote interface) ◆ *xxx*Delegate.java ◆ *xxx*Tie.java ◆ *xxx*_ServiceTieSkeleton.java ◆ *xxx*_ServiceSkeleton.java

It's possible (but not as common) to use the tie model when you have only a Java remote interface or WSDL file to provide as input to the Web Service Wizard. In this case, the wizard output leaves the delegation part of the model for you to complete later. You'll then need to code an implementation class and edit the generated tie class to instantiate it and delegate to it.

Skeleton model

Here's an overview of the skeleton model and when to use it:

Topic	Details
Typical use	The skeleton model is typically used when you know the methods you want to expose as a Web Service, but don't yet have an implementation of them. In this case, you tell the Web Service Wizard about these methods by providing a Java remote interface or WSDL file as input, then implement them later in the context of the generated Web Service files.
How it works	In the skeleton model, you implement your Web Service methods by subclassing the **servlet** that the wizard generates to handle HTTP SOAP processing. As a result, the same class that supports the logistics of your Web Service also processes the method calls.

Topic	Details
Advantages	The skeleton model is relatively simple, involving fewer classes to understand and maintain. At runtime, having less object overhead may also offer performance benefits.
How to generate it	When you specify class-generation and SOAP options in the Web Service Wizard, check both of these items: ◆ Generate skeletons ◆ Not tie-based
Files generated	If you start with a **Java remote interface**, the wizard generates: ◆ *xxx*_ServiceSkeleton.java If you start with a **WSDL file**, the wizard generates: ◆ *xxx*.java (remote interface) ◆ *xxx*_ServiceSkeleton.java
File you add	When the wizard is done, you must code a class that extends the generated servlet *xxx*_ServiceSkeleton and implements the remote interface for your Web Service. You'll use this manually coded class as the servlet for the Web Service.

Scenario: starting with a Java class

In this scenario, you'll see how the Web Service Wizard can be used to generate a Web Service based on an existing Java class that implements the methods to expose:

- Project setup
- Input to the wizard
- Generated files for the Web Service
- Generated files for testing
- Deployment descriptor
- Runtime test result

Implementation model This scenario illustrates use of the **tie** model. For an overview of that architecture, see "Choosing an implementation model" on page 149.

Project setup

The WAR project for this scenario is set up as follows:

- The **name** of this project is:
  ```
  WebServiceSample.spf
  ```
- The **archive** resulting from this project will be:
  ```
  WebServiceSample.war
  ```
- The **initial content** of this project is:
  ```
  WEB-INF
    lib
      wssdk.jar
    classes
      com
        exsamp
          obj
            MyObject.java
    web.xml
  ```

- The **classpath** needed for this project is:

```
...\WEB-INF\lib\wssdk.jar
...\exteNdn\tools\compilelib\j2ee_api_1_n.jar
```

Input to the wizard

Here's the input provided to the Web Service Wizard for this scenario:

- MyObject class
- Project location panel
- Class selection panel
- Method selection panel
- Binding style panel
- Class-generation and SOAP options panel

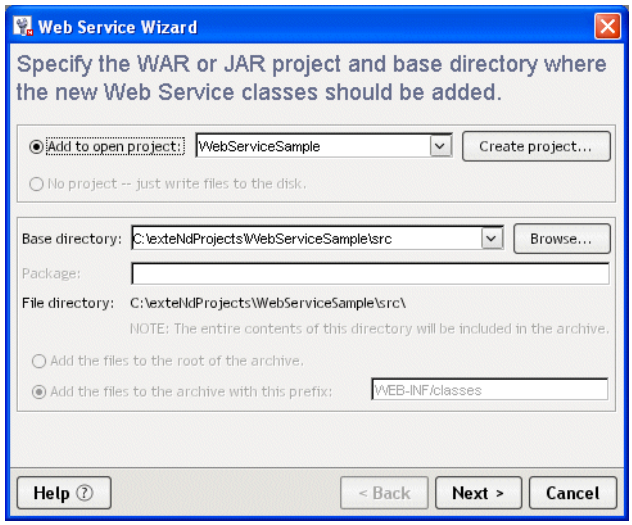
MyObject class

MyObject is an existing Java class from which the Web Service is to be generated. It implements the methods to expose. MyObject.java contains the following code (which must be compiled before you start the wizard):

```
package com.exsamp.obj;

public class MyObject {

    private String s;

    public MyObject() {
    }

    public MyObject(String xxx) {
    }

    public MyObject(String xxx, String yyy) {
    }

    public String getString() {
        return s;
    }

    public boolean setString(String s) {
        this.s = s;
        return true;
    }

    public String sayHello() {
        return "Hello there, I am on the server";
    }
}
```

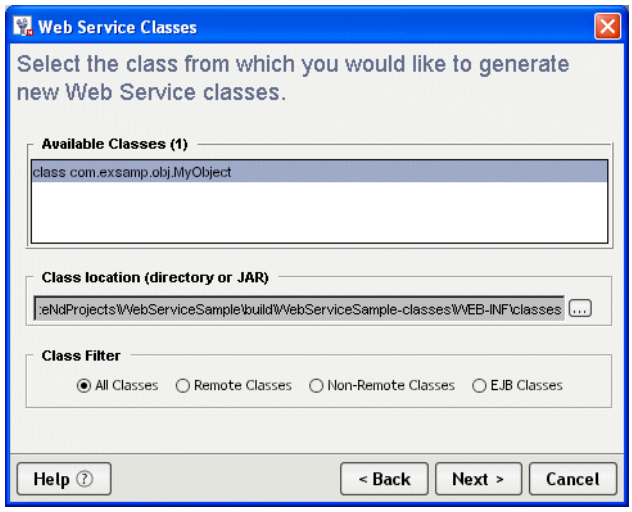
Project location panel

This wizard panel is completed as follows:



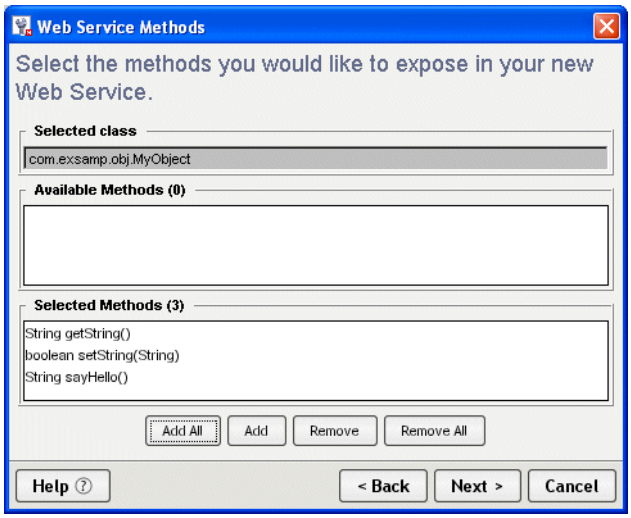
Class selection panel

This wizard panel is completed as follows:



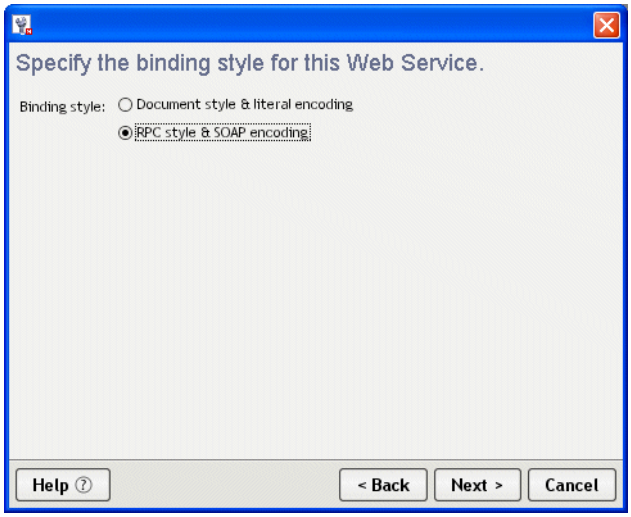
Method selection panel

This wizard panel is completed as follows:



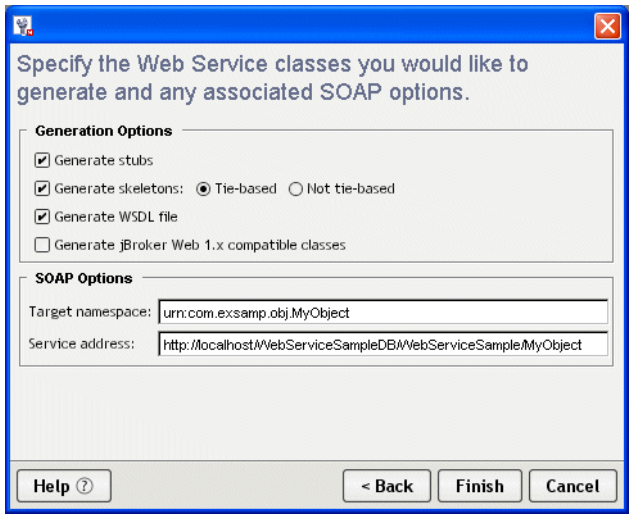
Binding style panel

This wizard panel is completed as follows:



Class-generation and SOAP options panel

This wizard panel is completed as follows:



Generated files for the Web Service

Based on the input provided for this scenario, the Web Service Wizard generates these files to implement the Web Service:

- **MyObjectWS.java** is the remote interface for the Web Service.
- **MyObjectWS_ServiceSkeleton.java** is the abstract servlet class that handles access to the Web Service.
- **MyObjectWS_ServiceTieSkeleton.java** is an abstract class that extends MyObjectWS_ServiceSkeleton to support the tie model.
- **MyObjectWSTie.java** extends the abstract servlet classes to function as the front end for the Web Service. To process requests (method calls) it receives, this servlet instantiates and delegates to MyObjectWSDelegate.
- **MyObjectWSDelegate.java** instantiates the implementation class (MyObject) and makes the requested method calls against that instance.
- **MyObjectWS.wsdl** describes the Web Service in standard WSDL format (useful when publishing to a registry).

Examining MyObjectWS.wsdl

Here's the generated WSDL for this new Web Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyObjectWSService"
 targetNamespace="urn:com.exsamp.obj.MyObject"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tns="urn:com.exsamp.obj.MyObject"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <types/>
 <message name="setStringRequest">
  <part name="arg0" type="xsd:string"/>
 </message>
 <message name="setStringResponse">
  <part name="result" type="xsd:boolean"/>
 </message>
 <message name="getStringRequest"/>
 <message name="getStringResponse">
  <part name="result" type="xsd:string"/>
 </message>
```

```
 <message name="sayHelloRequest"/>
 <message name="sayHelloResponse">
  <part name="result" type="xsd:string"/>
 </message>
 <portType name="MyObjectWS">
  <operation name="setString" parameterOrder="arg0">
   <input message="tns:setStringRequest"/>
   <output message="tns:setStringResponse"/>
  </operation>
  <operation name="getString">
   <input message="tns:getStringRequest"/>
   <output message="tns:getStringResponse"/>
  </operation>
  <operation name="sayHello">
   <input message="tns:sayHelloRequest"/>
   <output message="tns:sayHelloResponse"/>
  </operation>
 </portType>
 <binding name="MyObjectWSBinding" type="tns:MyObjectWS">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="setString">
   <soap:operation soapAction="urn:com.exsamp.obj.MyObject/setString"/>
   <input>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </input>
   <output>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </output>
  </operation>
  <operation name="getString">
   <soap:operation soapAction="urn:com.exsamp.obj.MyObject/getString"/>
   <input>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </input>
   <output>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </output>
  </operation>
  <operation name="sayHello">
   <soap:operation soapAction="urn:com.exsamp.obj.MyObject/sayHello"/>
   <input>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </input>
   <output>
    <soap:body
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:com.exsamp.obj.MyObject" use="encoded"/>
   </output>
  </operation>
 </binding>
 <service name="MyObjectWSService">
  <port binding="tns:MyObjectWSBinding" name="MyObjectWSPort">
   <soap:address location="http://localhost/WebServiceSampleDB/WebServiceSample/MyObject"/>
  </port>
 </service>
</definitions>
```

Generated files for testing

Based on the input provided for this scenario, the Web Service Wizard generates these files so you can test the Web Service once it's deployed:

- **MyObjectWSService.java** is the service interface that's used in JAX-RPC to help clients obtain the stub for the Web Service.
- **MyObjectWSServiceImpl.java** is the service implementation class that handles instantiation of the stub (MyObjectWSBinding_Stub).
- **MyObjectWSBinding_Stub.java** is used by clients as a proxy for accessing the Web Service. This stub class implements the remote interface (MyObjectWS) to handle the logistics of each method call.
- **MyObjectWSClient.java** is a simple client application that accesses the Web Service by:
  - Instantiating MyObjectWSService via JNDI lookup
  - Using the MyObjectWSService object to obtain the stub (MyObjectWSBinding_Stub)
  - Calling Web Service methods via the MyObjectWSBinding_Stub object

Editing MyObjectWSClient.java

The process() method of MyObjectWSClient must be edited to uncomment the Web Service method call to be tested. Here's the change:

```
// System.out.println("Test Result = " + remote.getString());
// System.out.println("Test Result = " + remote.setString(java.lang.String));
System.out.println("Test Result = " + remote.sayHello());
```

Deployment descriptor

Because this scenario uses the tie model, the Web Service Wizard automatically updates the **web.xml** file to declare MyObjectWSTie as the servlet class to handle requests for the MyObject Web Service:

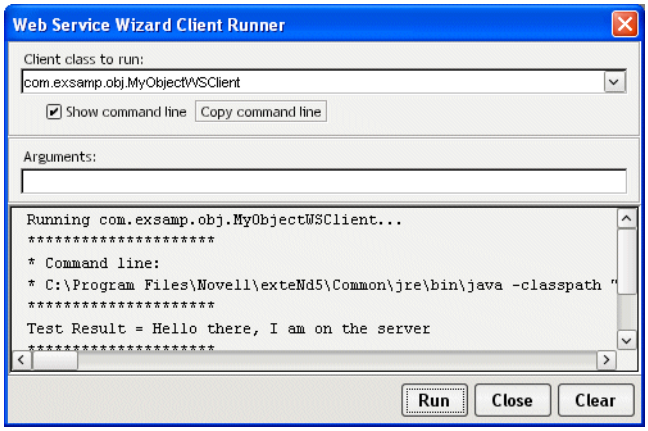
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
                         "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <servlet>
      <servlet-name>MyObject</servlet-name>
      <servlet-class>com.exsamp.obj.MyObjectWSTie</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>MyObject</servlet-name>
      <url-pattern>/MyObject</url-pattern>
   </servlet-mapping>

   <env-entry>
      <env-entry-name>wssdk.test.disable</env-entry-name>
      <env-entry-value>true</env-entry-value>
      <env-entry-type>java.lang.Boolean</env-entry-type>
   </env-entry>
</web-app>
```

Also note that, in this case, the developer has chosen to disable the Web Services SDK browser-based test environment for this Web Service. Instead, the Web Service will be tested by running the generated MyObjectWSClient application.

For more information on the browser-based test environment, see "Testing your Web Service from a browser" on page 149.

Runtime test result

Once this project is built and the WAR file is created and deployed to the J2EE server, the MyObject Web Service is ready for a test run. Here's the result of using the **Client Runner** in the exteNd Director development environment to execute the MyObjectWSClient application:



9 Generating Web Service Consumers

This chapter walks you through the basic steps and a typical scenario for using the **Web Service Wizard** to generate a Web Service consumer (a program that accesses a Web Service). Topics include:

- Basics
- Steps
- Preparing to generate
- Providing a WSDL file
- Generating the consumer files
- Examining the generated files
- Editing the generated files
- Using the generated files
- Running the consumer program

📖 To learn about the steps and scenarios for using the wizard when you want to create a Web Service, see Chapter 8, “Generating Web Services”.

Basics

You can use the Web Service Wizard of the Novell exteNd Director development environment to 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 the consumer program to call the Web Service as a **Java remote object** (using RMI) and invoke its methods.

For input, the wizard requires a **WSDL file** that describes the Web Service to access. It can handle a wide variety of Web Service implementations, including:

- Document-style and RPC-style bindings
- Basic and complex types
- J2EE providers, Microsoft .NET providers, and others

The wizard generates Java source files based on JAX-RPC (Java API for XML-based RPC) and the Novell exteNd Web Services SDK (the JAX-RPC implementation included with Novell 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.

📖 For an introduction to Web Service concepts, standards, and technologies, see Chapter 7, “Web Service Basics”.

📖 For detailed documentation on the wizard, see Chapter 10, “Web Service Wizard”.

Steps

The process of developing your consumer program involves:

1 Preparing to generate by setting up your project

2 Providing a WSDL file that describes the Web Service for which you want the wizard to generate consumer code

3 Generating the consumer files by using the wizard

4 Examining the generated files that the wizard creates, including Java source for:

- A **remote interface**, **service classes**, and a **stub class** that facilitate the Web Service access
- Any **type classes** needed for method arguments and return values
- A **simple Java client class** that uses the other classes to make method calls

5 Editing the generated files to adjust the method calls to make and the Web Service location to point to

6 Using the generated files either as is or by including the consumer code in some other Java application

7 Running the consumer program in your development environment (for testing) and in the production environment

Preparing to generate

To prepare for using the Web Service Wizard, you:

1 Set up an appropriate **project** in the exteNd Director development environment.

The type of project you should create depends on how you ultimately plan to use the consumer code that the wizard will generate. For instance:

If you plan to use the consumer code in	You should create
A standard Java application (perhaps based on the simple Java client class that the wizard generates)	A JAR project
A J2EE application client	A CAR project
A JSP page or servlet	A WAR project
An Enterprise JavaBean	An EJB JAR project

2 Add the **archives required by the Web Services SDK** to your project:

- **wssdk.jar**, which contains the Web Services SDK API classes needed at runtime
- activation.jar
- commons-httpclient.jar
- commons-logging.jar
- CSHelper.jar
- xercesImpl.jar
- xmlParserAPIs.jar
- agrootca.jar (when using SSL and the Novell exteNd trusted root certificates)
- jakarta-regexp-1.2.jar (when using a pre-1.4 JDK and XML Schema with patterns)
- mail.jar (when using attachments and javax.mail.internet.MimeMultipart)
- Phaos_Crypto_FIPS.jar, Phaos_Security_Engine.jar, and Phaos_SSLava.jar (when using SSL; note that these JARs are already on the classpath of the exteNd Application Server)

- xmlsec.jar (when using XML Signature)

You'll find these JARs in the Novell exteNd **tools\compilelib** directory.

3 Edit the **classpath** of your project so you can compile your consumer classes once they're generated and edited. You'll need to include:

- wssdk.jar and the supporting JARs listed in Step 2 (as appropriate)
- Any application-specific entries

For **J2EE projects**, you'll also need j2ee_api_1_*n*.jar (it's included automatically when you create a J2EE project in the exteNd Director development environment).

Providing a WSDL file

To generate consumer code, you'll need to provide the Web Service Wizard with a WSDL file that describes the target Web Service. It's a good idea to obtain the file location or URL of this WSDL file before you start the wizard.

These are common scenarios:

- **For a Web Service developed in your organization**, you might have the WSDL file on your file system or even in your project.
- **For an external Web Service**, you should be able to get the WSDL file's URL from the appropriate Web site or registry.

Example: WSDL file for Autoloan .NET Web Service

Suppose you want to generate consumer code to use the **Autoloan** .NET Web Service, which is listed on the XMethods public registry under the name **Equated Monthly Instalment (EMI) Calculator**. That Web Service calculates and returns the monthly loan payment for a given term (number of months), interest rate, and loan amount.

In this case, you can go to the Web site www.xmethods.net to discover the URL for the corresponding WSDL file:

```
http://upload.eraserver.net/circle24/autoloan.asmx?wsdl
```

When you provide this URL to the Web Service Wizard, it will read the WSDL file to learn what it needs to know about the Autoloan Web Service:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
             xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
             xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:s0="http://circle24.com/webservices/"
             targetNamespace="http://circle24.com/webservices/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema attributeFormDefault="qualified" elementFormDefault="qualified"
              targetNamespace="http://circle24.com/webservices/">
      <s:element name="Calculate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="Months" type="s:double" />
            <s:element minOccurs="1" maxOccurs="1" name="RateOfInterest" type="s:double" />
            <s:element minOccurs="1" maxOccurs="1" name="Amount" type="s:double" />
          </s:sequence>
        </s:complexType>
```

```
    </s:element>
    <s:element name="CalculateResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="CalculateResult" nillable="true"
                     type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="string" nillable="true" type="s:string" />
  </s:schema>
</types>
<message name="CalculateSoapIn">
  <part name="parameters" element="s0:Calculate" />
</message>
<message name="CalculateSoapOut">
  <part name="parameters" element="s0:CalculateResponse" />
</message>
<message name="CalculateHttpGetIn">
  <part name="Months" type="s:string" />
  <part name="RateOfInterest" type="s:string" />
  <part name="Amount" type="s:string" />
</message>
<message name="CalculateHttpGetOut">
  <part name="Body" element="s0:string" />
</message>
<message name="CalculateHttpPostIn">
  <part name="Months" type="s:string" />
  <part name="RateOfInterest" type="s:string" />
  <part name="Amount" type="s:string" />
</message>
<message name="CalculateHttpPostOut">
  <part name="Body" element="s0:string" />
</message>
<portType name="AutoloanSoap">
  <operation name="Calculate">
    <input message="s0:CalculateSoapIn" />
    <output message="s0:CalculateSoapOut" />
  </operation>
</portType>
<portType name="AutoloanHttpGet">
  <operation name="Calculate">
    <input message="s0:CalculateHttpGetIn" />
    <output message="s0:CalculateHttpGetOut" />
  </operation>
</portType>
<portType name="AutoloanHttpPost">
  <operation name="Calculate">
    <input message="s0:CalculateHttpPostIn" />
    <output message="s0:CalculateHttpPostOut" />
  </operation>
</portType>
<binding name="AutoloanSoap" type="s0:AutoloanSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                style="document" />
  <operation name="Calculate">
    <soap:operation soapAction="http://circle24.com/webservices/Calculate"
                    style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

```
  <binding name="AutoloanHttpGet" type="s0:AutoloanHttpGet">
    <http:binding verb="GET" />
    <operation name="Calculate">
      <http:operation location="/Calculate" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <binding name="AutoloanHttpPost" type="s0:AutoloanHttpPost">
    <http:binding verb="POST" />
    <operation name="Calculate">
      <http:operation location="/Calculate" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <service name="Autoloan">
    <documentation>This Web Service mimics a Simple Autoloan calculator.</documentation>
    <port name="AutoloanSoap" binding="s0:AutoloanSoap">
      <soap:address location="http://upload.eraserver.net/circle24/autoloan.asmx" />
    </port>
    <port name="AutoloanHttpGet" binding="s0:AutoloanHttpGet">
      <http:address location="http://upload.eraserver.net/circle24/autoloan.asmx" />
    </port>
    <port name="AutoloanHttpPost" binding="s0:AutoloanHttpPost">
      <http:address location="http://upload.eraserver.net/circle24/autoloan.asmx" />
    </port>
  </service>
</definitions>
```

Understanding the WSDL

In the Autoloan WSDL, you can ignore the definitions for **HttpGet** and **HttpPost** (including message, portType, binding, and service port). Only the **Soap** definitions apply to the Web Service consumer program you're developing.

Notice that this Web Service exposes one method named **calculate()**. It takes a **Calculate** object containing three doubles (Months, RateOfInterest, and Amount) and returns a **CalculateResponse** object containing one string (CalculateResult). The Web Service Wizard will generate a corresponding remote interface in Java to support calling this method.

The **types** section specifies the **XML Schema** definitions for Calculate and CalculateResponse. The Web Service Wizard will generate corresponding type classes in Java to represent these objects.

There's a twist in how those Calculate and CalculateResponse classes will be used in the generated Autoloan consumer code. The Autoloan WSDL happens to follow the **wrapper** pattern defined by JAX-RPC, which dictates that method parameters in the remote interface should use **basic types** directly rather than the objects that wrap them. As a result, the calculate() method will be generated to take the **three doubles** (instead of Calculate) and return the **string** (instead of CalculateResponse). The wrapping and unwrapping of Calculate and CalculateResponse will be handled under the covers by the generated Autoloan stub class.

If you look in the **binding** section for AutoloanSoap, you'll see that this Web Service is defined as **document style** (as opposed to **RPC style**). That's typical of .NET Web Services. Binding style describes the format of SOAP messages and can affect interoperability with other Web Service environments:

Binding style	What it means
Document (with literal use)	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.
RPC (with encoded use)	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.

The Web Service Wizard will generate the Java code needed to handle the specified binding style.

The **port** definition for AutoloanSoap (at the end of the WSDL file) specifies the **address** (URL) where the Web Service can be accessed:

```
http://upload.eraserver.net/circle24/autoloan.asmx
```

The Web Service Wizard will use this URL in the service and stub classes it generates for calling the Web Service.

Generating the consumer files

Once you've set up your project and located the appropriate WSDL file, you're ready to use the Web Service Wizard. The wizard produces one Web Service consumer at a time, so you'll need to use it multiple times if you have several to develop.

Each time you launch the wizard, it uses the WSDL file and other input you provide to generate a set of consumer source files. Here's a summary of the process:

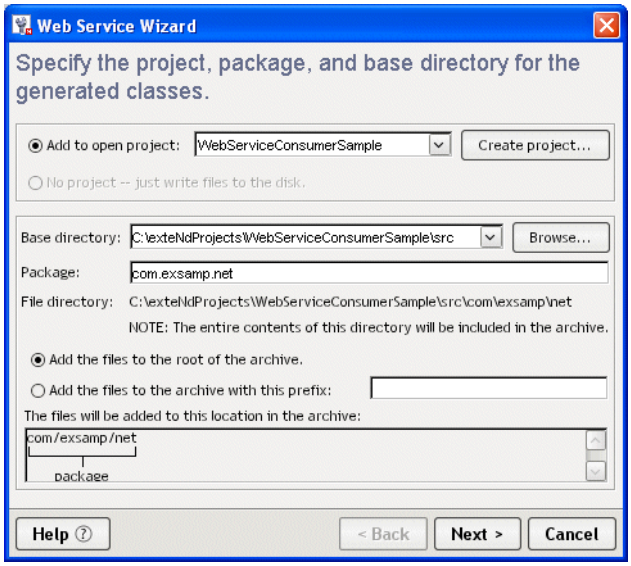
1 Select **File>New>File** to display the New File dialog and go to the **Web Services** tab.

2 Launch the Web Service Wizard by selecting **Existing Web Service**.

3 When the wizard prompts you for **project location** information, specify:

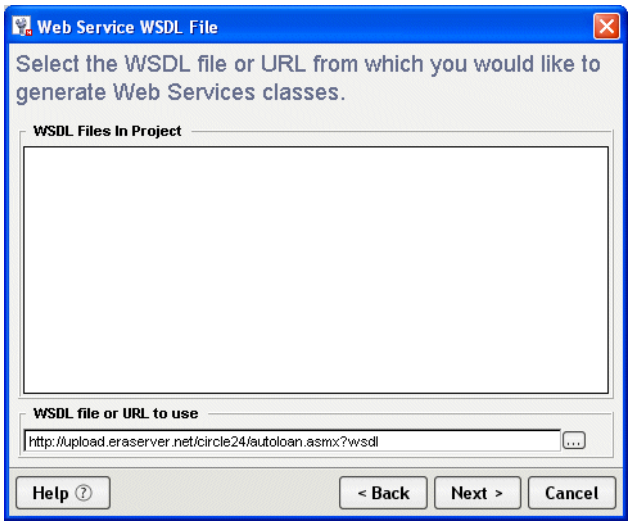
- The **project** you set up to contain the generated Web Service consumer files
- The target **directory and package** in that project

For example, suppose you're generating a consumer for the Autoloan Web Service. You might specify WebServiceConsumerSample as the target JAR project and com.exsamp.net as the package for generated classes:



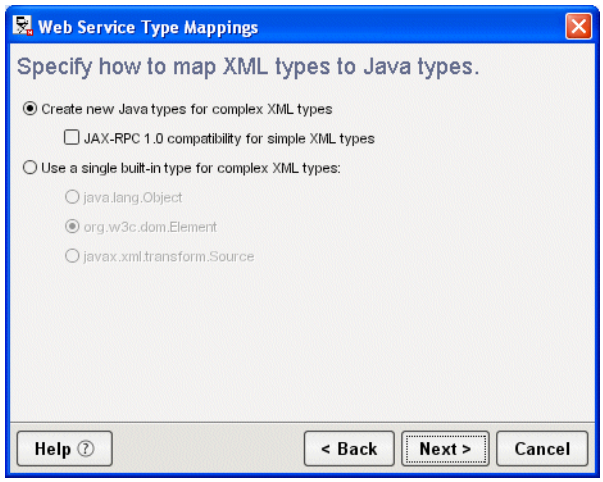
4 When the wizard prompts you, specify the **WSDL file** that describes your target Web Service.

For example, when generating a consumer for the Autoloan Web Service, you specify the WSDL file URL obtained from the XMethods public registry:



5 When the wizard prompts you for **Web Service type mappings**, specify how data types defined in the WSDL (via XML Schema) are to be represented in the generated Java code.

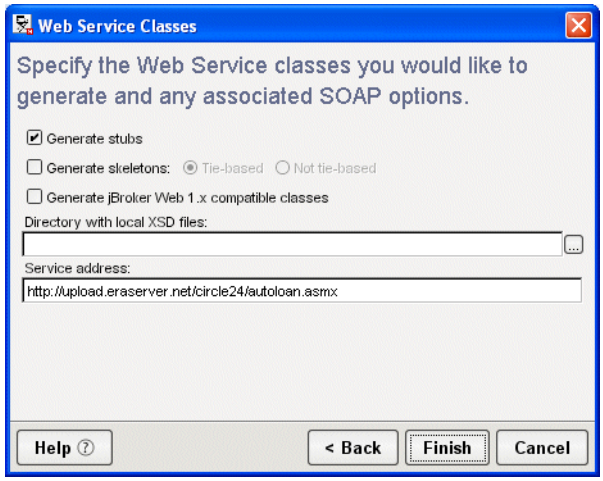
The default is to create specific **Java types** for all of the **complex XML types** in the WSDL. That's usually appropriate and it's the right choice here for the Autoloan Web Service example:



6 When the wizard prompts you for **class-generation and SOAP options**, you must specify details about the code to create:

- To get the files needed for a Web Service consumer, check **Generate stubs** (and leave **Generate skeletons** unchecked).
- If necessary, you can override the default **Service address** (obtained from the WSDL) by specifying a different URL for the Web Service.

For example, these options will generate the appropriate consumer source files for the Autoloan Web Service:



NOTE: Support for jBroker Web 1.*x* applications is available via a **backward-compatibility** option. For more information, see If you choose jBroker Web 1.x compatibility (in the preceding chapter).

7 Click **Finish** when you're done specifying options for the Web Service consumer.

Examining the generated files

Once you finish the wizard, it generates everything you've specified for your Web Service consumer and updates other parts of your project with supporting changes:

What the wizard generates	Details
Java source file for remote interface	***xxx*.java** An interface that extends java.rmi.Remote and declares the methods exposed by the target Web Service (as determined from the WSDL file). The generated stub class *xxx*_Stub implements this interface to support method calls for the Web Service.
Java source files for stubs	***xxx*Service.java** Service interface used by JAX-RPC clients to obtain the stub for the target Web Service. ***xxx*ServiceImpl.java** Service implementation class that handles instantiation of the stub (*xxx*_Stub). It also supports alternative ways of accessing the target Web Service, including dynamic (stubless) calls. (Note that the names generated for the service interface and implementation class depend on your WSDL and may omit the text **Service**.) ***xxx*_Stub.java** Facilitates method calls from a Java-based consumer to the target Web Service. *xxx*_Stub implements the generated remote interface by sending an appropriate HTTP SOAP request for each method call. ***xxx*Client.java** Simple client application that works as a consumer of the target Web Service. It obtains the stub (via the Service object) and then uses the stub to call Web Service methods. You can run *xxx*Client from the exteNd Director development environment (select **Tools**>**Run Web Service Client Class**) or from a command line.
Updates to project contents	The wizard updates your project to add generated files to it.

About generated file names

When generating file names, the Web Service Wizard follows the naming rules specified by JAX-RPC. For a Web Service consumer, the resulting file names are based on the definitions in the WSDL.

For simplicity, this documentation uses *xxx* to represent the portion of a generated Web Service consumer file name that's derived from a WSDL definition.

Additional details of generation

Under the covers, the Web Service Wizard uses the **Web Services SDK compilers** when generating the Web Service consumer files listed above. In some cases, these compilers may generate additional code or files to support requirements specific to your application, such as:

- Type mapping
- Faults
- Multiple portType definitions

📖 For more information, see the Web Services SDK help.

Example: generated consumer files for Autoloan .NET Web Service

The consumer code that the Web Service Wizard generates for the Autoloan Web Service consists of these **standard files for Web Service access**:

- **AutoloanSoap.java** is the remote interface used by the stub class to support method calls for the Autoloan Web Service.
- **Autoloan.java** is the service interface that's used in JAX-RPC to help clients obtain the stub for the Web Service.
- **AutoloanImpl.java** is the service implementation class that handles instantiation of the stub (AutoloanSoap_Stub).
- **AutoloanSoap_Stub.java** is the stub class. It passes method calls to the Autoloan Web Service as HTTP SOAP requests.
- **AutoloanSoapClient.java** is a simple client application that obtains the stub (via the Service object) and then uses it to call the calculate() method of the Autoloan Web Service. (Note that this method call is generated as a comment. You'll learn what to do with it a little later, in "Editing the generated files" on page 169.)

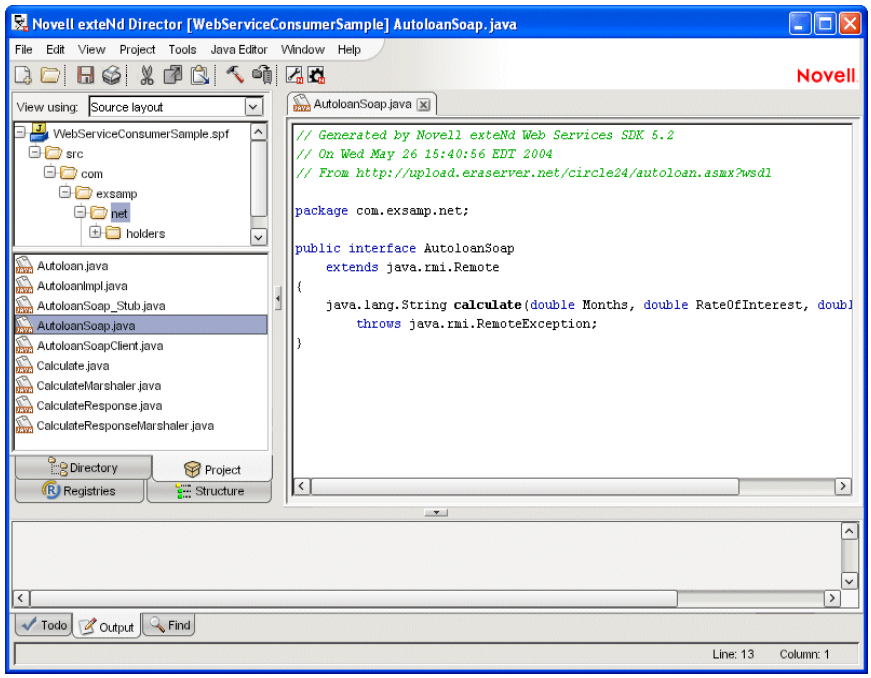
And these **application-specific files for mapping the complex types** defined in the WSDL:

- **Calculate.java** is a class that represents the complex type Calculate that's defined in the WSDL.
- **CalculateMarshaler.java** is a class that handles serialization and deserialization for Calculate.
- **CalculateHolder.java** is the Holder class required by JAX-RPC to implement type mapping support for Calculate. Note that this class is generated in the **holders** subdirectory.
- **CalculateResponse.java** is a class that represents the complex type CalculateResponse that's defined in the WSDL.
- **CalculateResponseMarshaler.java** is a class that handles serialization and deserialization for CalculateResponse.
- **CalculateResponseHolder.java** is the Holder class required by JAX-RPC to implement type mapping support for CalculateResponse. Note that this class is generated in the **holders** subdirectory.
- **autoloan.asmx.xmlrpc.type.mappings** specifies settings that tell the Web Services SDK how to configure the type mappings for Calculate and CalculateResponse. These mappings apply when data is converted from XML to Java or vice versa.

  Since the generated stub and service classes automatically configure the mappings, this mappings file is not typically needed. It is provided for special situations (such as when you want to override a mapping).

  The mappings file is generated in the base directory of the source tree (**src**).

When creating these files, the wizard adds them to your project on the directory path you've specified:



Editing the generated files

Follow these guidelines when editing the files generated by the Web Service Wizard:

Guideline	Details
File you **must** edit	◆ *xxx*Client.java
Files you should **not** edit	◆ *xxx*Service.java ◆ *xxx*ServiceImpl.java ◆ *xxx*_Stub.java

It's OK to edit any of the other generated files, but not typically required.

Editing the xxxClient.java file

Before using the generated *xxx*Client.java file, you:

- **Must edit the process() method** to call one or more methods of the target Web Service
- **May need to edit the getRemote() method** to specify the correct location (binding) for accessing the target Web Service

process() method

The process() method is where the generated client application calls methods of the Web Service. Here you'll find commented code for calling each method defined in the generated remote interface and displaying return values on the console. For example:

```
public void process(String[] args) throws Exception
{
    AutoloanSoap remote = getRemote(args);

    // The following code has been generated for your testing convenience. In
    // order to successfully test your Web Service, you must uncomment one or
    // more of these lines and supply meaningful arguments where necessary.
    // Once you have modified the test method(s) below, compile this class and
    // execute it from a command line with your class path set appropriately.

    // System.out.println("Test Result = " + remote.calculate(double, double, double));

}
```

You need to modify this code as follows:

1. **Uncomment one or more method calls** you want to execute.
2. **Provide appropriate arguments** for each method call, either as hardcoded values or as parameters to be furnished at runtime. For runtime arguments, you may also want to add code that validates the values supplied.
3. **Check the return data type** to make sure it can be converted using toString(). If not, use an alternative to System.out.println for displaying the data returned.

Here's what the line with the calculate() method call looks like after editing:

```
System.out.println("Autoloan Web Service\n  " +
  "Loan input data:\n    24 months, 8%, $15000\n  " +
  "Output from the Web Service:\n    " +
  remote.calculate(24, 8, 15000));
```

getRemote() method

This section explains the basic use of the getRemote() method and how to modify it when you need to specify binding information.

Basic use The getRemote() method is where the generated client application obtains the remote object to handle its method calls to the Web Service. That remote object is an instance of the generated stub class (*xxx*_Stub). To create the stub instance, getRemote() does the following:

1. **Instantiates the Service object** (from the service interface and implementation classes, *xxx*Service and *xxx*ServiceImpl) via JNDI lookup
2. **Calls a method** that the Service object provides (in the service interface) to get the stub

Here's an example of the typical code generated for getRemote(). Normally, you don't need to edit it:

```
public AutoloanSoap getRemote(String[] args) throws Exception
{
    InitialContext ctx = new InitialContext();

    String lookup = "xmlrpc:soap:com.exsamp.net.Autoloan";
    Autoloan service = (Autoloan)ctx.lookup(lookup);
    AutoloanSoap remote = (AutoloanSoap)service.getAutoloanSoap();

    return remote;
}
```

Specifying binding information The wizard includes the binding information for your target Web Service in the generated stub class (*xxx*_Stub.java) and service implementation class (*xxx*ServiceImpl.java). The binding provides the **service endpoint address** where the Web Service can be accessed. In a WSDL file, this address is the URL in the **soap:address location** element.

As an alternative, you can specify the binding to use when creating the stub instance in the getRemote() method. This enables you to override the binding in the stub class (such as when the Web Service has moved to a new location). You just need to add a line of code to set the address property for the stub:

```
public AutoloanSoap getRemote(String[] args) throws Exception
{
    InitialContext ctx = new InitialContext();

    String lookup = "xmlrpc:soap:com.exsamp.net.Autoloan";
    Autoloan service = (Autoloan)ctx.lookup(lookup);
    AutoloanSoap remote = (AutoloanSoap)service.getAutoloanSoap();

    ((javax.xml.rpc.Stub)remote)._setProperty("javax.xml.rpc.service.endpoint.address",
      "http://upload.eraserver.net/circle24/autoloan.asmx");

    return remote;
}
```

Using the generated files

How you use the Web Service consumer code that you have at this point depends on the nature of the application you're developing. Sometimes you might want to enhance the generated *xxx*Client.java file and include it in your application. At other times you may just copy syntax from *xxx*Client.java into your own classes. But in either case, you'll always need the generated remote interface, service, and stub files.

Before you start any application-specific coding, it's a good idea to test the basic *xxx*Client to make sure your consumer code works as expected. You'll first need to build your project to compile the source files. Then you can run *xxx*Client, as described in the next section.

Running the consumer program

The generated Web Service consumer program *xxx*Client is a standard Java application. You can run it in either of these ways:

- From the development environment
- From a command line

From the development environment

To help you test your generated client quickly and easily, the exteNd Director development environment provides the **Web Service Wizard Client Runner**. This facility lists the client applications in your current project and lets you select one to execute. For each run, it automatically sets the classpath to include all required files and lets you supply command-line arguments.

➢ **To use the Client Runner:**

1. Open the **project** that contains the compiled client class you want to run.
2. Select **Tools**>**Run Web Service Client Class** to display the Client Runner window.
3. Select a client from the **Client class to run** dropdown.

   This dropdown lists every compiled class in your project that has a main() method.

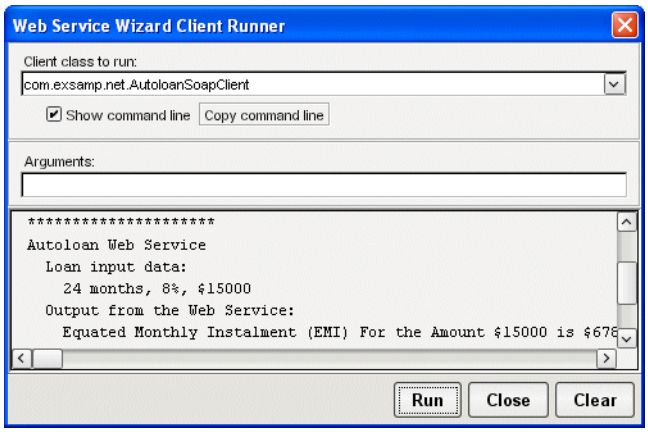
4 Check **Show command line** if you want to:

- See the complete command line that the Client Runner uses to execute your client (it will appear in the display console portion of the window after you click Run)
- Optionally copy that command line to the system clipboard by clicking **Copy command line** (after a run)

5 Type any command-line **Arguments** required by your client (use a space to separate each argument).

6 Click **Run** to execute your client and see its output in the display console portion of the window.

For example, here's what it looks like to execute the generated AutoloanSoapClient class using the Client Runner:



When AutoloanSoapClient runs, it calls the calculate() method of the Autoloan Web Service and passes a Calculate object containing loan data (term, rate, amount). The calculate() method returns a CalculateResponse object containing a string of payment information, which AutoloanSoapClient displays on the screen:

```
Running com.exsamp.net.AutoloanSoapClient...
**********
Autoloan Web Service
  Loan input data:
    24 months, 8%, $15000
  Output from the Web Service:
    Equated Monthly Instalment (EMI) For the Amount $15000 is $678
**********
```

From a command line

You can also execute the generated client from the command prompt of your operating system. Doing so demands that you set the classpath to include all required files (such as the generated consumer classes, wssdk.jar, and so on).

The recommended approach is to use the Web Service Wizard Client Runner to display and copy the command line for your client (as described in the preceding section). Then you can paste that line to your command prompt and run it.

If you plan to run the client on other computers (beyond your development machine), make sure they have access to all of the files listed in this command line.

10 Web Service Wizard

This chapter describes the Web Service Wizard of the Novell exteNd Director development environment, 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 Chapter 7, "Web Service Basics".

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 the Novell exteNd Web Services SDK (the JAX-RPC implementation included with Novell 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 the Web Services SDK)	A Java remote interface from a WSDL file
xsd2java (from the Web Services SDK)	Type classes (JavaBeans, marshalers) and mapping files from complex types defined in a WSDL file's XML Schema
rmi2soap (from the Web Services SDK)	Skeleton and tie classes, stub and service classes, as well as marshalers (for complex types) from a Java remote interface
rmi2wsdl (from the Web Services SDK)	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 Web Services SDK 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: ◆ A JavaBean or other Java class ◆ An EJB session bean ◆ A Java remote interface ◆ A WSDL file	Chapter 8, "Generating Web Services"
Using the wizard to create code for accessing an **existing Web Service** based on its WSDL file	Chapter 9, "Generating Web Service Consumers"

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	**1** Project location (and possibly WAR project selection) **2** Class selection **3** Method selection **4** Binding style (and possibly Schema information) **5** Class-generation and SOAP options
The home interface of an EJB session bean	**1** Project location (and possibly WAR project selection) **2** Class selection **3** EJB lookup information **4** Binding style (and possibly Schema information) **5** Class-generation and SOAP options
The remote interface of an EJB session bean or the SessionBean class itself	**1** Project location (and possibly WAR project selection) **2** Class selection **3** EJB home interface selection **4** EJB lookup information **5** Binding style (and possibly Schema information) **6** Class-generation and SOAP options

If you start with	You step through these panels
A Java remote interface	**1** Project location (and possibly WAR project selection) **2** Class selection **3** Binding style (and possibly Schema information) **4** Class-generation and SOAP options
A WSDL file	**1** Project location **2** WSDL file selection (and possibly Multiple namespace mapping) **3** Web Service type mappings **4** Class-generation and SOAP options

Panel details

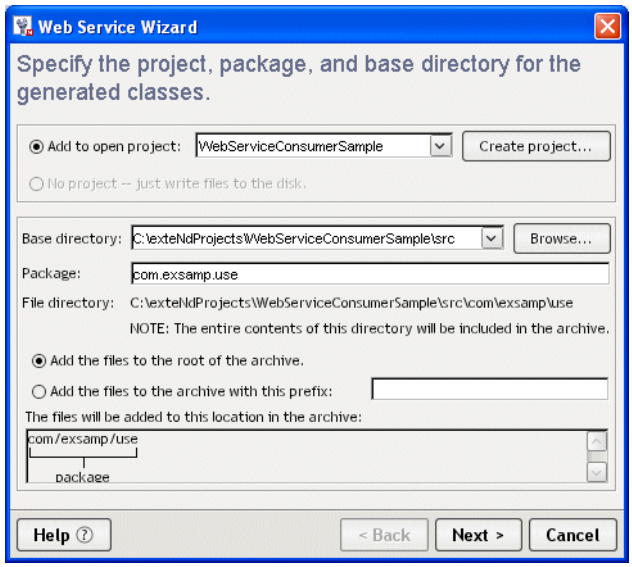
This section describes the options on each panel of the Web Service Wizard. The panels are:

- Project location
- WAR project selection
- Class selection
- WSDL file selection
- Multiple namespace mapping
- Web Service type mappings
- EJB home interface selection
- EJB lookup information
- Method selection
- Binding style
- Schema information
- Class-generation and SOAP options

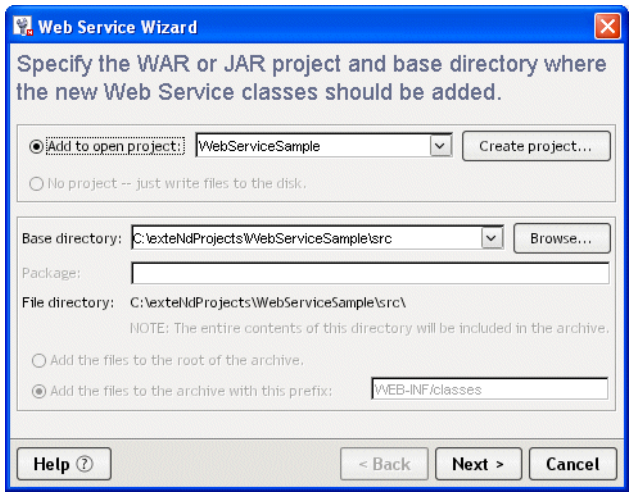
Project location

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	Select a project where the wizard is to store generated files. This option lets you choose from a list of the projects currently open. **If you're generating a Web Service**, you'll typically select a WAR project. 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 178. (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.) **If you're generating a Web Service consumer**, you can select any type of project.
Create project	Click this button if you want to create a new project to use. It displays the New Project dialog. 📖 See "Creating projects and subprojects" on page 50.
No project -- just write files to the disk	(This option is disabled. In the Web Service Wizard, generated files must be added to an open project.)

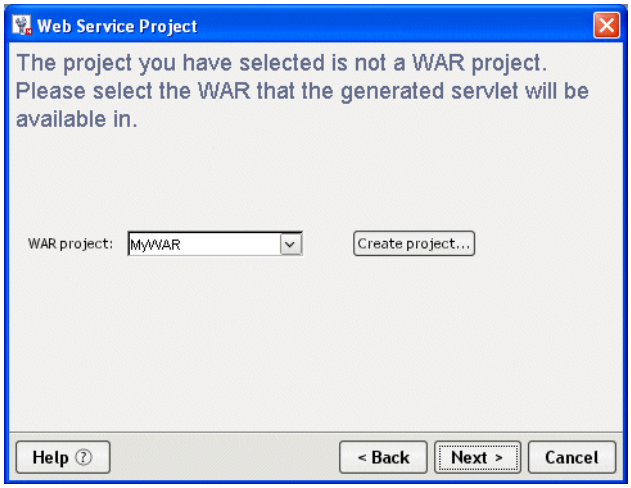
2 Specify the **directory and package**:

Option	What to do
Base directory	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**.
Package	(If enabled) Specify the fully qualified Java package name to be used for generated classes (for example, com.myco.mypkg).
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 that's currently open.
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 50.

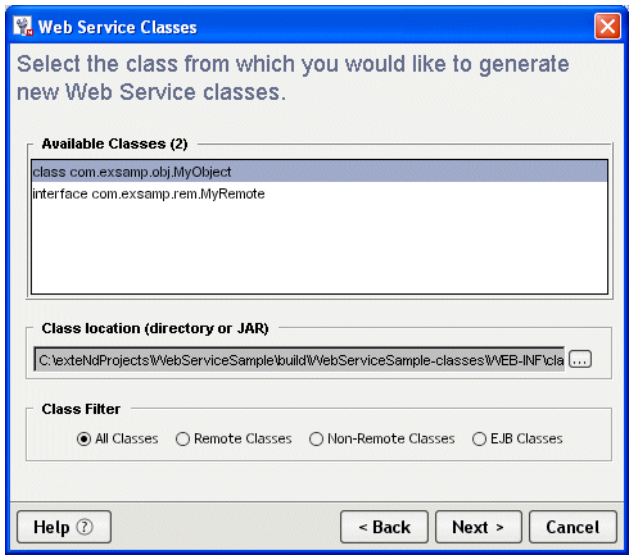
2 Click **Next**.

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

NOTE: This panel won't let you select an EJB or remote interface that defines overloaded method names. **Overloaded method names are not allowed** in Web Service interfaces (as of WSDL 1.2). You must remove them from your class before starting the wizard.



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:**

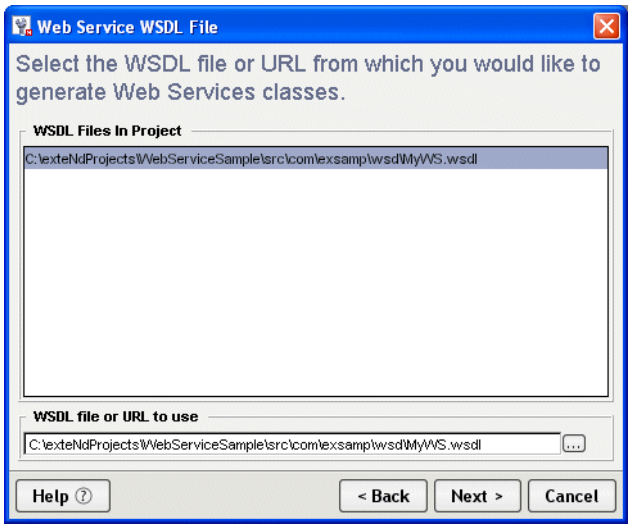
- Click the browse (...) button for **Class location (directory or JAR)** to select a different directory or JAR file.

  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: This panel won't let you select a WSDL file that defines overloaded method names. **Overloaded method names are not allowed** in Web Service interfaces (as of WSDL 1.2). You must remove them from your WSDL file before starting the wizard.



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:

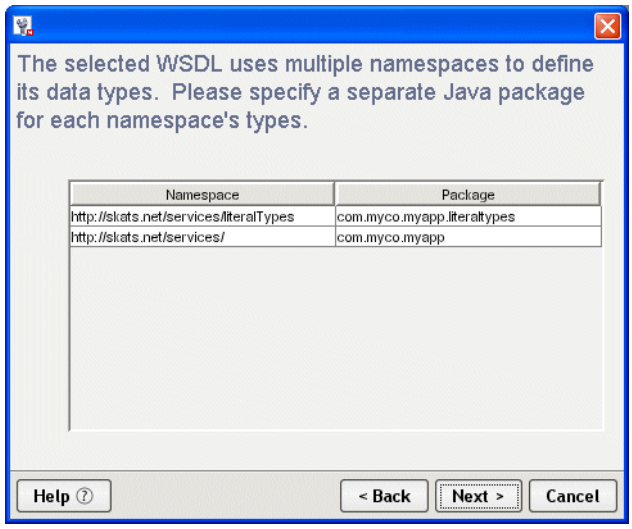
```
http://upload.eraserver.net/circle24/autoloan.asmx?wsdl
```

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 **Create new Java types for complex XML types** is selected on the Web Service type mappings panel.



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.

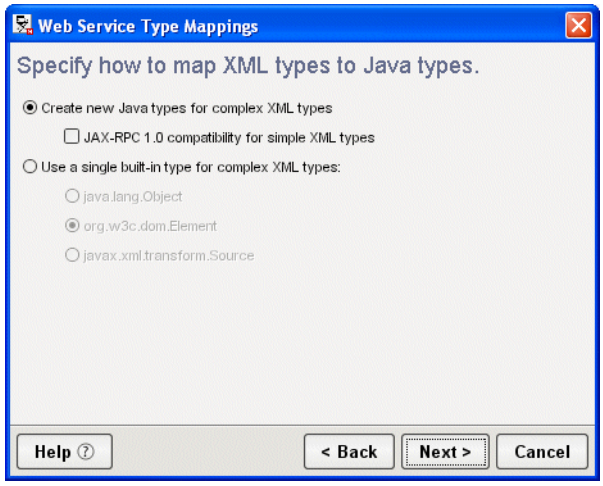
➢ **To edit the namespace-to-package mappings:**

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**.

Web Service type mappings

This panel is used to specify how data types are to be mapped when you're generating from a WSDL file. It provides choices for handling the complex types and simple types defined in the WSDL (via XML Schema).



➢ **To specify how data types are to be mapped:**

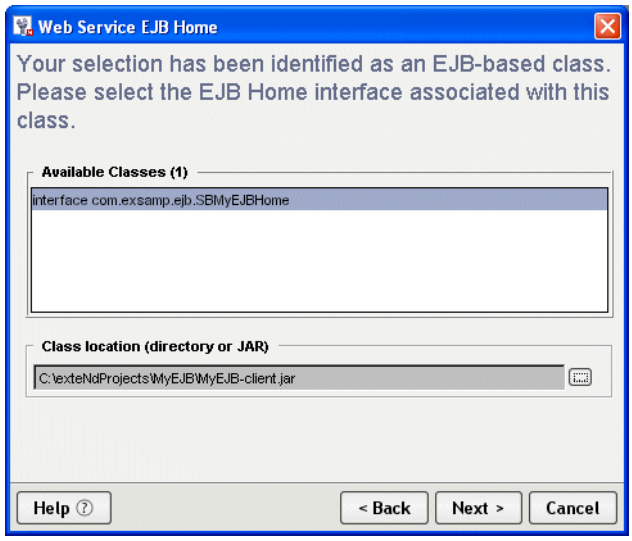
1 Do one of the following:

If you want to	Do this
Map the complex types defined in the selected WSDL file to specific Java types	**1** Select the radio button **Create new Java types for complex XML types**. If the WSDL file specifies RPC as the binding style, this radio button is automatically selected; you can't unselect it. **2** Uncheck the option **JAX-RPC 1.0 compatibility for simple XML types** unless you need it to avoid changing code in an existing project. Checking this option tells the wizard to handle simple types (and any affected complex types) the way it did prior to exteNd 5.2 (when the Web Services SDK followed JAX-RPC 1.0). That means it will generate a JavaBean for each simple type. Unchecking this option tells the wizard to handle simple types according to the latest supported JAX-RPC specification. That means simple types will be mapped directly to the most basic types they specify. If you're regenerating an existing project that includes simple type JavaBeans and you don't want to disrupt code that accesses those JavaBeans, you should check this option. 📖 For more information, see the Web Services SDK help.
Map all complex types to one selected built-in type that lets you access the Web Service data as XML (instead of Java classes)	**1** Select the radio button **Use a single built-in type for complex XML types**. **2** Select one of the following radio buttons to specify which built-in type to use: ◆ **java.lang.Object** ◆ **org.w3c.dom.Element** ◆ **javax.xml.transform.Source**

2 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.

2 Click **Next**.

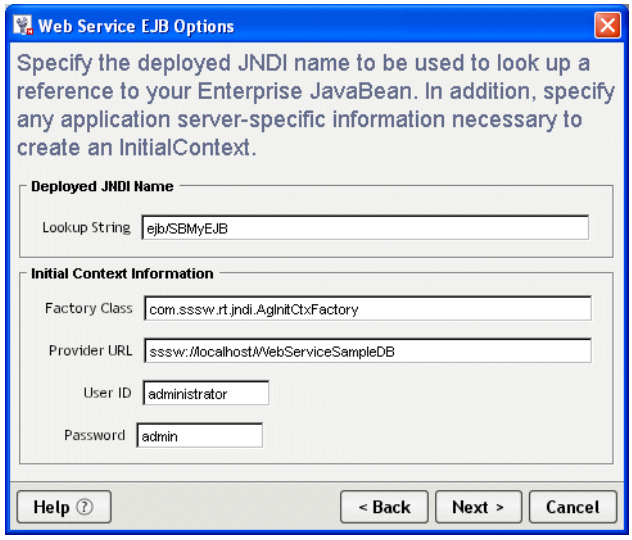
➢ **To list classes from another location:**

- 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.)



This panel displays default initial context values appropriate for looking up a session bean deployed to the Novell 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 ***xxx*Delegate.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.

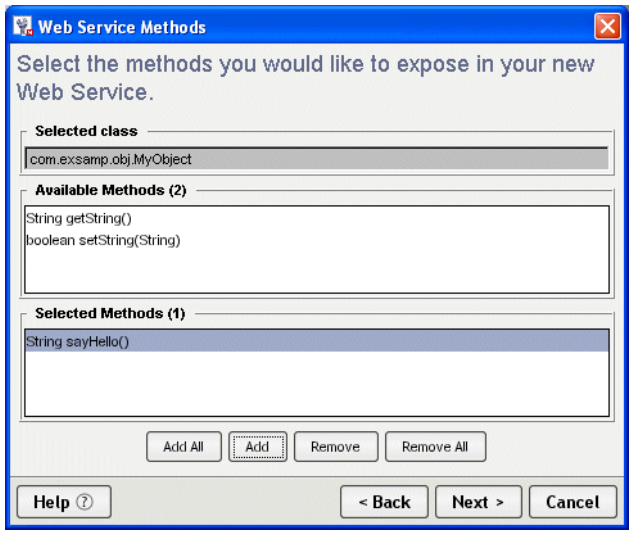
The wizard includes this information in the **servlet** declaration it generates within the deployment descriptor **web.xml**. To learn how these values are used at runtime, see the **getInitialContext()** method of ***xxx*Delegate.java**.

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.

NOTE: Overloaded method names are not allowed in Web Service interfaces (as of WSDL 1.2). As a result, this panel lets you select **no more than one method with a given name**.



This panel examines the selected class and lists its eligible methods in the Available Methods box.

➢ **To select methods to expose:**

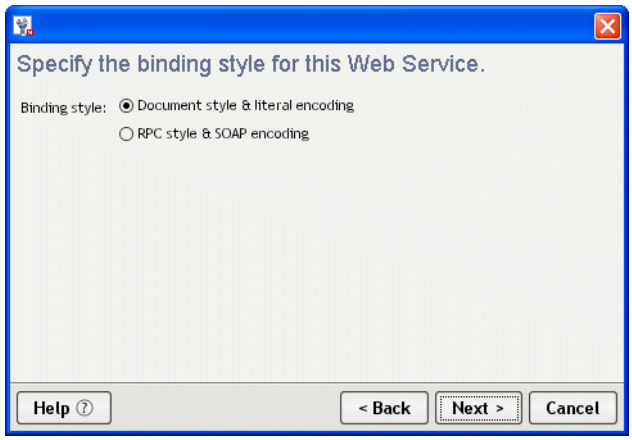
1 Use the **Add** and **Add All** buttons to move one or more items from Available Methods to Selected Methods.

If necessary, you can use **Remove** and **Remove All** to move one or more items back.

2 Click **Next**.

Binding style

This panel is used to specify the binding style when generating a Web Service from a JavaBean, Java class, EJB session bean, or Java remote interface.



➢ **To specify the binding style:**

1 Choose one of the following:

- **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
- **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

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.

2 Click **Next**.

Schema information

These panels are used to specify XML Schema information to include in the WSDL file you generate for a Web Service. The Web Service Wizard prompts for this information if:

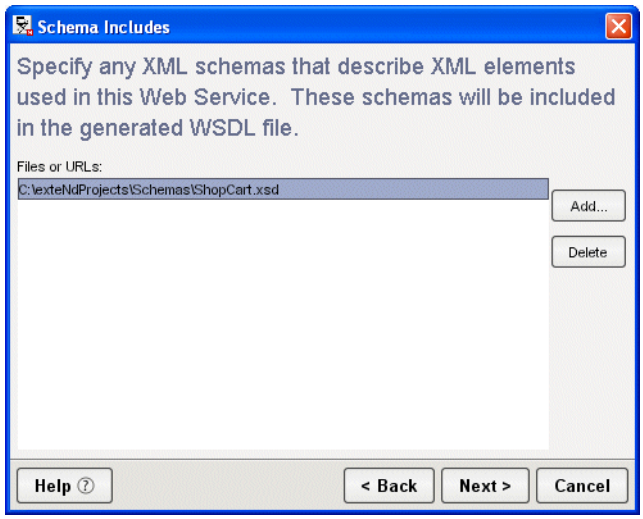
- You're generating from a **class** (JavaBean, Java class, EJB session bean, or Java remote interface) that defines method parameters (arguments or return values) of type **org.w3c.dom.Element**

  AND

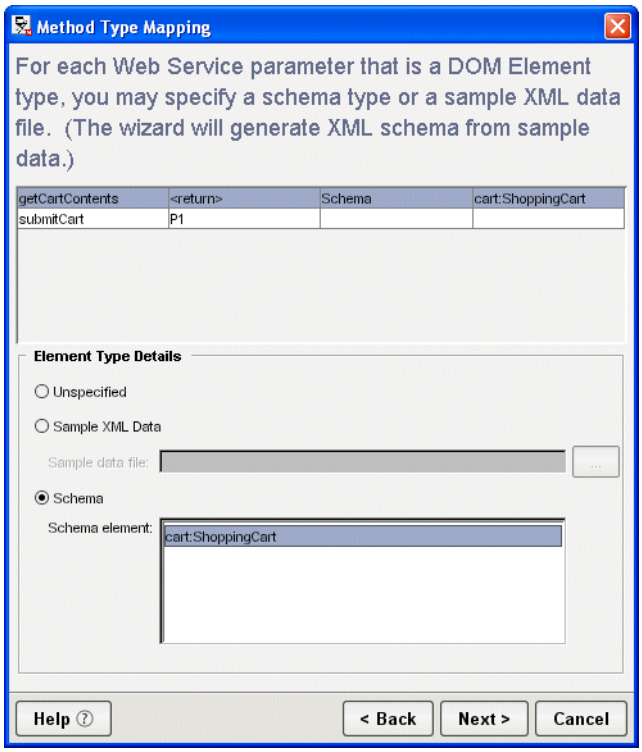
- You specified the **Document style & literal encoding** option on the Binding style panel

There are two related Schema information panels:

- On the **Schema includes** panel, you specify the path or URL of zero or more Schema (XSD) files to include in the generated WSDL.



- On the **Method type mapping** panel, you specify how each Element method parameter maps to a qualified name in one of those Schemas.



As an alternative, you can map a parameter to a sample XML **instance data file**. In that case, the wizard generates the Schema information for you based on the instance data.

NOTE: The information on these panels is used only if the option **Generate WSDL file** is checked on the Class-generation and SOAP options panel.

➢ **To specify Schema includes:**

1 Use the **Add** button to specify the path or URL of each Schema (XSD) file to include in the list. If necessary, you can use the **Delete** button to remove one or more selected files from the list.

2 Click **Next**.

➢ **To specify method type mapping:**

1 For each method parameter listed on the panel, do one of the following:

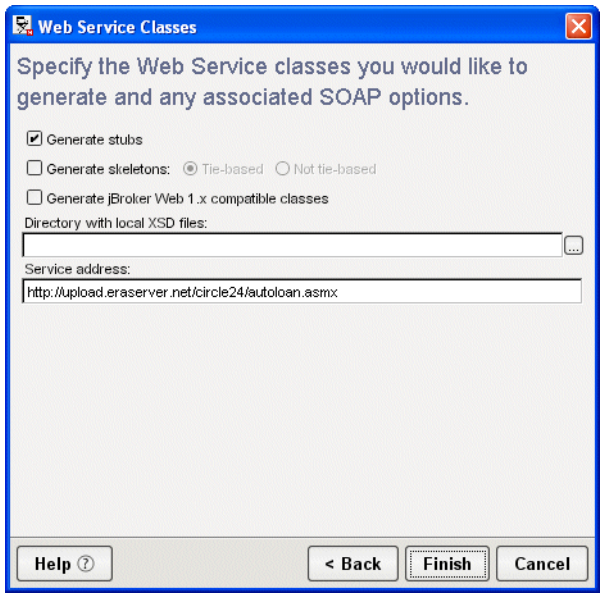
If you want to	Do this
Map a method parameter to a qualified name in a listed Schema	**1** Select the row of information for that parameter. **2** In the Element Type Details section, select the **Schema** radio button. **3** Select a qualified name from the **Schema element** list.
Map a method parameter to an instance data file	**1** Select the row of information for that parameter. **2** In the Element Type Details section, select the **Sample XML Data** radio button. **3** Use the browse (...) button to select the **XML file** containing the appropriate instance data.
Skip mapping a method parameter	**1** Select the row of information for that parameter. **2** In the Element Type Details section, select the **Unspecified** radio button. The parameter will be represented as type **any** in the generated WSDL.

2 Click **Next**.

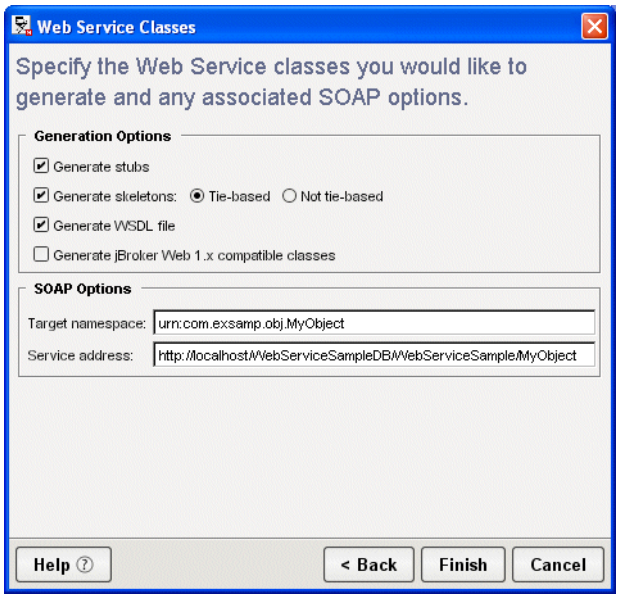
Class-generation and SOAP options

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:



Only this variation provides the ability to generate a WSDL file.

NOTE: When you create a new Web Service from WSDL, the Web Service Wizard does not generate a new or updated copy of that WSDL file for you. That means you must **edit the WSDL yourself** if you want it to reflect changes you've specified via the wizard (such as an override of the default Service address URL).

➢ **To complete this panel:**

1 Specify **Generation Options**:

Option	What to do
Generate stubs	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: ◆ *xxx*Service.java ◆ *xxx*ServiceImpl.java ◆ *xxx*_Stub.java ◆ *xxx*Client.java
Generate skeletons	Check this option to generate classes for implementing the Web Service. Then choose one of these implementation models: ◆ **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: ◆ *xxx*_ServiceSkeleton.java ◆ *xxx*_ServiceTieSkeleton.java ◆ *xxx*Tie.java 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): ◆ *xxx*Delegate.java ◆ **Not tie-based** Generates just a skeleton servlet class used to handle requests for your Web Service. You'll get the following source file: ◆ *xxx*_ServiceSkeleton.java

Option	What to do
Generate WSDL file	(If displayed) Check this option to generate the following file: ◆ *xxx*.wsdl 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**).
Generate jBroker Web 1.x compatible classes	Check this option to generate the specified files according to the original jBroker Web (Version 1.*x*) conventions for: ◆ File names ◆ Stub access in client code Except for these conventions, the generated files will conform to the latest version of the Web Services SDK. 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). 📖 For details on what this option generates, see "If you choose jBroker Web 1.x compatibility" on page 145.
Directory with local XSD files	(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. 📖 For more information about XSD files, see the WSDL specification.

2 Specify **SOAP Options**:

Option	What to do
Target namespace	(If displayed) 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. 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. When generating a WSDL file, the wizard uses this value for the targetNamespace definition.

Service address	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:
	◆ The **stub class** (*xxx*_Stub.java) and **service implementation class** (*xxx*ServiceImpl.java) use it as the default URL for binding to the Web Service.
	◆ The **WSDL file** (*xxx*.wsdl) uses it as the SOAP address in the service definition.
	If you are generating from WSDL, you can override the default value of Service address (which is obtained from the WSDL) by specifying a different URL in this option.
	If you are generating from Java, the default value for this option includes the name of the selected WAR project and the servlet mapping for the Web Service. For example:
	`http://localhost/WebServiceSample/MyObject`
	If you plan to deploy the Web Service to the Novell exteNd Application Server, you need to insert the name of the target database in the URL:
	`http://localhost/WebServiceSampleDB/WebServiceSample/MyObject`

3 Click **Finish**.

11 WSDL Editor

The WSDL Editor provides a quick and easy way to create, edit, and view WSDL documents. This chapter contains the following topics:

About WSDL

WSDL (**Web Services Description Language**) is a general-purpose XML vocabulary for describing Web Services. Using WSDL, it's 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 interacting online.

📖 For the complete WSDL standard, go to www.w3.org/TR/wsdl.

About the WSDL Editor

The WSDL Editor lets you:

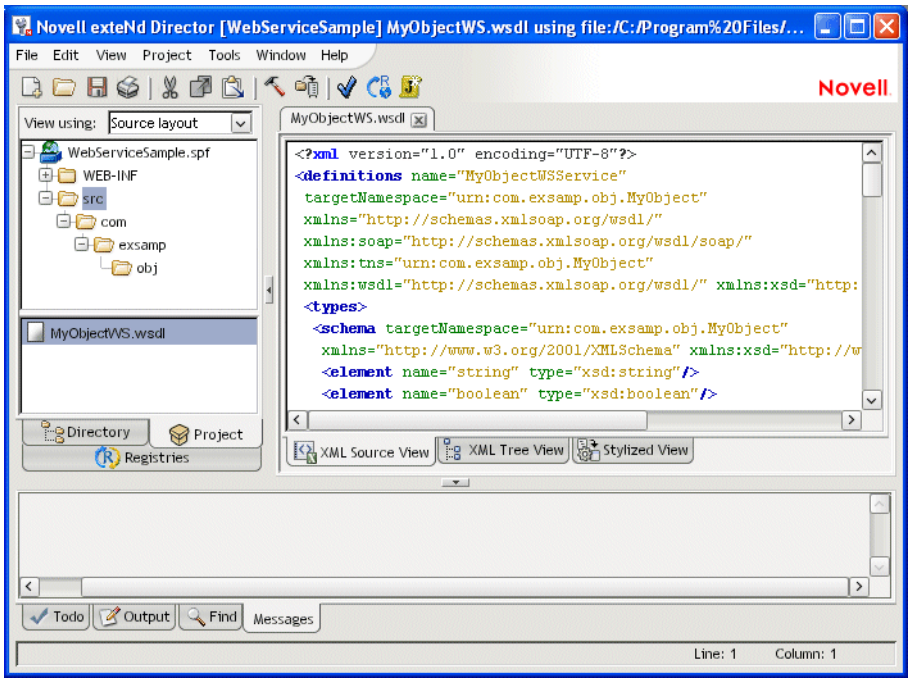
- **Create and edit** WSDL documents (files with the .wsdl extension)
- **Insert** any of the four canonical WSDL document elements (message, port type, binding, or service)
- **Validate** WSDL documents
- **View** WSDL documents in a stylized format
- **Publish** WSDL documents to Web Service registries

Editor features

The WSDL Editor is **based on the XML Editor** and its features, including:

- **XML Source View tab** for text editing of WSDL documents
- **XML Tree View tab** for visual editing of WSDL documents

The WSDL Editor also provides its own **Stylized View tab** for formatted display of WSDL documents.



To learn about using the **basic editing features** of the XML Source View and XML Tree View tabs, see Chapter 4, “XML Editors”.

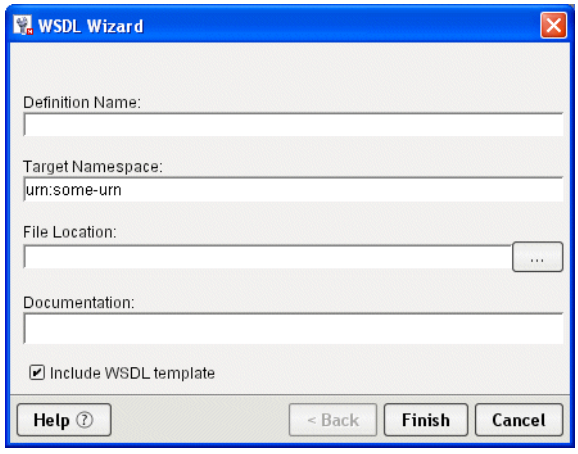
Creating a WSDL document

➢ **To create a WSDL document:**

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

2 On the Web Services tab, select **WSDL** and click **OK**.

The WSDL Wizard displays.



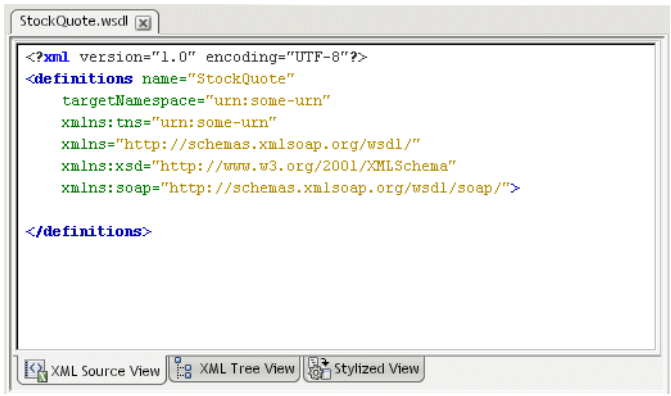
3 Enter a **Definition Name**.

4 Complete the rest of the panel as needed:

- (Optional) Enter a **Target Namespace**. This can be the Uniform Resource Name associated with this WSDL document. You cannot specify a relative URN.
- (Optional) Enter a **File Location**. You can use the browse (...) button to select a target directory on your file system. If you don't specify a File Location, the new file will be saved to the root of your hard drive.
- (Optional) In the **Documentation** text box, enter any human-readable comment or descriptive language you would like to associate with the definition element.
- Check **Include WSDL template** if you want a skeleton document to be created for you using values provided in this wizard. Uncheck it to start with a blank document.

5 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
- Adding a port type element
- Adding a binding element
- Adding a service element

Adding a message element

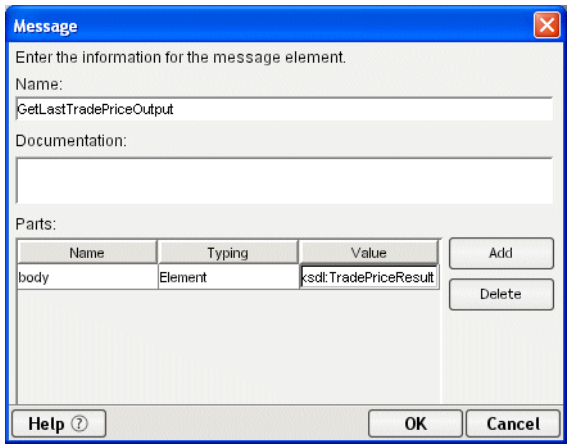
In WSDL, a **message** is an abstract definition of the data being exchanged.

➢ **To add a message element to a WSDL document:**

1 In the **XML Source View**, position the insertion point where you want to insert the definition, then 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: ◆ The name attribute ◆ The typing value (Element or Type) ◆ Under Value, the element attribute To add another part entry to the message, click **Add**. To remove an entry, select the entry and click **Delete**.

4 Click **OK**.

A new message section is added to your document.

Adding a port type element

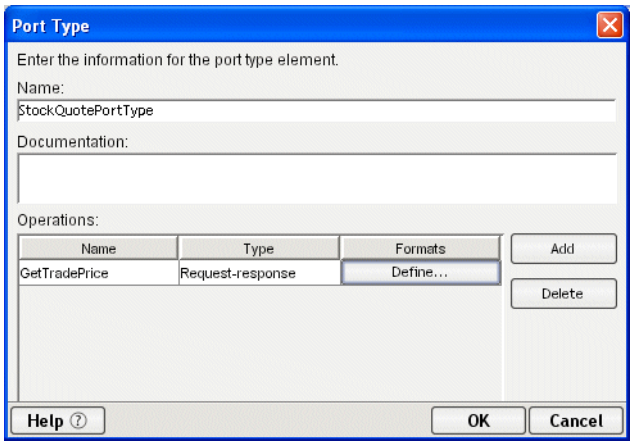
A WSDL **port type** is an abstract definition of the operations supported by a service and the communications mode (one-way, request-response, and so on) that will be used in the service.

➢ **To add a port type to a WSDL document:**

1 In the **XML Source View**, position the insertion point where you want to insert the definition, then 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.
Operations	Specify this information for each **<operation>** element of your port type: ◆ 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 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. To add another operation entry to the port type, click **Add**. To remove an entry, select the entry and click **Delete**.

4 Click **OK**.

A new port type section is added to your document.

Adding a binding element

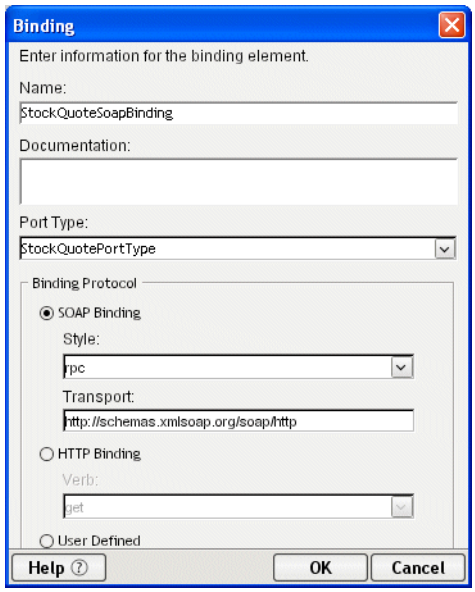
A WSDL **binding** specifies concrete protocol and data format specifications for the operations and messages defined by a particular port type.

➢ **To add a binding to a WSDL document:**

1 In the **XML Source View**, position the insertion point where you want to insert the definition, then right-click.

A popup menu displays.

2 Select **Insert WSDL Element>Binding**.



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 196).
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).
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.

Adding a service element

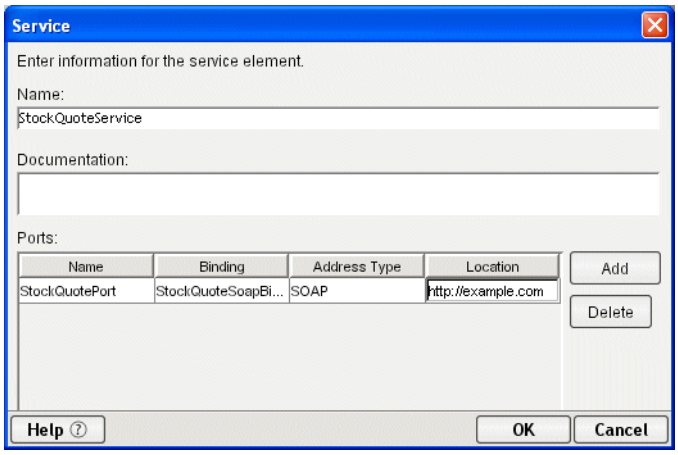
A WSDL **service** names the entry-point address (or addresses) for the Web Service in question. These addresses are in the form of URIs and constitute **ports**.

➢ **To add a service to a WSDL document:**

1 In the **XML Source View**, position the insertion point where you want to insert the definition, then right-click.

A popup menu displays.

2 Select **Insert WSDL Element>Service**.



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	Specify this information for each **<port>** element of your service: ◆ 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 197) ◆ The address type (None, SOAP, or HTTP) ◆ The location (the URI by which your service will be available) To add another port entry to the service, click **Add**. To remove an entry, select the entry and click **Delete**.

4 Click **OK**.

A new service entry is added to your document.

Validating a WSDL document

When a WSDL document is displayed in the XML Source View, you can validate the document by clicking the **Validate** toolbar button (which looks like a check mark). 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

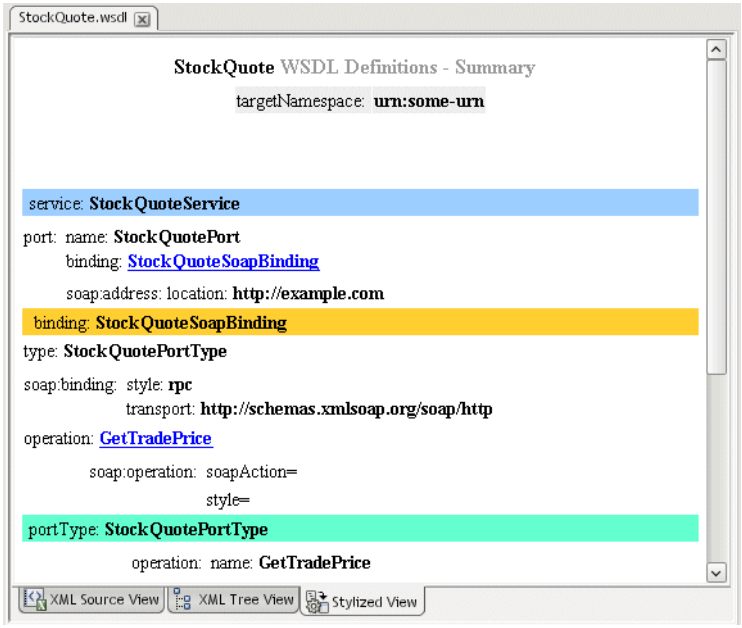
For easier reading, you can display your WSDL document in the **Stylized View**, which applies an XSL style sheet to the document. The WSDL Editor comes with two built-in style sheets: **Summary** and **Details**.

➢ **To display a stylized view of a WSDL document:**

1 Open the WSDL document.

2 Click the **Stylized View** 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.

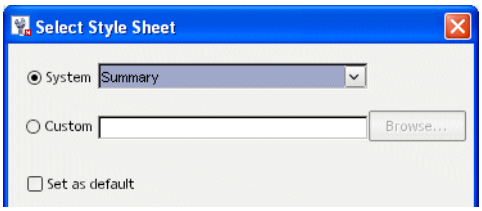
➢ **To choose a different style for the stylized view:**

1 With the **Stylized View** 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 development environment 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 213.

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 **project** is open.

2 Open the WSDL document in the **XML Source View**.

3 Click the **Generate Java Class** button.

The Web Service Wizard is invoked.

📖 For more information, see Chapter 10, "Web Service Wizard".

12 Registry Manager

This chapter describes the registry browsing and publishing functionality provided by the Novell exteNd Director development environment. It contains the following topics:

About registry standards

The exteNd Director development environment supports the following registry standards:

Registry standard	Description
UDDI	Universal Description, Discovery, and Integration UDDI is designed to give businesses a uniform way to describe their Web Services, discover other companies' services, and understand the methods needed to conduct e-business in an automated or semiautomated way with remote partners.
ebXML	Electronic Business using eXtensible Markup Language The ebXML Registry and Repository, like UDDI, enables the storing and sharing of information between parties to allow e-business collaboration.
WSIL	Web Services Inspection Language (also known as WS-Inspection) WSIL is designed to be more lightweight and portable than UDDI. It's an emerging standard intended to pick up where UDDI leaves off.

About the Registry Manager

The exteNd Director development environment provides a Registry Manager (accessible through the Registries tab in the Navigation Pane) and a facility for defining registry profiles. The registry capabilities include:

- **Defining** registry profiles
- **Selecting** registries to include in the search process
- **Viewing** business information on selected organizations in a given registry

  Viewing information on Web Services offered by a given organization
- **Searching** for organizations or services within a registry or group of registries, optionally using extended query parameters

- **Publishing** services or organizations to a registry
- **Deleting** services or organizations from a registry

Defining registry profiles

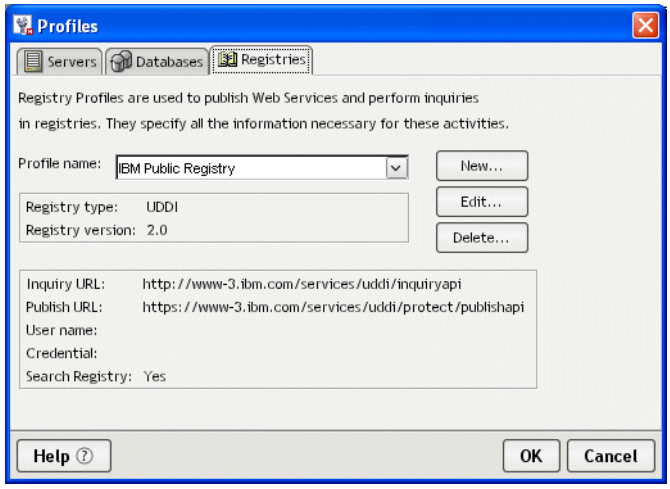
Registries are specified by URL and can be local or Web-based. Before accessing a registry in the exteNd Director development environment, you define a **profile** for that registry. Some predefined registry profiles are provided.

➢ **To define a registry profile:**

1 Select **Tools>Profiles** from the menu.

The Profiles dialog opens.

2 Select the **Registries** tab.

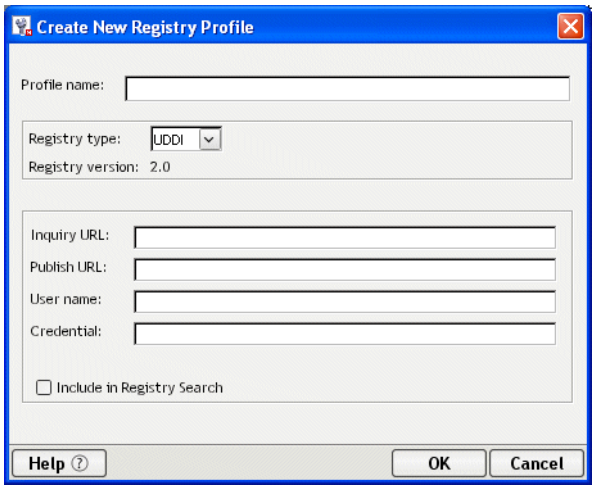


3 If you are editing or deleting an existing profile, select it from the **Profile name** list box and click **Edit** or **Delete**. If you are creating a new profile, click **New**.

4 When the New (or Edit) dialog displays, first specify the following:

Option	Description
Profile name	Name of the profile
Registry type	Type of registry: ebXML, UDDI, or WSIL The dialog displays different options depending on which registry type you select.

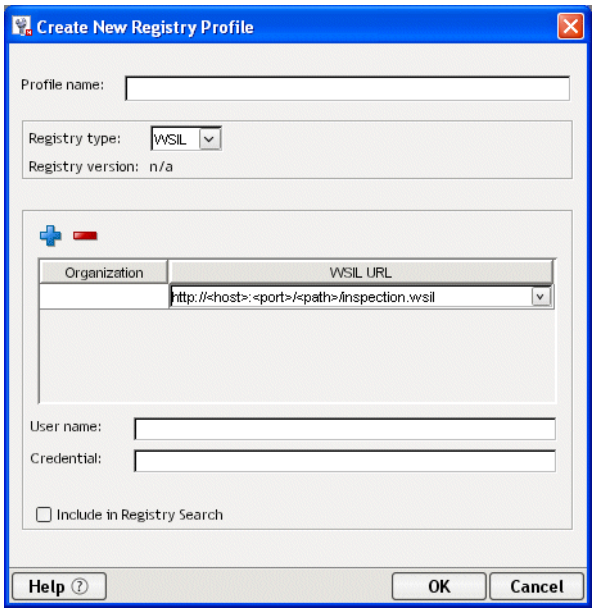
5 If you select **ebXML or UDDI** as the registry type, the dialog displays:



Complete the dialog as follows:

ebXML/UDDI option	Description
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

If you select **WSIL** as the registry type, the dialog displays:



Complete the dialog as follows (use the **+ and - buttons** to add or delete entries in the WSIL registry list):

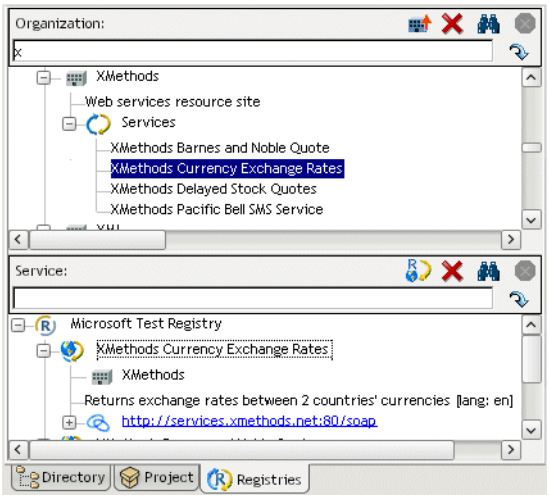
WSIL option	Description
Organization	A name for the organization
WSIL URL	The fully qualified WSIL URL, ending in `inspection.wsil`
User name and Credential	The information (if any) that the registry provider assigned to you for publishing access
Include in Registry Search	Specifies whether you want to include this registry in the default search set

6 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 Organization Pane and the Service Pane.



Information displayed

The Registry Manager displays the following types of information.

Organization Pane The organization section of a registry might include these types of information:

Information	Icon	Description
Organization name		Organization name used in this registry
Description		Short phrase describing the organization

Information	Icon	Description
Categories		Categories to which the organization belongs Classification schemes come from at least three sources: NAICS codes for industry segments, UNSPSC for product and service classifications, and geographic information
Identifiers		Information about the organization, such as a DUNS number
Services		A list of services offered by the organization, 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
Organization name		The organization offering the service
Description		A short phrase describing the service
Binding		The URL for invoking the service
tModel	tM	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.

Organization Pane To view the popup menu for Organization, place the cursor in an entry in the Organization Pane and right-click. The following menu displays.

Menu item	Description
Copy Text	Allows you to copy text from the currently selected organization tree node to another area or file
Clear Tree	Clears the pane of organization information that you retrieved from your search
Delete Organization	Deletes the selected organization (if you have permission). Asks you to confirm before deleting the organization.

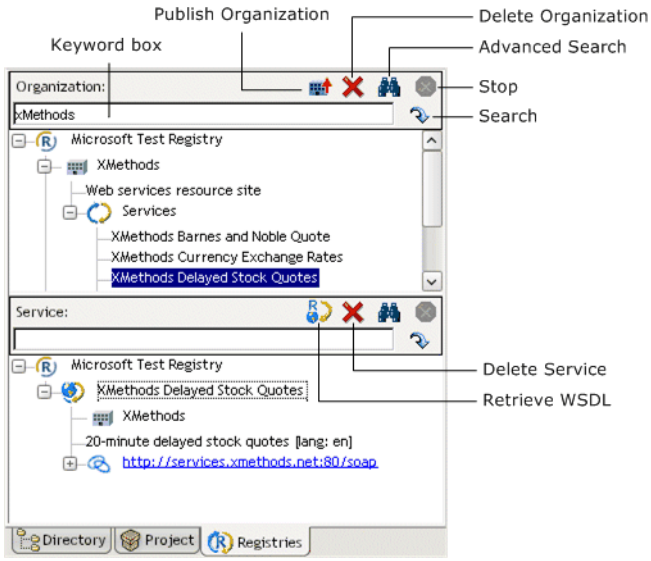
Menu item	Description
Advanced Search	Allows you to perform a sophisticated search by organization For more information, see “Searching by organization” on page 209

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 211

Action buttons

The following illustration shows the location of the various action buttons on the Organization and Service panes.



Searching by organization

You search by organization in the Organization Pane.

➢ **To search organizations by name or keyword:**

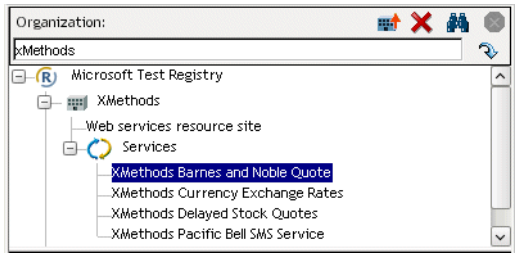
1 Enter a complete or partial organization name or keyword in the text box below **Organization**.

TIP: You can also enter a group of organization names separated by a vertical bar, which allows you to search for multiple groups of organizations. For example: XMethods|IBM|Sun.

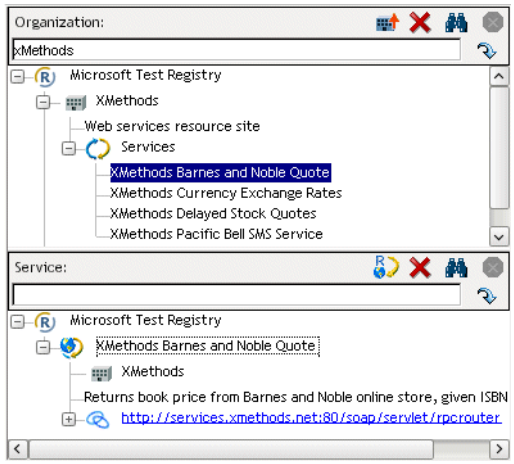
2 Click the **Search** button (shaped like a downturned arrow).

While the search is under way, 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 Organization Pane.

A list of matching organizations appears in tree-view form. Each top-level node in the tree is a registry, each child of a registry is an organization name, and below each organization is detail information consisting of descriptions, categories, and services.



Clicking a service entry in the (upper) Organization tree causes that service's detail information to appear in tree form in the (lower) Service Pane.

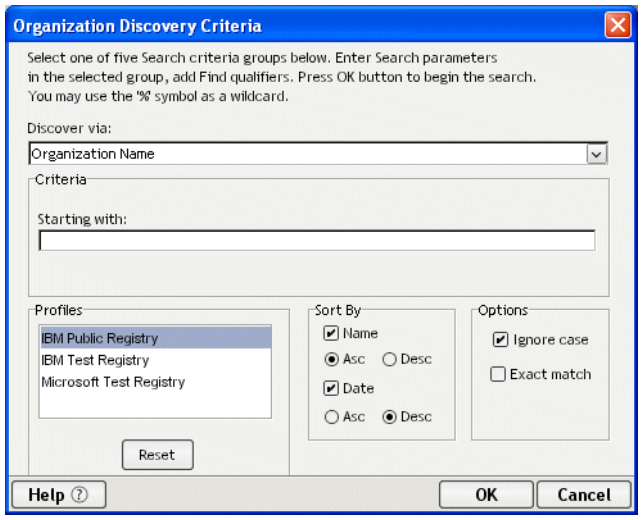


➢ **To set advanced organization search criteria:**

1 Leave the keyword text box blank.

2 Click the **Advanced Search** button (shaped like binoculars).

The Organization Discovery Criteria dialog displays.



3 Select one of the search-criteria options:

- **Organization Name**—Enter a complete or partial organization 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 ellipsis and pick an item from a list of choices.
- **Service Type Name**—Select to search organizations 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 organizations with identical names.
- In **Options**, select Ignore Case and/or Exact Match.

5 Under **Profiles**, select the registry or registries to search from the 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 under way, 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 organization name associated with the service, a description of the service, and bindings for the service.

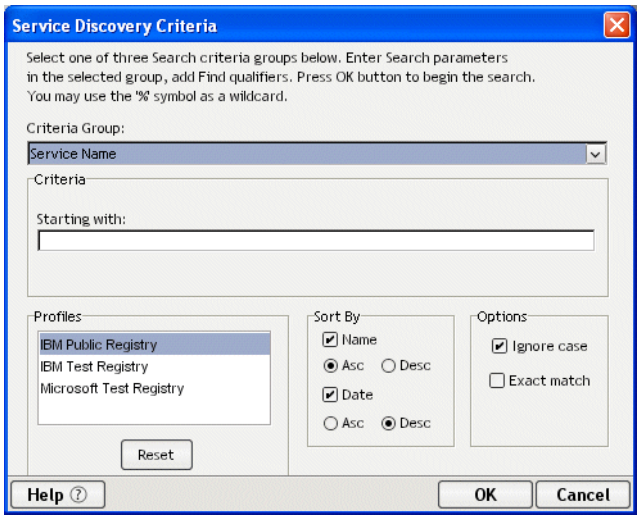
Clicking a service node in the (lower) Service tree causes that organization's detail information to appear in tree form in the (upper) Organization Pane.

➢ **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 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 ellipsis 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 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.

Using wildcards in searches

While searching in the Registry Manager, you can use the **percent sign (%)** as a wildcard symbol, meaning one or more of any character. This is especially useful when you want to search for organization or service names that contain a particular word but might not start with that word.

The default search logic is Start With. So a search on `Books` will turn up BooksRUs but not ABC Booksellers or Used Books. The way to override this behavior is to search instead on `%Books%`, which will turn up all three.

You can also use the **| (pipe) symbol** as a logical OR to look for hits that contain any combination of specified words. You can chain together any number of keywords this way. For example:

```
%Booking% | %Travel% | %Airline%
```

returns all names that contain at least one of these words, no matter where in the name that word appears.

Retrieving WSDL from the registry

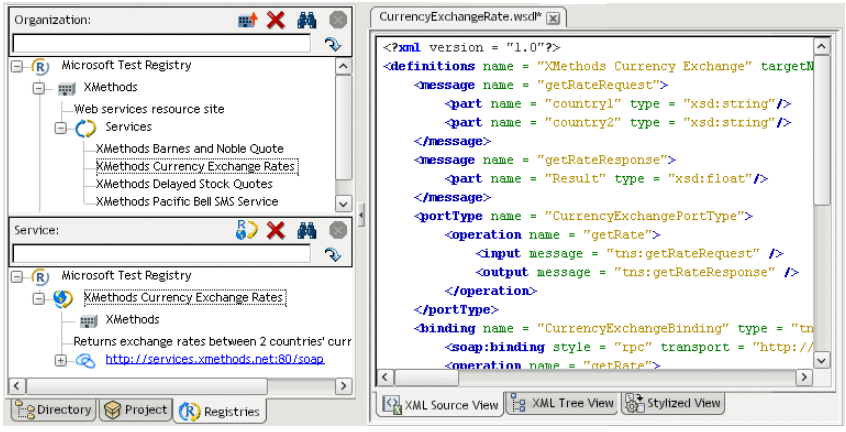
After you have found the service you searched for, you can retrieve the WSDL definition for this service from the registry. For this you use the Service Pane.

➢ **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 11, “WSDL Editor”.

Publishing to a registry

When you have created a WSDL document, you can publish it to a registry via the WSDL Editor.

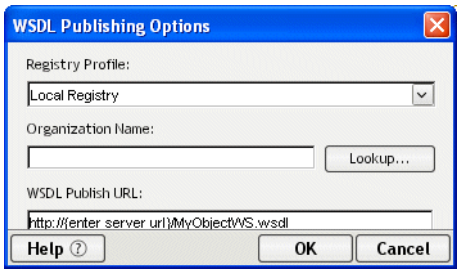
📖 For information about the WSDL Editor, see Chapter 11, “WSDL Editor”.

➢ **To publish WSDL 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.
- **Organization Name**—Specify the organization to associate the service with. Click the **Lookup** button to look up organizations in the registry.
- **WSDL Publish URL**—Specify the URL to which the service will be published.

4 Click **OK**.

If your service is successfully published, you see a confirmation dialog. Otherwise, you see a dialog describing the error.

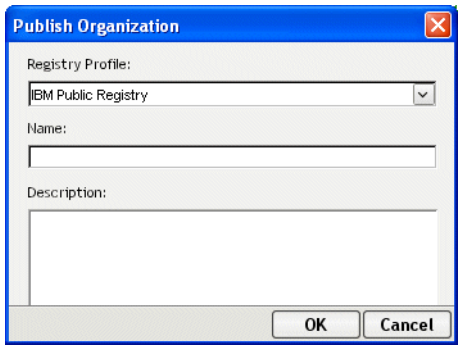
If you want to publish an organization to a registry, you can do that from the Organization Pane of the Registry Manager.

➢ **To publish an organization to a registry:**

1 Go to the **Registry Manager**.

2 Click the **Publish Organization** button in the Organization Pane.

The Publish Organization dialog displays.



3 Specify these options:

- **Registry Profile**—Select the registry you want to publish to.
- **Name**—Specify a name for the organization.
- **Description**—Specify text that describes the organization.

4 Click **OK**.

If your organization is successfully published, you see a confirmation dialog. Otherwise, you see a dialog describing the error.

III J2EE

These chapters present the exteNd Director utility tools for working with J2EE components:

- Chapter 13, “J2EE Wizards”
- Chapter 14, “How to Handle J2EE Versions”

Related links:

- Sun J2EE home
- Sun J2EE documentation

13 J2EE Wizards

To speed project development, use the following wizards when creating standard J2EE components and other Java classes:

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

You access these wizards by selecting **File>New>File** 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 from the Project tab of the Navigation Pane.

Starting the EJB Wizard

➢ **To start the EJB Wizard:**

1. Select **File>New>File**.
2. On the General tab, choose **EJB** (in the Advanced section) and click **OK**. (Alternatively, you can double-click **EJB**.)
3. The steps you follow depend on the type of bean you want to generate; see "Panel sequence" (below) for details.

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. Click the link to display more information about how to complete the panel.

If you want to create	You step through these panels
A stateful or stateless session bean	**1** Specifying the EJB type **2** Specifying the EJB JAR configuration **3** Specifying the project, package, and directory **4** Specifying the EJB source ◆ Specifying the source class or interface **5** Specifying the EJB class and interface names **6** Specifying methods **7** Specifying additional classes or packages to import **8** Completing the EJB
A message-driven bean	**1** Specifying the EJB type **2** Specifying the EJB JAR configuration **3** Specifying the project, package, and directory **4** Specifying the EJB source ◆ Specifying the source class or interface **5** Specifying the EJB class and interface names **6** Specifying methods **7** Specifying additional classes or packages to import **8** Completing the EJB
A BMP entity bean	**1** Specifying the EJB type **2** Specifying the EJB JAR configuration **3** Specifying the project, package, and directory **4** Specifying the EJB source ◆ Specifying the source class or interface or ◆ Specifying the source database and Selecting a database table **5** Specifying the EJB class and interface names **6** Specifying persistent (data) fields **7** Specifying primary key fields **8** Specifying fields that require get/set methods **9** Specifying create() methods **10** Specifying find() methods **11** Specifying additional classes or packages to import **12** Specifying resource references **13** Completing the EJB

If you want to create	You step through these panels
A 1.x CMP entity bean	**1** Specifying the EJB type **2** Specifying the EJB JAR configuration **3** Specifying the project, package, and directory **4** Specifying the EJB source ◆ Specifying the source class or interface or ◆ Specifying the source database and Selecting a database table **5** Specifying the EJB class and interface names **6** Specifying persistent (data) fields **7** Specifying primary key fields **8** Specifying fields that require get/set methods **9** Specifying create() methods **10** Specifying find() methods **11** Specifying additional classes or packages to import **12** Completing the EJB
A 2.x CMP entity bean	**1** Specifying the EJB type **2** Specifying the EJB JAR configuration **3** Specifying the project, package, and directory **4** Specifying the EJB source ◆ Specifying the source class or interface or ◆ Specifying the source database and Selecting a database table **5** Specifying the EJB class and interface names **6** Specifying persistent (data) fields **7** Specifying primary key fields **8** Specifying fields that require get/set methods **9** Specifying relationships **10** Specifying create() methods **11** Specifying find() methods **12** Specifying additional classes or packages to import **13** Completing the EJB

Panel reference

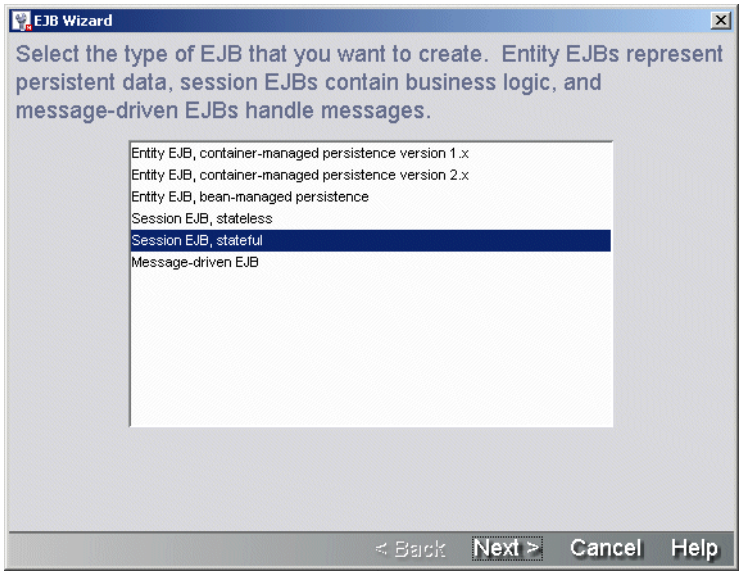
This section describes the options on each panel of the EJB Wizard. The panels are:

- 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:

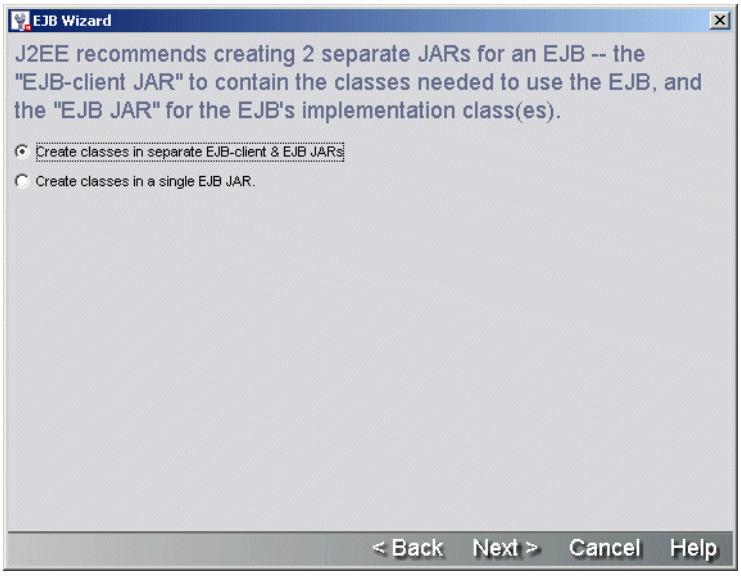
Option	What to do
Entity EJB, container-managed persistence Version 1.x	Select this option when you want the EJB Wizard to create an entity bean that uses container-managed persistence (CMP) defined by the EJB1.1 specification
Entity EJB, container-managed persistence Version 2.x	Select this option when 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 218.

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: ◆ **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 218.

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:

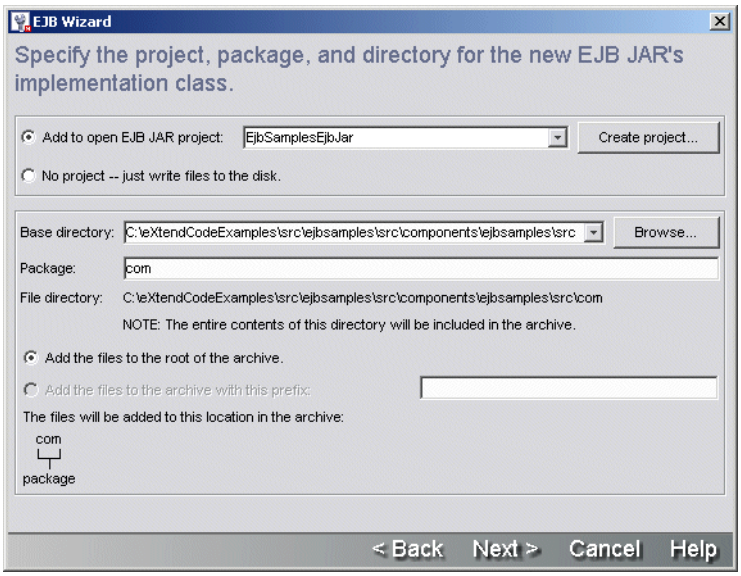
- EJB JAR project classpath includes EJB-client JAR project
- EJB JAR has a manifest file that includes a Class-Path entry for the EJB-client JAR archive
- The EJB JAR's deployment descriptor contains an <ejb-client-jar> element containing the name of the EJB-client JAR archive

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	If you currently have one or more EJB projects open, 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. If the EJB project is defined as an EJB1.1 project, you cannot add EJBs that use EJB2.0 features—and the wizard prevents you from doing so.
Create project	Click **Create project** to start the New Project Wizard. 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. 📖 For details, see "Creating projects and subprojects" on page 50.
No project -- just write files to the disk	If you do not want to associate the EJB with a project, you can still use the wizard to create the class in a nonproject directory on the file system.

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

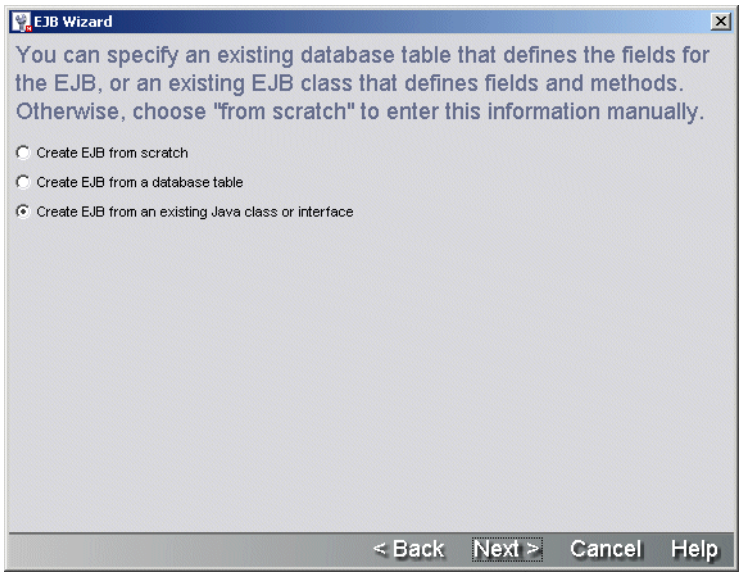
Option	What to do
Base directory	If you specified an EJB project, the default base directory is the project directory. Otherwise, this field is empty. (Click **Browse** to specify a file system location.)
Package	Specify the EJB's package name. This is **required**.
File directory	The contents of **Base** 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 218.

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:

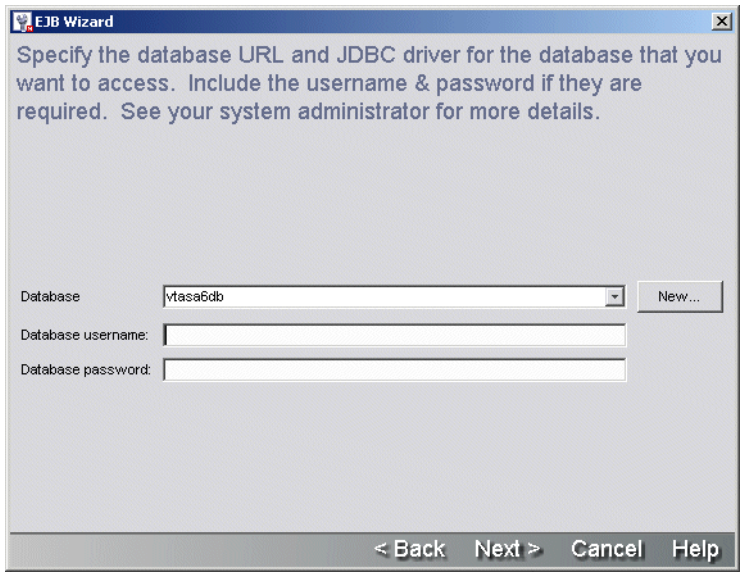
Option	What to do
Create EJB from scratch	Select this option if you want to create a new EJB
Create EJB from database	Select this option to create an entity bean whose fields are based on the fields in a specific database table
Create EJB from an existing Java class or interface	Select this option to 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 218.

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.



➢ **To complete this panel:**

1 Specify:

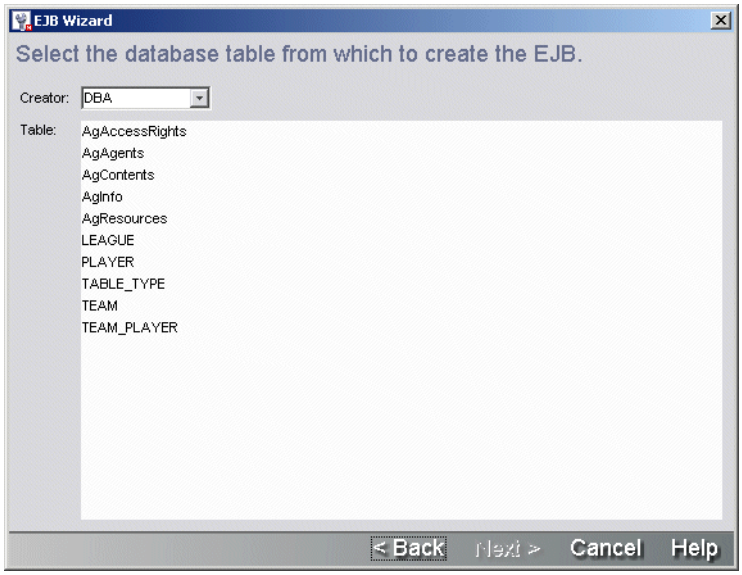
Option	What to do
Database	Select a database profile from the dropdown list box. If the dropdown is not populated or if the existing profiles are unsuitable, you must create a database profile by clicking **New** (or by leaving the wizard to select **Tools>Profiles** and click the **Databases** tab). When you complete this panel (by clicking **Next**), the wizard creates a client connection to the database. This means that the database driver (specified in the database profile) must be available to the exteNd Director development environment. 📖 For more information on database profiles and driver setup requirements, see "Database profile" on page 34.
Database username and Database password	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 the wizard can access the database's metadata.

2 Click **Next** to continue.

Return to "Panel sequence" on page 218.

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:

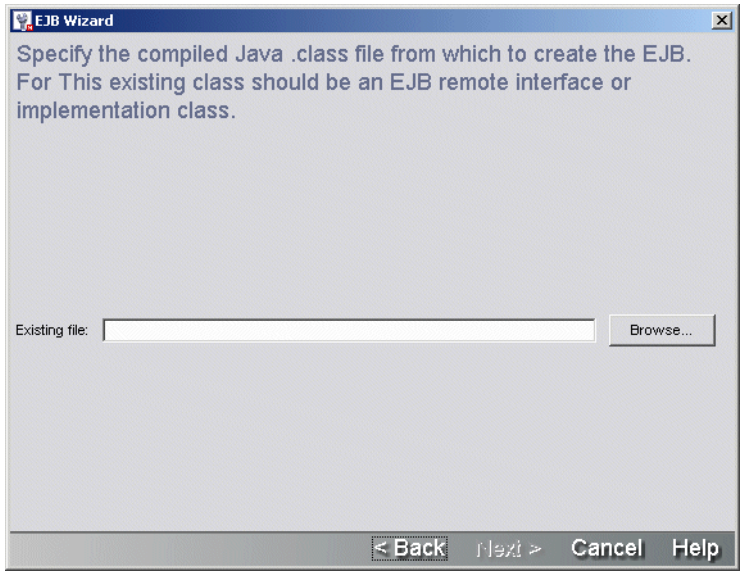
Option	What to do
Catalog/Creator/Schema	Select the Catalog/Creator/Schema containing the database table you want to use for the entity bean
Table	Select the database table that contains the fields you want to include in the entity bean

2 Click **Next** to continue.

Return to "Panel sequence" on page 218.

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 218.

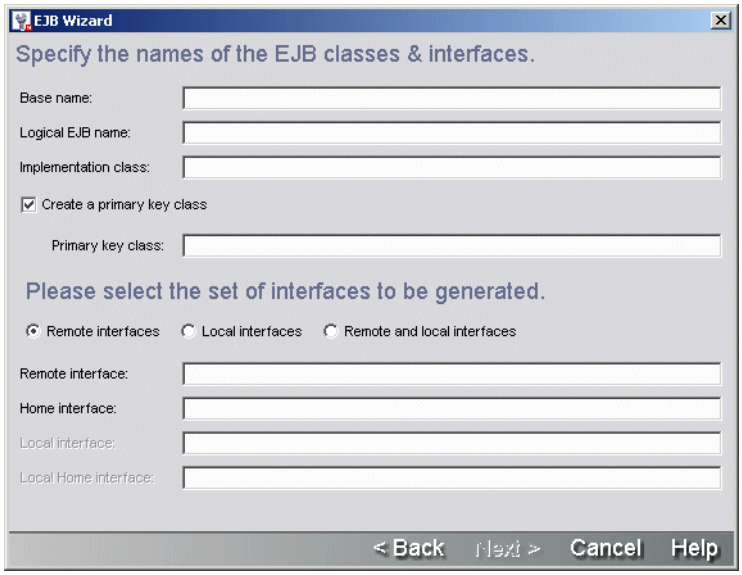
Specifying the EJB class and interface names

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	Example
Bean class	Prepends EB, SB, or MB and appends Bean to the base name	**SB**Calculator**Bean**
Remote interface	Prepends EB or SB to the base name	**SB**Calculator
Home interface	Prepends EB or SB and appends Home to the base name	**SB**Calculator**Home**
Local interface (EJB2.x only)	Prepends EB or SB and appends Local to the base name	**SB**Calculator**Local**

EJB component	Naming conventions	Example
Local home interface (EJB2.x only)	Prepends EB or SB and appends LocalHome to the base name	**SB**Calculator**LocalHome**
Primary key classes (entity beans only)	Prepends EB and appends PK to the base name	**EB**Customer**PK**



➢ **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	Specify a legal name for the EJB class and interfaces. This name is used to construct the names for the other EJB components 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 227 just above
Logical EJB name	Accept the default or provide a legal name This name is used: ◆ 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)
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.

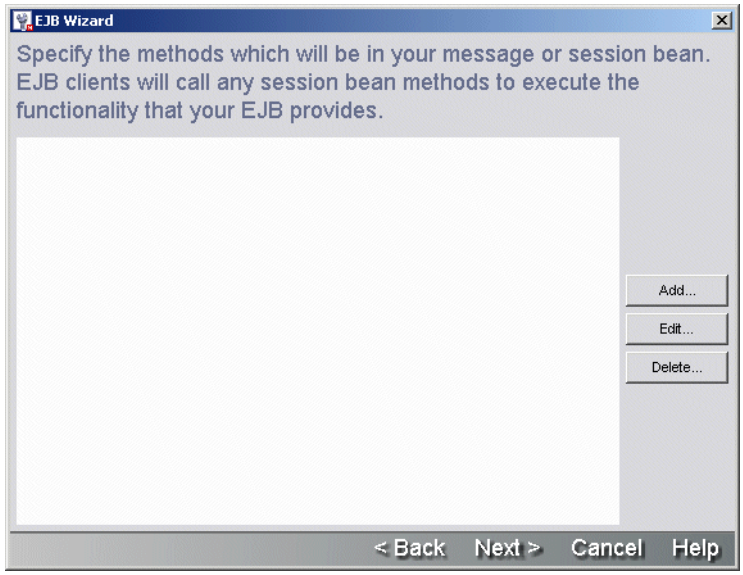
3 Accept the default names for the set of interfaces, or specify legal Java names.

4 Click **Next** to continue.

Return to "Panel sequence" on page 218.

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 will be 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: ◆ **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.

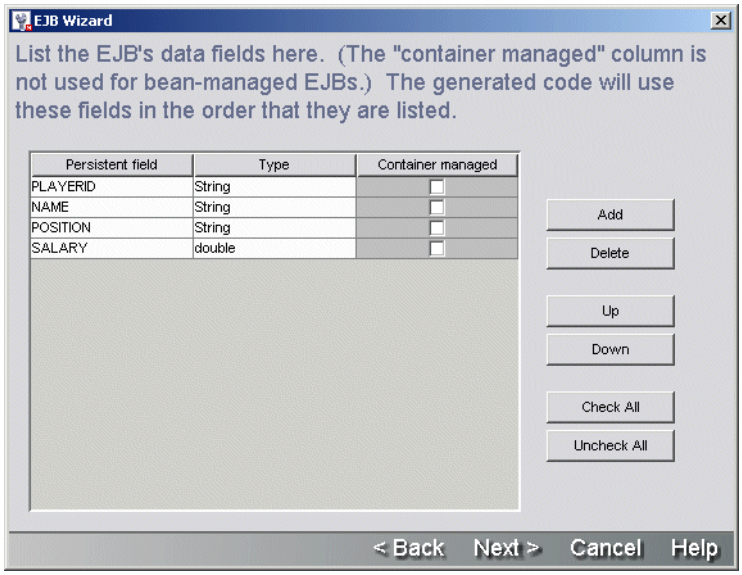
3 Repeat these steps to create other methods or click **Next** to continue.

Return to "Panel sequence" on page 218.

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.

NOTE: The EJB2.0 specification requires CMP field names to begin with a lowercase letter. If you base your entity bean on a table with field names that begin with uppercase (like Customer) or are all uppercase (CUSTOMER), the wizard generates lowercase variable names to comply with the specification. The wizard does not modify the names for BMP beans, because the specification does not require it.



➢ **To complete this panel:**

1 Specify:

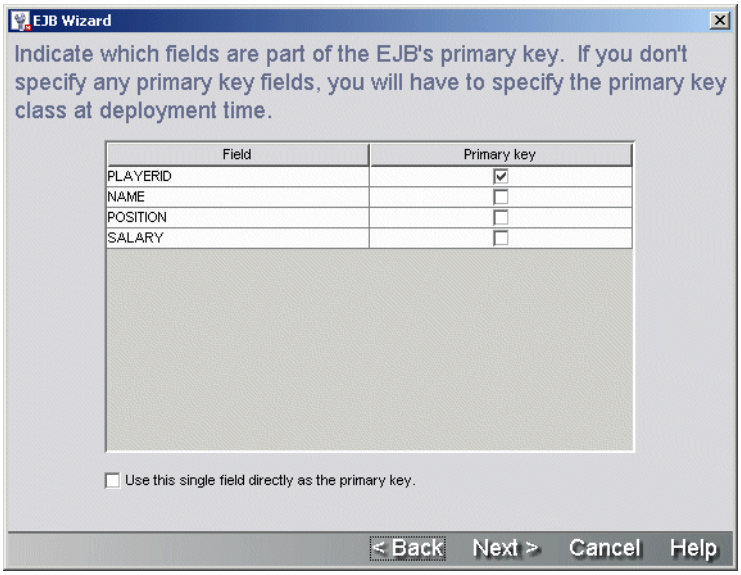
Option	What to do
Persistent field	If the fields that should be managed by the container are listed, make sure the check box in the container-managed column is checked Use the **Check All/ Uncheck All** and **Add/Delete** buttons to manage the list of container-managed data fields Use the **Up/Down** buttons to move the fields to the appropriate position if the entity bean should have a composite key Container-managed fields are listed in the deployment descriptor and can be mapped to a database field at deployment time
Type	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 java.sql.Types.
Container managed	Check or uncheck the fields as needed

2 Click **Next** to continue.

Return to "Panel sequence" on page 218.

Specifying primary key fields

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



➢ **To complete this panel:**

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

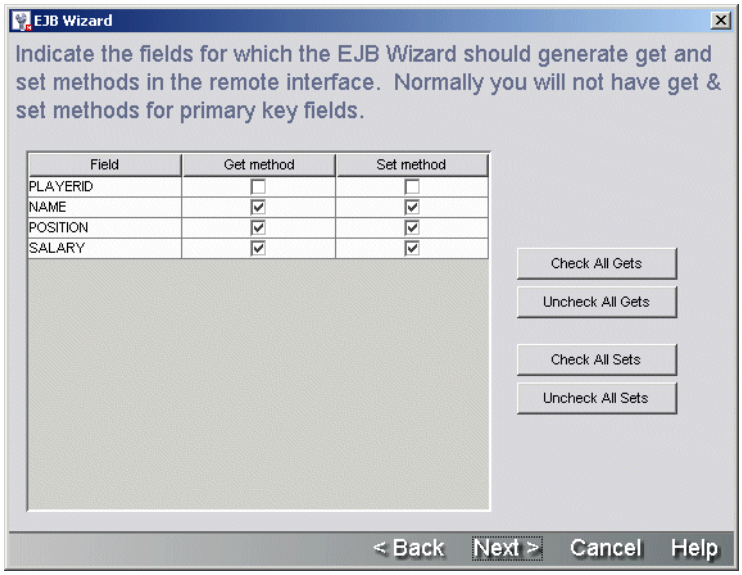
Option	What to do
Field	Move the cursor to the field
Primary Key	Depends on how you responded to the Create primary key checkbox on the wizard panel described in "Specifying the EJB class and interface names" on page 227. If you: ◆ Checked the **Create primary key** class checkbox, you can check one or more fields to be included in the primary key class that the wizard will generate. ◆ 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: ◆ 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 java.lang.Object. You will have to specify the primary key class type at deployment.
Use this single field directly as the primary key	Check this if you've selected a single field that the wizard should use as the primary key. The field must be a String or a wrapper class for a primitive type (such as java.lang.Integer). The wizard generates the code so that the wrapper-class type is in the deployment descriptor's <prim-key-class> element and the primary key field name is in the <primkey-field> element.

2 Click **Next** to continue.

Return to “Panel sequence” on page 218.

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:

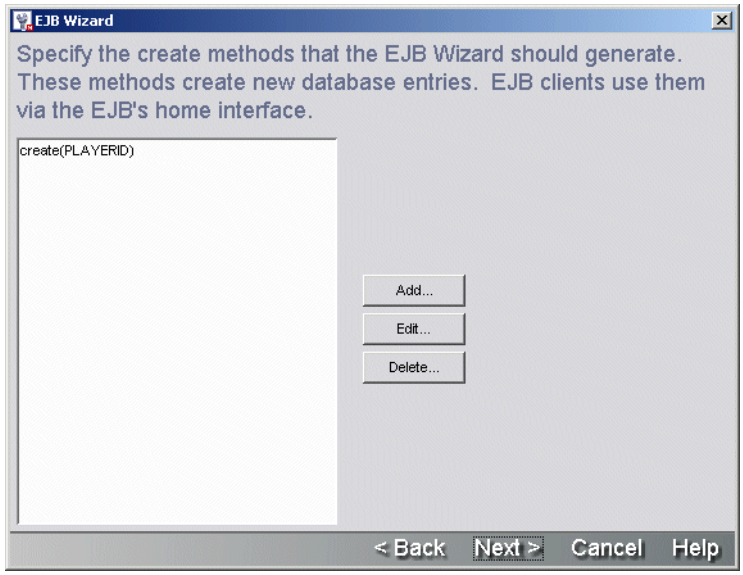
Option	What to do
Field	Move the cursor to the field
Get method	Specify whether the wizard should generate (checked) a get method for this data field If your entity bean will be doing any kind of read-only or read-write data access on this data field, you should have the wizard generate a get method
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 218.

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 218.

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.

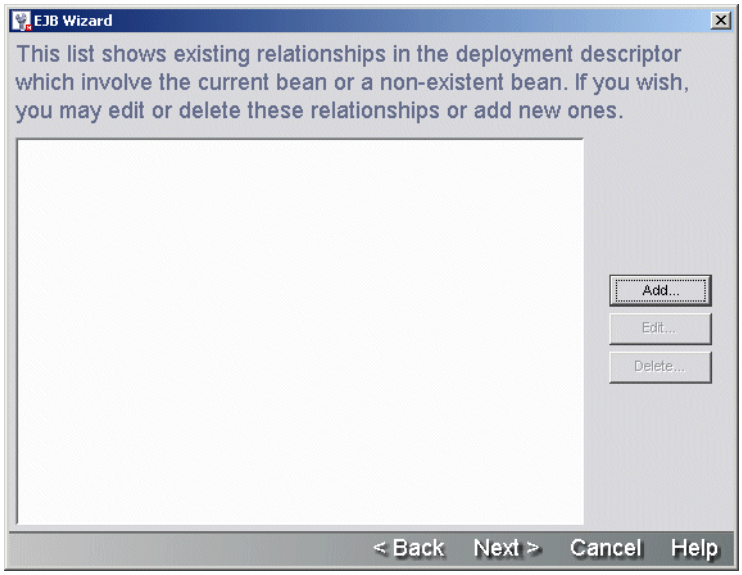
Relationships can be defined as bidirectional or unidirectional.

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 boths 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 a 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:**

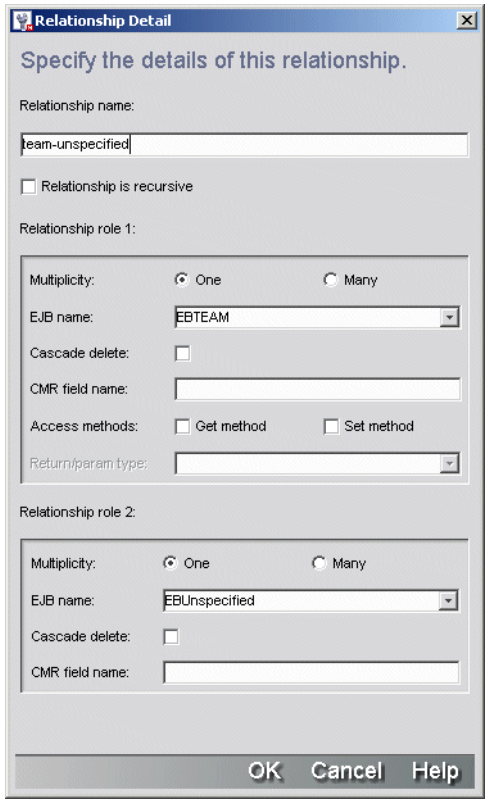
1 To add a relationship, choose **Add**.

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



➢ **To complete this panel:**

1 Specify:

Option	What to do
Relationship name	Enter a unique name that identifies the relationship you are constructing. This corresponds to the <ejb-relation-name> element in the Deployment Descriptor. This element is not required by the Deployment Descriptor or the wizard. For each relationship, you must specify two beans.
Relationship role 1	
Multiplicity	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. This corresponds to the <multiplicity> element in the deployment descriptor.
EJB name	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. The wizard adds this entry to the <ejb-name> element of the <relationship-role-source> element in the deployment descriptor.
Cascade delete	Check this if you want the current bean to be deleted when the related bean is deleted. This is available only when the related bean's multiplicity is One. This corresponds to the <cascade-delete> element of the deployment descriptor.
CMR field name	If this bean is in a bidirectional relationship or is in a unidirectional relationship and knows about the related bean, enter a name that begins with a lowercase letter. 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.
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	Enter the bean name. This should match an <ejb-name> element in the enterprise-beans section of the deployment descriptor (although it may not exist yet).
Cascade delete	Check this check box if you want this bean removed when the current bean is removed.

Option	What to do
CMR field name	If this bean is in a bidirectional relationship or is in a unidirectional relationship and knows about the current bean, enter a name that begins with a lowercase letter. 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.

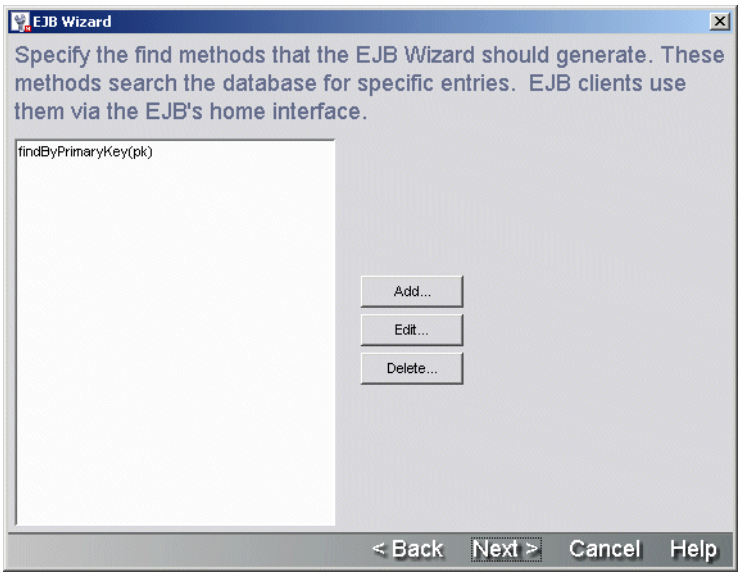
2 Click **OK** to return to the relationship panel.

3 Click **Next** to continue.

Return to "Panel sequence" on page 218.

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	Click **Add** to enable the Type and Name text boxes and then: ◆ Specify the Java data type of the parameter. ◆ Specify a legal Java parameter name.

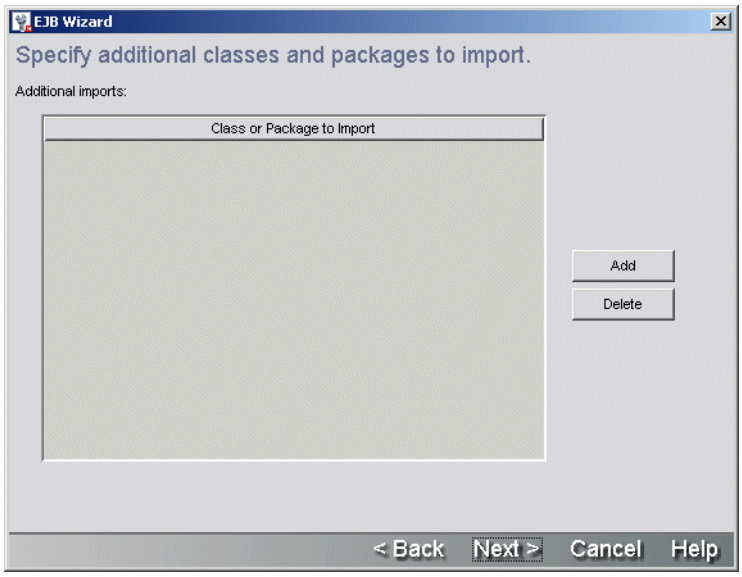
3 Click **OK** to return to the find() methods panel.

4 Click **Next** to continue.

Return to "Panel sequence" on page 218.

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.

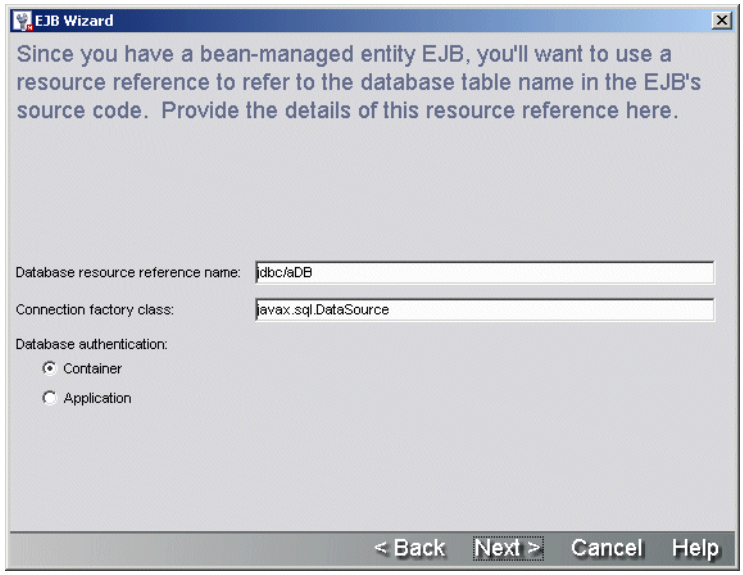
2 Type the fully qualified path name of the Java class or the full package name in the text field.

3 When you are done adding or removing additional classes or packages, click **Next** to continue.

Return to "Panel sequence" on page 218.

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.



➢ **To complete this panel:**

1 Specify:

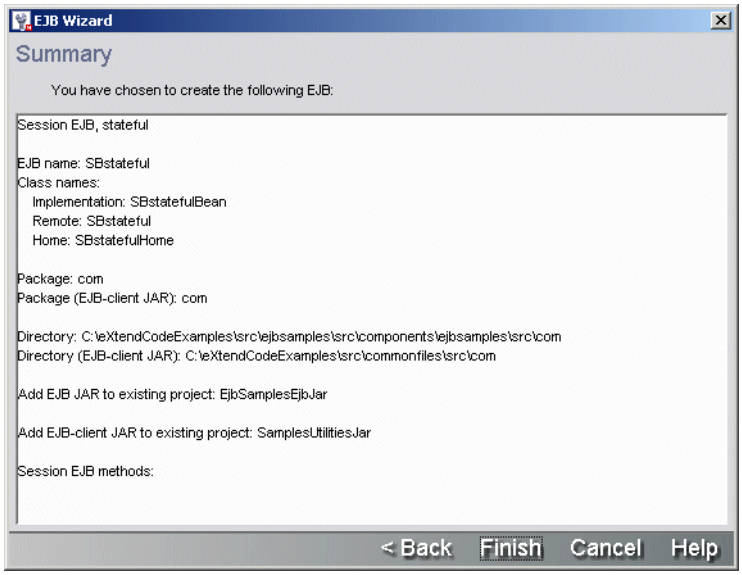
Option	What to do
Database resource reference name	Specify the name that will be put in the JNDI environment and looked up by anyone trying to get the named resource The EJB specification recommends that this be prefaced with **jdbc/**—for example: `jdbc/MyDataSource` The deployer will later map this 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: ◆ Specifying **container** means the container signs onto the resource manager in order to obtain the resource factory ◆ Specifying **application** means the code in the EJB signs onto the resource manager programmatically

2 Click **Next** to continue.

Return to "Panel sequence" on page 218.

Completing the EJB

This panel shows all of the classes and interfaces that the wizard will generate. Review it carefully to make sure that you have specified everything 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 218.

JSP Wizard

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

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. Select **File>New>File**.
2. On the General tab, choose **JSP** and click **OK**. (Alternatively, you can double-click **JSP**.)
3. Continue as described in "Specifying the JSP page name and other options" (below).

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.
Page title	Specify the text for the JSP page's title. Generated as <title>*text*</title>.
Content type	Specify the MIME-type of the response generated by the JSP page. Choose from the list provided. The default is HTML. Generated as the **contentType** attribute of the page directive.
Template	Specify a code-generation template if you want to use one other than the default. Depending on your exteNd product configuration, you may have a choice of templates (which tailor class generation for different needs).
Use session	Specify whether the JSP page participates in session management (in other words, is part of a session). Generated as the **session** attribute of the page directive.
Thread safe	Specify whether the JSP page, once compiled into a servlet, can respond to multiple simultaneous requests. If not, deselect the check box. Generated as the **isThreadSafe** attribute of the page directive.
Form-based page	Specify whether a simple HTML form is generated on the JSP page (enabled only if you are generating an HTML or XHTML page).
Create error page	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. Generated as the **errorPage** attribute of the page directive.
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".

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, 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 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 50 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.
Base directory	If you specified a WAR project, the default base directory is the **jsps** subdirectory of 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	Specify a package hierarchy (with levels separated by periods) to place the JSP page in a subdirectory of the base directory. 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. For example, if the base directory is *ProjectDir*/jsps and you specified **com.myco** as the package, the JSP page will be created in *ProjectDir*/jsps/com/myco.
File directory	The contents of **Base directory** and **Package** are combined to specify the location of the JSP page, which is displayed under **File directory**. This is the file system location where the wizard creates the JSP page. 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 JSP page at the root of the archive (taking into account any package structure you specified).
Add the files to the archive with this prefix	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).
The files will be added to this location in the archive	The location in the archive of the JSP page, as specified by the two preceding selections, is reflected in this field. You cannot change the contents of this field directly; you must change the preceding two selections.

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".

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".

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".

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 *JSPPageName*ErrorPage.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. The following sections describe:

- 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 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.

Starting the Servlet Wizard

➢ **To start the Servlet Wizard:**

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

2 On the General tab, choose **Servlet** and click **OK**. (Alternatively, you can double-click **Servlet**.)

3 Continue as described in "Specifying the class name and other servlet options" (below).

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 you want to use one other than the default. Depending on your exteNd product configuration, you may have a choice of templates (which tailor class generation for different needs).
Implement SingleThreadModel	Specify whether the servlet class is to implement the SingleThreadModel interface. 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. The default is to allow multithreaded access to the servlet.

2 Click **Next** to go to the next wizard panel. See "Specifying the project, directory, and package".

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, you can add the servlet to one of those projects by selecting it from the list.
Create project	If you do not have a WAR project open but want to associate the servlet with a WAR project, you can click **Create project** to start the New Project Wizard. 📖 For details, see "Creating projects and subprojects" on page 50.
No project—just write files to the disk	If you do not want to associate the servlet with a 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.
Add the files to the archive with this prefix	When generating the project archive, place the generated class files for the servlet in a directory tree as specified in the prefix. The default is to place the servlet classes in a WEB-INF/classes directory under the root of the 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 classes.
The files will be added to this location in the archive	The location in the archive of the generated servlet class files, as specified by the two preceding selections, is reflected in this field. You cannot change the contents of this field directly; you must change the preceding two selections.

2 Click **Next** to go to the next wizard panel. See "Specifying which HttpServlet methods to override".

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".

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".

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

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. Select **File>New>File**.
2. On the General 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" (below).

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 you want to use one other than the default. Depending on your exteNd product configuration, you may have a choice of templates (which tailor class generation for different needs).
Bottom group (check boxes)	Use any of the following check boxes to further specify class attributes: ◆ 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".

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".

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".

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, you can add the class file to one of those projects by selecting it from the list.
Create project	If you do not have a project open but want to associate the class file 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 class file to. 📖 For more information, see "Project design considerations" on page 48.
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	If you specified a 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. The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.
Package	Specify the fully-qualified Java package name for the new class. You can specify a package hierarchy with levels separated by periods. The Java class you are creating is saved in the **Base directory** combined with the **Package** directory. For example, if the base directory is *ProjectDir*/classes and you specified **com.myco** as the package, the class will be created in *ProjectDir*/classes/com/myco.
File directory	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**. This is the file system location where the wizard creates the Java class 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	Adds the compiled Java 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	Adds the compiled Java class file 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 class.
The files will be added to this location in the archive	The location in the archive of the generated Java class file, 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 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 58.

JavaBean Wizard

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

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 Select **File>New>File**.

2 On the General tab, choose **JavaBean** (in the Advanced section) and click **OK**. (Alternatively, you can double-click **JavaBean**.)

3 Continue as described in "Specifying the class name and other options" (below).

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.
Base class	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. Generated as **extends *class***.
Template	Specify a code-generation template if you want to use one other than the default. Depending on your exteNd product configuration, you may have a choice of templates (which tailor class generation for different needs).

2 Click **Next** to go to the next wizard panel. See "Specifying the data fields".

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 go to the next wizard panel. See "Specifying which interfaces to implement".

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".

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".

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, 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 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 48.
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	If you specified a 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. The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.
Package	Specify the fully qualified Java package name for the new bean class. You can specify a package hierarchy with levels separated by periods. The bean you are creating is saved in the **Base directory** combined with the **Package** directory. For example, if the base directory is *ProjectDir*/classes and you specified **com.myco** as the package, the bean will be created in *ProjectDir*/classes/com/myco.
File directory	The contents of **Base directory** and **Package** are combined to specify the location of the bean, which is displayed under **File directory**. This is the file system location where the wizard creates the bean 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	Adds the compiled JavaBean to the archive root combined with the package path when generating the project archive.
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, is 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

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 on the Project tab of the Navigation Pane.

Starting the Tag Handler Wizard

An open project is required before you start this wizard.

➢ **To start the Tag Handler Wizard:**

1. Select **File>New>File**.
2. On the General tab, choose **Tag handler** (in the Advanced section) and click **OK**. (Alternatively, you can double-click **Tag handler**.)
3. Continue as described in "Specifying the class name and other options" (below).

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 if you want to use one other than the default. Depending on your exteNd product configuration, you may have a choice of templates (which tailor class generation for different needs).
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).

2 Click **Next** to go to the next wizard panel. See "Specifying the project, directory, and package".

Specifying the project, directory, and package

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

1 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, you can add the class file to one of those projects by selecting it from the list.
Create project	If you want to associate the class file with a new 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 class file to. 📖 For more information, see "Project design considerations" on page 48.
No project—just write the files to the disk	**Disabled—you must associate the tag handler class with a project**

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

Option	What to do
Base directory	The default base directory is the project's **src** subdirectory located directly under the project directory. Click **Browse** to specify a file system location. The Base directory is the project root combined with whatever other directories are in the project directory structure above the package path.
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	This is the file system location where the wizard creates the tag handler source file and the TagExtraInfo class source file, if any. The tag handler class you are creating is saved in the **Base directory** combined with the **Package** directory. For example, if the base directory is *ProjectDir*/classes and you specified **com.myco** as the package, the class will be created in *ProjectDir*/classes/com/myco. 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	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	Adds the compiled tag handler class file 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 class.

Option	What to do
The files will be added to this location in the archive	The location in the archive of the generated tag handler class file, as specified by the two preceding selections, is reflected in this field. You cannot change the contents of this field directly; you must change the preceding two selections.

3 Click **Next** to proceed to the next wizard panel. See "Specifying the tag library descriptor file".

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: `/mytags`.
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.
Archive location	Specify the directory location for the TLD within the archive: ◆ 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 257.
- If the custom tag uses tag element attributes, see "Specifying tag handler attributes" on page 257.

- If the custom tag uses or creates scripting variables, see "Specifying tag handler scripting variables" on page 258.
- Otherwise, see "Specifying TagExtraInfo class" on page 258.

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 257.
- If the custom tag uses or creates scripting variables, see "Specifying tag handler scripting variables" on page 258.
- Otherwise, see "Specifying TagExtraInfo class" on page 258.

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. The value must be a nonprimitive type.
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.

2 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 258.
- Otherwise, see "Specifying TagExtraInfo class" on page 258.

Specifying tag handler scripting variables

➢ **To specify tag handler scripting variables:**

1 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: ◆ 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

2 Click **Next** to go to the next wizard panel. See "Specifying TagExtraInfo class".

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	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. (Optional) Choose the appropriate checkbox if you want the wizard to implement either of the following methods: ◆ getVariableInfo() ◆ isValid()

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

See "What happens".

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 *TagHandlerClassName*ExtraInfo.java.
- Adds the tag handler class (and theTagExtraInfoClass 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.

14 How to Handle J2EE Versions

This chapter helps you target your application at an appropriate version of J2EE (Java 2 Platform, Enterprise Edition). It is especially useful if you need to know when and how to **migrate** existing projects to a newer J2EE version. Topics include:

- Support for J2EE versions
- Your choices
- Versions for new projects and components
- Migrating projects from J2EE 1.2 to 1.3
- exteNd Application Server considerations

📖 For an overview of J2EE, see the Sun J2EE Web site.

Support for J2EE versions

Choices you make about J2EE versions are largely determined by the J2EE server(s) you deploy to. To begin, find out whether your J2EE server is compatible with J2EE **1.2** or **1.3**. Then read the following topics to learn what that means for your application and how you handle it in the Novell exteNd Director development environment:

- What J2EE 1.2 servers support
- What J2EE 1.3 servers support
- What the development environment supports

What J2EE 1.2 servers support

If you develop applications for deployment to J2EE 1.2 servers, you can use J2EE 1.2 archives and J2EE 1.2 technologies. But J2EE 1.2 servers **do not** support J2EE 1.3 archives and J2EE 1.3 technologies.

J2EE 1.2 archives The J2EE 1.2 archives are:

Name and version	Description
EAR 1.2	Enterprise (application) archive
WAR 2.2	Web archive (for servlets and JSP pages)
EJB JAR 1.1	Enterprise JavaBean (EJB) archive
Client JAR 1.2	Application client archive (CAR)

J2EE 1.2 technologies The J2EE 1.2 technologies include:

- Java Servlets 2.2
- JavaServer Pages (JSP) 1.1
- Enterprise JavaBeans (EJB) 1.1

- JDBC Standard Extension 2.0
- Java Transaction API (JTA) 1.0
- JavaMail 1.1
- Java Messaging Service (JMS) 1.0
- Java Naming and Directory Interface (JNDI) 1.2
- RMI-IIOP

What J2EE 1.3 servers support

If you develop applications for deployment to J2EE 1.3 servers, you can use either of the following:

- J2EE 1.2 archives and J2EE 1.2 technologies
- J2EE 1.3 archives and J2EE 1.3 technologies

In other words, J2EE 1.3 servers support J2EE 1.2 applications as well as J2EE 1.3 applications. For further flexibility, you also have the option of mixing J2EE 1.2 modules and features into your J2EE 1.3 applications as follows:

These J2EE 1.3 archives	Can contain
EARs	Both J2EE 1.2 modules (WARs, EJB JARs, client JARs) and J2EE 1.3 modules
EJB JARs	Entity beans that use either 1.*x* or 2.*x* container-managed persistence (CMP)

J2EE 1.3 archives The J2EE 1.3 archives are:

Name and version	Description
EAR 1.3	Enterprise (application) archive
WAR 2.3	Web archive (for servlets and JSP pages)
RAR 1.0	Resource adapter archive (for Connector architecture)
EJB JAR 2.0	Enterprise JavaBean (EJB) archive
Client JAR 1.3	Application client archive (CAR)

J2EE 1.3 technologies The J2EE 1.3 technologies include:

- Java Servlets 2.3
- JavaServer Pages (JSP) 1.2
- Enterprise JavaBeans (EJB) 2.0
- J2EE Connector 1.0
- JDBC Standard Extension 2.0
- Java Transaction API (JTA) 1.0
- JavaMail 1.2
- Java Messaging Service (JMS) 1.0.2
- Java API for XML Parsing (JAXP) 1.1
- Java Authentication and Authorization Service (JAAS) 1.0

The following technologies previously included with J2EE are now included with the Java 2 Platform, Standard Edition (J2SE): JNDI, RMI-IIOP.

What the development environment supports

The exteNd Director development environment provides built-in support for multiple versions of J2EE, including 1.2 and 1.3. It helps you handle version-related tasks throughout the life cycle of a project, including development, migration, and deployment:

Task	What you can do
Development	You can use the development environment to develop projects for any of the following: ◆ J2EE 1.2 archives ◆ J2EE 1.3 archives This includes the ability to add J2EE 1.2 and 1.3 modules to a 1.3 EAR as well as add 1.*x* and 2.*x* CMP entity beans to a 2.0 EJB JAR. See "Versions for new projects and components" on page 264.
Migration	You can use the development environment to migrate existing projects to a newer J2EE version. Scenarios include: ◆ Migrating a J2EE 1.2 project (and any 1.2 subprojects it contains) to 1.3 ◆ Migrating the J2EE 1.2 subprojects of a 1.3 project to 1.3 See "Migrating projects from J2EE 1.2 to 1.3" on page 266.
Deployment	You can deploy projects to J2EE servers by using the development environment (or, if necessary, vendor deployment tools). Scenarios include: ◆ Deploying a J2EE 1.2 project to a 1.2 or 1.3 server ◆ Deploying a J2EE 1.3 project to a 1.3 server For details on selecting an appropriate J2EE server and deploying to it, see Chapter 15, "Archive Deployment".

Your choices

This section examines specific kinds of projects and offers advice about making J2EE version choices for them. It covers:

Project scenarios

Use the following table to find the project scenarios that apply to you and learn how you can handle them in the exteNd Director development environment:

If you have this	You can
Source files for a J2EE 1.2 module or application (but no project)	Create one or more J2EE 1.2 projects for the source files, specifying appropriate project types and versions. Possible choices are: ◆ EAR 1.2 ◆ WAR 2.2 ◆ EJB 1.1 ◆ CAR 1.2
Source files for a J2EE 1.3 module or application (but no project)	Create one or more J2EE 1.3 projects for the source files, specifying appropriate project types and versions. Possible choices are: ◆ EAR 1.3 ◆ WAR 2.3 ◆ RAR 1.0 ◆ EJB 2.0 ◆ CAR 1.3
Packaged J2EE 1.2 archive (but no source files or project)	Create a J2EE 1.2 deploy-only project for the archive, specifying an appropriate project type and version. Possible choices are: ◆ EAR 1.2 ◆ WAR 2.2 ◆ EJB 1.1 ◆ CAR 1.2
Packaged J2EE 1.3 archive (but no source files or project)	Create a J2EE 1.3 deploy-only project for the archive, specifying an appropriate project type and version. Possible choices are: ◆ EAR 1.3 ◆ WAR 2.3 ◆ RAR 1.0 ◆ EJB 2.0 ◆ CAR 1.3

J2EE 1.2 enterprise archive project (EAR 1.2)	Do any of the following:
	◆ **Edit it** and its subprojects (J2EE 1.2 modules) as needed, while adhering to the J2EE 1.2 APIs and specifications
	◆ **Deploy it** to a J2EE 1.2 or 1.3 server
	◆ **Add subprojects to it** (J2EE modules you add must be 1.2 modules)
	◆ **Migrate it** and its subprojects (J2EE 1.2 modules) from J2EE 1.2 to 1.3:
	  ◆ EAR 1.2 to EAR 1.3
	  ◆ WAR 2.2 to WAR 2.3
	  ◆ EJB 1.1 to EJB 2.0
	  ◆ CAR 1.2 to CAR 1.3

If you have this	You can
J2EE 1.3 enterprise archive project (EAR 1.3)	Do any of the following: ◆ **Edit it** and its subprojects as needed, according to these guidelines: ◆ For the EAR itself and its J2EE 1.3 modules, adhere to the J2EE 1.3 APIs and specifications ◆ For its J2EE 1.2 modules, adhere to the J2EE 1.2 APIs and specifications ◆ **Deploy it** to a J2EE 1.3 server ◆ **Add subprojects to it** (J2EE modules you add can be 1.2 or 1.3 modules) ◆ **Migrate it** if you want to migrate the J2EE 1.2 modules (subprojects) it contains to J2EE 1.3: ◆ WAR 2.2 to WAR 2.3 ◆ EJB 1.1 to EJB 2.0 ◆ CAR 1.2 to CAR 1.3
J2EE 1.2 project for any module: ◆ Web archive (WAR 2.2) ◆ Enterprise JavaBean archive (EJB 1.1) ◆ Application client archive (CAR 1.2)	Do any of the following: ◆ **Edit it** as needed, while adhering to the J2EE 1.2 APIs and specifications ◆ **Deploy it** to a J2EE 1.2 or 1.3 server ◆ **Add it as a subproject** to a J2EE 1.2 or 1.3 EAR project ◆ **Migrate it** from J2EE 1.2 to 1.3: ◆ WAR 2.2 to WAR 2.3 ◆ EJB 1.1 to EJB 2.0 ◆ CAR 1.2 to CAR 1.3
J2EE 1.3 project for any module: ◆ Web archive (WAR 2.3) ◆ Resource adapter archive (RAR 1.0) ◆ Enterprise JavaBean archive (EJB 2.0) ◆ Application client archive (CAR 1.3)	Do any of the following: ◆ **Edit it** as needed, while adhering to the J2EE 1.3 APIs and specifications ◆ **Deploy it** to a J2EE 1.3 server ◆ **Add it as a subproject** to a J2EE 1.3 EAR project
J2EE 1.2 deploy-only project (EAR 1.2, WAR 2.2, EJB 1.1, or CAR 1.2)	Do any of the following: ◆ **Deploy it** to a J2EE 1.2 or 1.3 server ◆ **Add it as a subproject** to a J2EE 1.2 or 1.3 EAR project (if it's a WAR, EJB, or CAR module) ◆ **Migrate it** from J2EE 1.2 to 1.3
J2EE 1.3 deploy-only project (EAR 1.3, WAR 2.3, RAR 1.0, EJB 2.0, or CAR 1.3)	Do either of the following: ◆ **Deploy it** to a J2EE 1.3 server ◆ **Add it as a subproject** to a J2EE 1.3 EAR project (if it's a WAR, RAR, EJB, or CAR module)

Approaching new development

When you start work on a new application or module, a good rule of thumb is to use the latest J2EE version and technologies supported by your target J2EE server(s). This typically provides a more mature platform, including:

- **J2EE improvements** such as additional features, architecture enhancements, and other updates
- **Server improvements** that accompany the newer J2EE implementation, such as faster deployment, better runtime performance, and easier maintenance

For example, if your deployment target is a J2EE 1.3 server, you should develop J2EE 1.3 archives for it (even though 1.2 archives are also supported). Within your archives, it's also recommended that you code to the latest standards available. If you develop EJB entity beans with container-managed persistence, that means using 2.*x* CMP instead of 1.*x* CMP.

Deciding when to migrate

Use the following table to decide when to migrate an existing J2EE archive project to a newer J2EE version:

Decision	When it applies
Migrate	If the project needs to use new features or other enhancements of the more recent J2EE version
	If you want to take advantage of improvements that your J2EE server introduces with its implementation of the more recent J2EE version
Don't migrate	If you need to deploy the project to a J2EE server that does not support the more recent J2EE version

For example, suppose you have a J2EE 1.2 WAR project and want to add servlet filters to it. In that case, you must first migrate this project to J2EE 1.3 (in other words, from WAR 2.2 to WAR 2.3).

Versions for new projects and components

Because the exteNd Director development environment supports multiple versions of J2EE, it offers you a choice of versions when you create parts of an application. This applies:

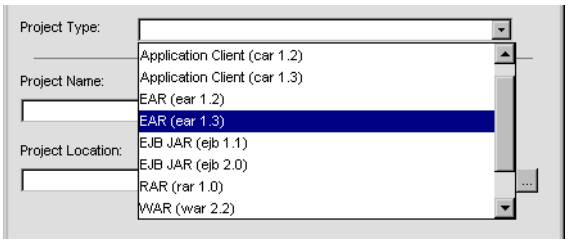
- When creating projects
- When creating JSP tag libraries
- When creating EJB entity beans

When creating projects

When you create a generic J2EE project, you can specify the J2EE version (1.2 or 1.3) of the archive that the project represents. The default is J2EE 1.3.

For example, suppose you need a new Web archive project. To create it, you select **File>New>Project** and choose WAR (on the Generic tab) as the project type. Then the New Project Wizard prompts for details about the project, including its J2EE version:

When you create other types of generic J2EE archive projects (EAR, RAR, EJB, CAR), the New Project Wizard adjusts the list of J2EE version choices accordingly. For deploy-only projects, you specify project type and J2EE version at the same time by choosing from a combined list:



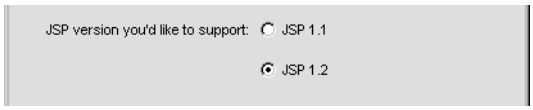
How your J2EE version choice affects a project When a project is created, your J2EE version choice is reflected in several places:

In this part of the project	Your J2EE version choice affects
Project (SPF) file	The following project settings: ◆ The **j2eeVersion** and **moduleVersion** attributes in the SPF file. They record the current version status of the project. For example: `j2eeVersion="j2ee 1.3" moduleVersion="war 2.3"` ◆ The project **classpath** entry for the JAR file that provides the J2EE API packages (needed for compiling). For J2EE 1.2 projects, the entry is `j2ee_api_1_2.jar`; for J2EE 1.3 projects, the entry is `j2ee_api_1_3.jar`.
Deployment descriptor	Which version of the appropriate J2EE deployment descriptor is created for the project. Depending on the project's J2EE version, you get a 1.2 or 1.3 deployment descriptor for your project type: EAR, WAR, RAR, EJB, or CAR. You can examine the DOCTYPE statement of the deployment descriptor to determine which DTD it follows. 📖 For details, see Chapter 18, "J2EE Deployment Descriptor DTDs". (Note that deployment descriptors are not created for deploy-only projects because their archive contents are static.)
Deployment plan (for exteNd Application Server 4.x or 5.x)	Which version of the appropriate exteNd deployment plan you get by default if you create one for the project. ◆ If you specify **SilverStream 4.*x*** as the server type: Depending on the project's J2EE version, the default is a 1.2 or 1.3 deployment plan for your project type: EAR, WAR, RAR, EJB, or CAR. You can examine the DOCTYPE statement of the deployment plan to determine which DTD it follows. ◆ If you specify **Novell exteNd 5.*x*** as the server type: Whether your project's J2EE version is 1.2 or 1.3, you always get a 1.3 deployment plan for your project type. 📖 For details, see Chapter 19, "exteNd Application Server Deployment Plan DTDs".

When creating JSP tag libraries

You can use the Tag Handler Wizard to develop a new or existing JSP tag library. If you ask to create a new tag library, the wizard requires you to specify which JSP version (1.1 or 1.2) to support in it.

To start the Tag Handler Wizard, you select **File>New>File** and choose Tag handler. It then prompts for several panels of information, including details about the tag library descriptor (TLD) file. The JSP version is one of those details (for a new TLD file):



Your JSP version choice is recorded in the TLD file via the **jsp-version** element. This choice also determines which DTD is used for the TLD file:

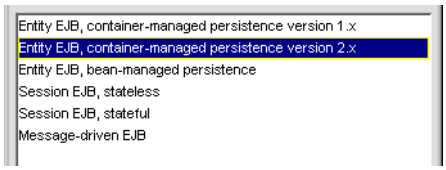
JSP version	TLD DTD
1.1	web-jsptaglibrary_1_1.dtd
1.2	web-jsptaglibrary_1_2.dtd

WAR 2.3 projects support both JSP 1.1 and 1.2 tag libraries (although JSP 1.2 is recommended). WAR 2.2 projects support only JSP 1.1 tag libraries.

When creating EJB entity beans

You can use the EJB Wizard to add various kinds of Enterprise JavaBeans to an EJB project. That includes entity beans using container-managed persistence, with a choice of 1.*x* or 2.*x* CMP.

To start the EJB Wizard, you select **File>New>File** and choose EJB as the component to create. It then prompts for the kind of EJB you want:



For an EJB 2.0 project, you can choose either version of CMP entity bean (although 2.*x* CMP is recommended). Your CMP version choice is recorded in the project's deployment descriptor via the **cmp-version** element.

EJB 1.1 projects support only 1.*x* CMP.

Migrating projects from J2EE 1.2 to 1.3

The exteNd Director development environment provides an **Update Project Version** command that you can select to migrate an existing J2EE 1.2 project to 1.3. For most kinds of projects, this command does the entire migration for you. In a few cases, it will notify you about additional migration tasks that you need to perform manually. The development environment also provides a command for when you just want to update an exteNd deployment plan from J2EE 1.2 to 1.3.

If you have one or more projects to migrate, read the following topics:

- Using the Update Project Version command
- Using the Update Deployment Plan Version command
- Projects that require some manual migration

Using the Update Project Version command

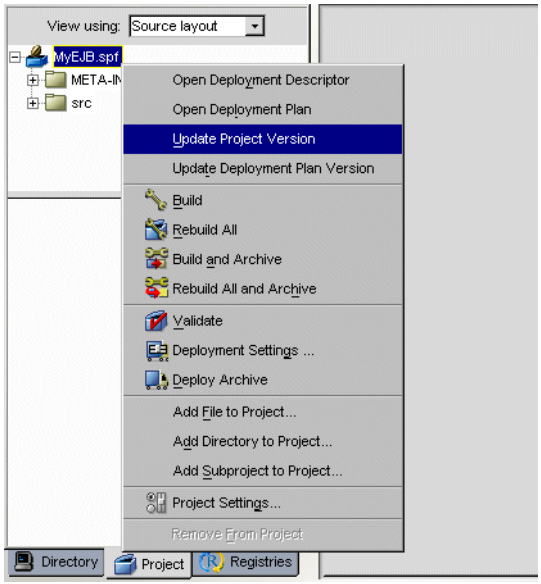
This section describes the steps to follow once you've decided to migrate a project. For background information on when and what to migrate, see these earlier sections:

- "Deciding when to migrate" on page 264
- "Project scenarios" on page 262

➢ **To migrate a project:**

1 In the development environment, open a **J2EE archive project** (EAR, WAR, EJB, or CAR) that you want to migrate.

2 On the Project tab of the Navigation Pane, right-click the **project (SPF) file** to display the popup menu.



3 Select **Update Project Version**.

First, the update utility checks the J2EE version of your selected SPF file. If it is already set to J2EE 1.3, the update utility notifies you, asking if you want to proceed. Migrating such a project can make sense if it includes J2EE 1.2 items (deployment descriptors, deployment plans, or subprojects) that you want to update to 1.3. The update utility will migrate just those items that need it.

Once past the SPF check, the update utility prompts you for confirmation before migrating the project. Files that are modified during the migration will be backed up first **only if** you have enabled backup in your development environment preferences.

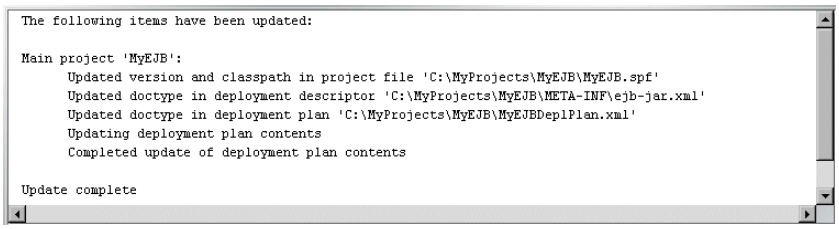
📖 For information about enabling backup, see "Backup preferences" on page 28.

4 Click **Yes** to start the migration.

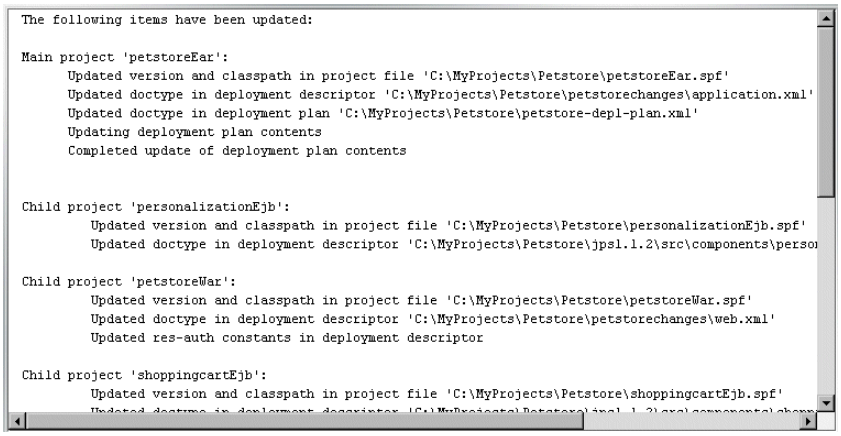
The update utility migrates the project from J2EE 1.2 to 1.3, then displays status information about:

- **Changes it made** to the project
- **Unresolved issues** (if any) that require you to make some manual changes to the project

This status information appears on the Output tab of the Output Pane. For example:



If you migrate an **EAR project**, the update utility automatically migrates the J2EE archive subprojects (WARs, EJBs, and CARs) that the EAR contains as well. The update utility includes details about these migrations in the status information it displays:



How migration affects a project When a project is migrated, the update utility makes changes in several places to reflect the new J2EE version:

In this part of the project	Migration affects

Project (SPF) file	The following project settings:
	◆ The **j2eeVersion** and **moduleVersion** attributes in the SPF file. When you migrate, the update utility sets these to J2EE 1.3 values. For example:
	`j2eeVersion="j2ee 1.3" moduleVersion="war 2.3"`
	◆ The project **classpath** entry for the JAR file that provides the J2EE API packages (needed for compiling). When you migrate, the update utility sets this to:
	`j2ee_api_1_3.jar`

In this part of the project	Migration affects
Deployment descriptor	The following descriptor items: ◆ **DOCTYPE** statement. When you migrate, the update utility edits it to use the 1.3 deployment descriptor DTD for your project type: EAR, WAR, EJB, or CAR. ◆ The **cmp-version** element for entity beans using container-managed persistence. When you migrate an EJB project, the update utility adds this element for each CMP entity bean, specifying the version as 1.*x*. ◆ The **res-auth** element for WARs. When you migrate a WAR project, the update utility finds any of these that use 1.2 constant values (SERVLET or CONTAINER) and edits them to use the 1.3 equivalents (Application or Container). 📖 See Chapter 18, "J2EE Deployment Descriptor DTDs". (Note that deployment descriptors are not updated for deploy-only projects because their archive contents are static.)
Deployment plan, if present (for exteNd Application Server)	The following plan items: ◆ **DOCTYPE** statement. When you migrate, the update utility edits it to use the 1.3 deployment plan DTD for your project type: EAR, WAR, EJB, or CAR. ◆ **EJB plan structure**. When you migrate an EJB project, the update utility restructures the deployment plan in accordance with the 1.3 deployment plan DTD (which has changed significantly to support EJB 2.0 features). One major change involves **persistenceInfo**, a new element for deployment details about entity beans using container-managed persistence. The update utility moves existing field mapping information for CMP entity beans into this element. ◆ **EAR plan structure**. When you migrate an EAR project, the update utility restructures the deployment plan in accordance with the EJB changes described above. 📖 See Chapter 19, "exteNd Application Server Deployment Plan DTDs".

Using the Update Deployment Plan Version command

In addition to the Update Project Version command, the exteNd Director development environment provides an **Update Deployment Plan Version** command on the popup menu for a J2EE project (SPF file). Update Deployment Plan Version works just like Update Project Version except that it only updates the project's exteNd deployment plan.

Update Deployment Plan Version is useful when you want to deploy J2EE 1.2 projects (including deploy-only projects) to Novell exteNd Application Server 5.x, which requires 1.3 deployment plans. When you invoke this command, it migrates the deployment plans of those projects from 1.2 to 1.3 format, but leaves other project characteristics unchanged.

Projects that require some manual migration

When you use the Update Project Version command or the Update Deployment Plan Version command to migrate certain kinds of projects, the update utility may not be able to resolve all issues automatically. In that case, it displays information about changes you need to make to the project yourself.

The need for manual migration typically occurs with **EJB and EAR projects** containing **CMP entity beans**. While updating **exteNd** **deployment plans** for such projects, the update utility does not handle some items, including:

- Complex CMP fields
- CMP fields that map to foreign beans
- Database profiles for persistence information

You should edit the updated deployment plan to specify these items and check for any other changes that may be needed.

exteNd Application Server considerations

This section presents J2EE version issues specific to the **exteNd Application Server** (specifically, Novell exteNd Application Server 5.x and SilverStream eXtend Application Server 4.x). It includes some details on the server's implementation of J2EE 1.2 and 1.3 that may help you make decisions about deploying and migrating projects:

About the J2EE containers

The 4.x and 5.x servers are compatible with J2EE 1.3. To implement this support, they provide J2EE 1.3 versions of the Web, client, and EJB containers:

- The **Web and client containers** for J2EE 1.3 are used when you deploy J2EE 1.3 projects (WAR 2.3, CAR 1.3) as well as J2EE 1.2 projects (WAR 2.2, CAR 1.2).
- The **EJB container** for J2EE 1.3 is used when you deploy J2EE 1.3 projects (EJB 2.0) and, in normal cases, when you deploy J2EE 1.2 projects (EJB 1.1).

  Because of container implementation changes introduced to support EJB 2.0, some EJB 1.1 deployments can't use the J2EE 1.3 version of the EJB container. For these scenarios, the 4.x server also provides a J2EE 1.2 version of this container. In the 5.x server, that older EJB container is no longer provided, so you'll need to adjust such projects before deploying.

Deploying projects

The following table summarizes the possible project deployment scenarios and shows which version of the appropriate container is used in each case:

Project	Deployment plan	Container
WAR 2.2	WAR 2.2 (4.x server only)	J2EE 1.3 Web container
WAR 2.2	WAR 2.3	J2EE 1.3 Web container
WAR 2.3	WAR 2.3	J2EE 1.3 Web container
CAR 1.2	CAR 1.2 (4.x server only)	J2EE 1.3 client container
CAR 1.2	CAR 1.3	J2EE 1.3 client container
CAR 1.3	CAR 1.3	J2EE 1.3 client container
EJB 1.1	EJB 1.1 (4.x server only)	J2EE 1.2 EJB container

Project	Deployment plan	Container
EJB 1.1	EJB 2.0 (for limitations, see EJB deployment notes)	J2EE 1.3 EJB container
EJB 2.0	EJB 2.0 (for limitations, see EJB deployment notes)	J2EE 1.3 EJB container
EAR 1.2	EAR 1.2 (4.x server only)	J2EE 1.3 Web container J2EE 1.3 client container J2EE 1.2 EJB container
EAR 1.2	EAR 1.3 (for limitations, see EJB deployment notes)	J2EE 1.3 Web container J2EE 1.3 client container J2EE 1.3 EJB container
EAR 1.3	EAR 1.3 (for limitations, see EJB deployment notes)	J2EE 1.3 Web container J2EE 1.3 client container J2EE 1.3 EJB container
RAR 1.0	RAR 1.0	J2EE 1.3 containers

EJB deployment notes

In some situations, you may want (or need) to deploy an EJB 1.1 project to the J2EE 1.3 EJB container. One reason is that this lets you take advantage of performance improvements and other implementation enhancements introduced with the newer container version.

For **EJB 1.1 projects containing session beans or BMP entity beans**, you just need to create an EJB 2.0 deployment plan or include the project in a J2EE 1.3 EAR deployment. No other modifications to the project are required. This simple approach is not possible for **EJB 1.1 projects that contain CMP entity beans** and use SilverStream extensions to specify CMP details (such as expressions, foreign bean mappings, and complex fields). These extensions are supported by the EJB 1.1 deployment plan, but not the 2.0 plan.

There are also some limitations to note for **EJB 2.0 projects containing CMP entity beans**. In such projects, beans with 1.*x* CMP can't use elements specific to EJB 2.0 (such as abstract schema names, local interfaces, and queries). In addition, these beans can't use SilverStream extensions to specify CMP details (because, as mentioned above, the EJB 2.0 deployment plan doesn't support them).

IV Deployment

These chapters present the exteNd Director utility tools for deploying J2EE archives:

- Chapter 15, “Archive Deployment”
- Chapter 16, “Deployment Descriptor Editor”
- Chapter 17, “Deployment Plan Editor”
- Chapter 18, “J2EE Deployment Descriptor DTDs”
- Chapter 19, “exteNd Application Server Deployment Plan DTDs”

15 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 the Novell exteNd Director development environment and includes the following topics:

- Supported J2EE servers
- Deployment types
- Deploying J2EE archives
- What happens when you deploy
- Deploying Web Services
- Undeploying archives

Supported J2EE servers

The exteNd Director development environment provides built-in support for deploying archives to the following J2EE servers:

Server	Server archive support
Novell exteNd Application Server 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.
Apache Tomcat	Allows you to directly deploy WARs.

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

NOTE: IBM WebSphere does not provide built-in support for direct deployment. You must use WebSphere's deployment tools.

Deployment types

To deploy the archives you generate in the exteNd Director development environment, you can use the following:

- Rapid deployment
- Production deployment
- External deployment tools

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. The exteNd Director development environment lets you do this using **rapid deployment**. You specify rapid deployment by simply checking a checkbox in the Deployment Settings dialog (described in "Creating deployment settings" on page 278). When you deploy the application, this 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 (such as a WAR tag library or a deployment descriptor), exteNd Director automatically performs a full deployment.

 For more information about setting up a rapid deployment environment, see "Creating deployment settings" on page 278. For more information about the target server's native file system deployment, see "What happens when you deploy" on page 281.

exteNd Director can only support rapid deploy if this feature is supported by the application server vendor. The following table lists the J2EE servers that support a rapid deploy feature and the kind of archives that you can rapid deploy.

Server	EAR	WAR	EJB	CAR	RAR
Novell exteNd Application Server SilverStream eXtend Application Server	Yes	Yes	No	No	No
BEA WebLogic	Yes	Yes	Yes	No	No
Apache Tomcat	No	Yes	No	No	No

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. exteNd Director uses the target server's native deployment tools to deploy the application in the appropriate production deployment directory.

External deployment tools

Alternatively, you can take your generated archives and deploy them outside of the exteNd Director development environment, using instead the deployment facilities provided by your J2EE server. That's because exteNd Director generates standard J2EE archives.

Deploying J2EE archives

To deploy a J2EE archive using the exteNd Director development environment, the **archive** must:

- Be properly structured according to the J2EE specification (see "Archive contents" on page 277)
- Reside in a project (see Chapter 2, "Projects and Archives")

You must supply:

- A server profile (see "Server profile" on page 32)
- Server-specific deployment information (see "Server deployment information" on page 278)
- Deployment settings (see "Creating deployment settings" on page 278)

The **development environment** requires:

- Access to the target server
- Permission to write to the server's deployment area
- Permission to write temporary files when deploying to a Novell exteNd Application Server or SilverStream eXtend Application Server

  exteNd Director 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 defined, it must point to a reachable and writable location. The temporary file location is:

Server	Home environment variable
Novell exteNd Application Server	NOVELL_EXTEND_APPSERVER_HOME\.appsrv
SilverStream eXtend Application Server	SILVERSTREAM_HOME\.silverstream

When all of these requirements are met, see "Deploying a project" on page 281.

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: java.sun.com/j2ee/docs.html.

J2EE module	Standard archive requirements
Application client	A JAR file containing: ◆ 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
EAR	An EAR file containing: ◆ 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
EJB JAR	A JAR file containing: ◆ 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
RAR	A RAR file containing: ◆ The classes needed to implement the resource adapter ◆ A deployment descriptor called ra.xml located in the JAR's /META-INF directory

J2EE module	Standard archive requirements
WAR	A WAR file containing: ◆ 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

For more information, see Chapter 18, "J2EE Deployment Descriptor DTDs".

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 supported server:

J2EE server	Archives	Server deployment information
Novell exteNd Application Server SilverStream eXtend Application Server	Application client (CAR) EAR EJB RAR WAR	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. The server defines a DTD for each archive type. For more information, see Chapter 19, "exteNd Application Server Deployment Plan DTDs". You use the **Deployment Plan Editor** to create and populate the deployment plan. For more information, see "Deployment Plan Editor" on page 289.
BEA WebLogic	Application client EAR EJB RAR WAR	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. For more information, see your WebLogic documentation.
Apache Tomcat	WAR	No specific file is needed.

When specifying server deployment information, you can override information in your J2EE deployment descriptors, enabling you to customize a particular deployment as needed.

Creating deployment settings

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

➢ **To create deployment settings:**

1 Choose **Project>Deployment Settings**.

NOTE: If you are deploying to an exteNd application server and the project's current deployment plan is not associated with a server profile, you will be told that you need to specify one in the Deployment Settings dialog.

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 **New** to create a new profile. 📖 For more information on server profiles, see "Server profile" on page 32.
Use this server profile as the default for all projects	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 for servers that support rapid deployment:

Option	What to do
Enable Rapid Deployment	Check this box when you want to deploy the archive using the rapid deployment feature for testing. Uncheck it when you want to do a production deploy. **What happens** When this checkbox is checked, exteNd Director writes files to the rapid deployment directory specified in the server profile. **NOTE:** If you have not set a rapid deployment directory in the server profile, you are prompted for one. This directory is a location on disk where you want the server to write the deployment files and is defined by the application server vendor. Many servers require a specific directory; see "Server profile" on page 32 for the list. **Further action** exteNd Director manages updates to the deployment area on subsequent rapid deploys. You do not have to do any manual procedure (as you would have to when directly using the server's rapid deployment). **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. 📖 For more information on rapid deployment and how each server supports this feature (plus any special requirements), see "What happens when you deploy" on page 281.

5 Specify server-specific information.

For **Novell exteNd application servers** and **SilverStream eXtend application servers**, specify the following:

Option	What to do
Deployment Plan	Specify the file name and disk location of the deployment plan.
Overwrite existing deployment	Check this box when you want the current deployment to overwrite any previously deployed objects of the same type and name. If you deselect this box and objects of the same name and type already exist on the server, the deployment will fail.
Verbosity	Specify the level of informational messages to display. Values range from 0 (for no messages) to 5 (for the most messages).

Option	What to do
Ignore JSP compile errors	Applies only to WARs and to EARs containing WARs. Check this box when you want the deployment to ignore any errors when compiling JSP pages and to deploy only those items that build successfully. If this box is not checked and a compile error occurs, deployment fails.
SilverCmd Flags	(Optional) Specify command-line arguments for the deployment command. 📖 For more information on the deployment commands that are executed, see "What happens when you deploy" on page 281. If you specify multiple arguments, use spaces as the delimiters. If you want to pass VM arguments, you must precede them with +. For example: `+Xmx256` All of the values entered here are appended to the end of the deployment command that gets constructed.

For **BEA WebLogic servers**, specify:

Option	What to do
WebLogic Application Name	Specify the deployment name for your application; this is the name your users will use in the URL for the application. If this is a rapid deploy, this is the directory name under the deployment directory. The default is the project name.
Generate Targets	Click 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: ◆ Accept the values created when the Generate Targets button is clicked. ◆ Edit the values created when the Generate Targets button is clicked. ◆ Manually type the names of the components and their target servers using proper WebLogic syntax.
Deployment options	Choose one of these options: ◆ **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. (Not available for all server versions.) ◆ **undeploy**—Disables the application with the option to delete it. ◆ **list**—Provides a list of all deployed applications on the server specified by the current project's server profile. ◆ **user defined**—Allows you to specify a WebLogic command. When this value is specified, exteNd Director uses only the command in the **WebLogic options** text field (described below).

Option	What to do
JVM options	Specify command-line arguments for the server's JVM using spaces to delimit the options. These values are appended to the end of the deployment command that gets constructed.
WebLogic options	Specify server-specific options using spaces to delimit them. You must supply the complete command for the action you want the WebLogic server to perform. For example: `-unprepare -name War23` Make sure that **user defined** (under Deployment options) is selected or exteNd Director ignores the values you enter in this text box. These values are appended to the end of the deployment command that gets constructed. exteNd Director does no error checking, but error messages returned by the server are displayed in the Output Pane.
debug	Check this option when you want to see the debug information produced by the WebLogic deploy tool.

6 Click **OK** to store the deployment settings with the project file.

Deploying a project

➢ **To deploy a project:**

1 Open the project.

Any archive that you want to deploy must be defined in a project. If you created the archive using another IDE, you must create a 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 278.

3 Define the deployment settings for the project.

📖 For more information, see "Creating deployment settings" on page 278.

4 Select **Project>Deploy Archive**.

NOTE: The deployment fails if your server is not running.

What happens when you deploy

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

1 It 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, it 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
Novell exteNd Application Server SilverStream eXtend Application Server	CAR	**Standard/Production deploy:** SilverCmd DeployCAR **Rapid deploy:** Not supported for CARs
	EAR	**Standard/Production deploy:** SilverCmd DeployEAR **NOTE:** To deploy individual modules from within an EAR, pass the -m option as a SilverCmd Flag. **Rapid deploy:** Supports the rapid deployment of WARs in the EAR. It works like this: ◆ When a JSP page, an HTML page, a CLASS file, or a JAR file in a WAR within the EAR changes, exteNd Director 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), exteNd Director invokes the standard/production deployment. ◆ exteNd Director 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:** Not supported for EJBs
	WAR	**Standard/Production deploy:** SilverCmd DeployWAR **Rapid deploy:** JSP/FS During the JSP/FS process: ◆ exteNd Director expands the WAR file in the server's /webapps/*DBname*/*URL* directory, where *DBname* is the name of the database containing the application deployed to the file system, and *URL* is the URL specified in the deployment plan for the application (if you have specified more than one, the first one is used). ◆ exteNd Director 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). ◆ exteNd Director updates the <deployToFileSystem> attribute automatically when you specify a rapid deploy.
BEA WebLogic	All supported archives	**Standard/Production deploy:** weblogic.deploy or weblogic.deployer (depending on the server version) **Rapid deploy:** The exteNd Director development environment uses the server's native utilities to provide rapid deployment of EARs, EJBs, and WARs. You'll need to enable WebLogic Auto-Deployment through the WebLogic Management console before performing a rapid deploy. For more information on setting Auto-Deployment, see your WebLogic documentation. ◆ During a rapid deploy, exteNd Director copies the modified files to the user-specified deployment directory and touches the REDEPLOY file.

Server	Archive	Deploy command description
Apache Tomcat	WAR	**Standard/Production deploy:** copy ◆ Copies the archive to the server's \webapps directory ◆ You must restart the server after a standard deploy **Rapid deploy:** copy ◆ 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

3 The target server's deployment command creates the appropriate deployment objects on the target server.

4 exteNd Director displays a message stating the status (success/failure) or any warning/error messages in the Output tab of the Output Pane.

Deploying Web Services

When you create a Web Service in the exteNd Director development environment by using the Web Service Wizard or by using the Novell exteNd Web Services SDK 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 archives required by the Web Services SDK:

For a list of the JARs required by	See
A Web Service	Preparing to generate in the chapter on Generating Web Services
A Web Service consumer	Preparing to generate in the chapter on Generating Web Service Consumers

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
- Apache Tomcat

If you deploy to the Novell exteNd Application Server or 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 your server's help.)

You can obtain the required JARs by copying them from the Novell exteNd **tools\compilelib** directory.

Undeploying archives

Depending on the deployment server, you can **disable** or **delete** deployed archives on the server from within the exteNd Director development environment.

exteNd Director 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 the development environment.

Typically, disabling leaves the files on the server but makes them unavailable, and deleting physically removes the files from the server. However, since exteNd Director 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's a summary of the server undeployment support provided in the exteNd Director development environment:

Server	Disable?	Delete?	Notes
Novell exteNd Application Server SilverStream eXtend Application Server	No	Yes	
BEA WebLogic	Yes	Yes	
Apache Tomcat	No	Yes	Undeploy deletes the WAR and the web application directory from the webapps directory

➢ **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 issued by looking in the Output tab of the Output Pane.

16 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

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 (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 of a deployment descriptor for each J2EE archive. For more information, see Chapter 18, "J2EE Deployment Descriptor DTDs".

How deployment descriptors are created exteNd Director automatically constructs and adds a J2EE-compatible deployment descriptor file to your project in the appropriate location, as follows:

J2EE archive	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

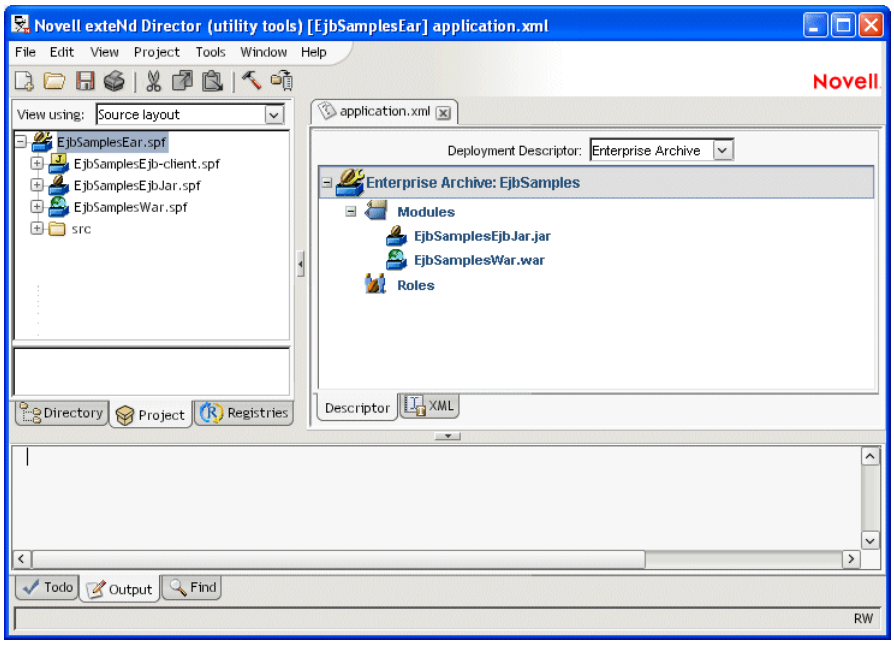
As you add J2EE components to a project, exteNd Director adds the corresponding elements to the deployment descriptor when it has enough information to do so.

About the Deployment Descriptor Editor

The Deployment Descriptor Editor allows you to fine-tune the deployment descriptor by modifying or completing entries that exteNd Director 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's a sample of the Deployment Descriptor Editor for an EAR 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) that was in use when you last saved.

Nodes displayed in **bold** (such as Modules and Roles) 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, which you can provide by completing a property sheet. To display the Property Inspector for a node, highlight the node, right-click, and select **Properties**.

To save your 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 exteNd Director 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>File**.
3. Select the **General tab**.
4. Select **Deployment Descriptor** and click **OK**.

   This 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 of the exteNd Director development environment, you can still use it with a project by following these steps.

1 Open the project that you want to associate the deployment descriptor with.

2 Go to the **Directory Pane**.

3 Double-click the deployment descriptor you want.

You are prompted to associate the descriptor with the current project or 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 click **OK**.

This opens the deployment descriptor in the Deployment Descriptor 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 (SPF) file, right-click, and select **Open Deployment Descriptor** from the popup menu.

You are prompted for build preferences. Once you specify your 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 Untitled*XXX*.

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 the 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, that means 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 (from graphical to XML and vice versa)	Checks the syntax of the deployment descriptor
Save a deployment descriptor	Validates the deployment descriptor against the corresponding DTD

But you can force validation anytime.

➢ **To force validation of a deployment descriptor:**

- Select **Validate Archive** from the Project menu.

This validates both the deployment descriptor and the archive.

17 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 an exteNd application server. This chapter describes how to use the Deployment Plan Editor and includes these topics:

About deployment plans

A **deployment plan** is an XML document that describes how a J2EE module (such as a WAR or an EJB JAR) or application (EAR) should run in the Novell exteNd Application Server or the SilverStream eXtend Application Server environment.

A 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 data sources or map roles to users and groups. Settings in a deployment plan override those in the deployment descriptor, enabling you to customize a particular deployment as needed.

 For more information on the contents and structure of the deployment plans for each J2EE archive, see Chapter 19, “exteNd Application Server Deployment Plan DTDs”.

NOTE: Other J2EE servers require different types of information (possibly in different formats) for deployment. To deploy J2EE archives on another J2EE server, see the server vendor’s documentation.

Using the Deployment Plan Editor

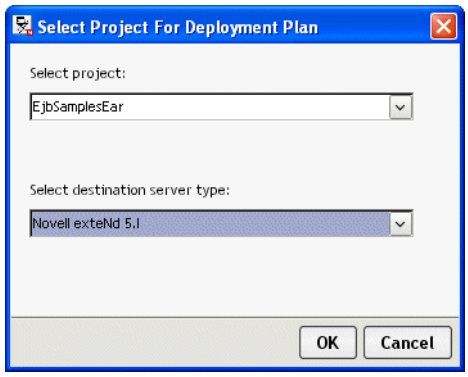
This section describes how to use the Deployment Plan Editor to perform these tasks:

- Create a deployment plan
- Modify a deployment 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. Select **File>New>File**.
4. Select the **General tab**.
5. Select **exteNd Server Deployment Plan** and click **OK**.

The Select Project For Deployment Plan dialog displays.



6 Choose a project from the **Select project** dropdown.

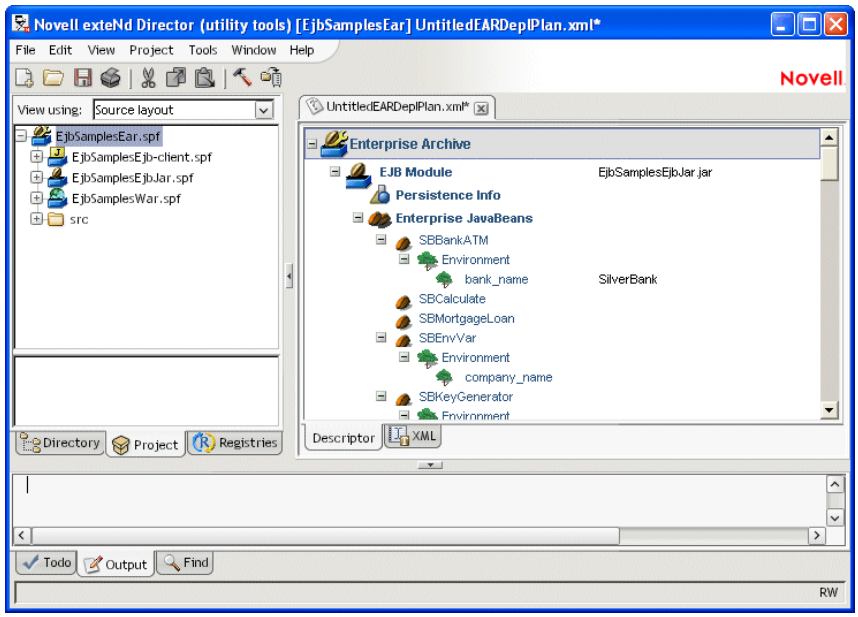
NOTE: When the project is an EAR, you see multiple files in the dropdown.

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 (**Tools>Preferences**). For more information, see "Deployment preferences" on page 31.

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 elements to create. (If you later change the deployment descriptor, exteNd Director updates the deployment plan accordingly the next time you edit it.)

The deployment plan elements are displayed in a tree structure. Here's a sample deployment plan for an EAR:



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) that was in use when you last saved.

8 Select **File>Save** (or click the Save icon).

If there are other 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 (SPF) file, right-click, and select **Open Deployment Plan** from the popup menu.

3 If your project has more than one deployment plan, choose the one you want from the dropdown and click **OK**.

This 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 of the exteNd Director development environment, you can still use it with a project by following these steps.

1 Open the project you want to associate the deployment plan with.

2 Go to the **Directory Pane**.

3 Double-click the deployment plan you want.

You are prompted to associate the plan with the current project or 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 click **OK**.

This opens the deployment plan in the Deployment Plan Editor.

5 Save the deployment plan to complete the association.

6 Click **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 (from graphical to XML and vice versa)	Checks the syntax of the deployment plan
Save a deployment plan	Validates the deployment plan against the corresponding DTD

But you can force validation anytime.

➢ **To force validation of a deployment plan:**

- Select **Validate Archive** from the Project menu.

This validates both the deployment plan and the archive.

18 J2EE Deployment Descriptor DTDs

This chapter provides information about the DTDs for the standard J2EE deployment descriptors. It includes these sections:

- DTD files
- Location
- Use
- Documentation

DTD files

Each type of J2EE archive has its own corresponding deployment descriptor and DTD:

J2EE version	Deployment descriptor type	DTD file	
1.2	EAR	application_1_2.dtd	view
	WAR	web-app_2_2.dtd	view
	EJB JAR	ejb-jar_1_1.dtd	view
	Client JAR	application-client_1_2.dtd	view
1.3	EAR	application_1_3.dtd	view
	WAR	web-app_2_3.dtd	view
	RAR	connector_1_0.dtd	view
	EJB JAR	ejb-jar_2_0.dtd	view
	Client JAR	application-client_1_3.dtd	view

Location

These DTD files are located in the lib\dtds directory of your J2EE SDK installation. Copies of them are also provided in the Novell exteNd Common\Resources\DTDCatalog directory.

Use

The deployment descriptor DTDs are used when you package J2EE archives. These DTDs (XML document type definitions) describe the structure you must follow when writing deployment descriptors (XML files) to supply declarative data and assembly instructions for your archives.

The deployment descriptors you write must adhere to these guidelines:

Deployment descriptor type	Guidelines

EAR	**File name:** application.xml
	Stored in: META-INF directory of the archive
	DOCTYPE statement (J2EE 1.2):
	`<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc. //DTD J2EE Application 1.2//EN" "http://java.sun.com/j2ee/dtds/application_1_2.dtd">`
	DOCTYPE statement (J2EE 1.3):
	`<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN" "http://java.sun.com/dtd/application_1_3.dtd">`

WAR	**File name:** web.xml
	Stored in: WEB-INF directory of the archive
	DOCTYPE statement (J2EE 1.2):
	`<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">`
	DOCTYPE statement (J2EE 1.3):
	`<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">`

RAR	**File name:** ra.xml **Stored in:** META-INF directory of the archive **DOCTYPE statement (J2EE 1.3):** `<!DOCTYPE connector PUBLIC "-//Sun Microsystems, Inc.//DTD Connector 1.0//EN" "http://java.sun.com/dtd/connector_1_0.dtd">`

EJB JAR	**File name:** ejb-jar.xml
	Stored in: META-INF directory of the archive
	DOCTYPE statement (J2EE 1.2):
	`<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">`
	DOCTYPE statement (J2EE 1.3):
	`<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">`

Client JAR	**File name:** application-client.xml
	Stored in: META-INF directory of the archive
	DOCTYPE statement (J2EE 1.2):
	`<!DOCTYPE application-client PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application Client 1.2//EN" "http://java.sun.com/j2ee/dtds/application-client_1_2.dtd">`
	DOCTYPE statement (J2EE 1.3):
	`<!DOCTYPE application-client PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application Client 1.3//EN" "http://java.sun.com/dtd/application-client_1_3.dtd">`

Documentation

DTDs For detailed information on these DTDs (including the elements they define for use in your deployment descriptors):

1. Go to the table of DTD files at the beginning of this chapter.
2. Click **view** to display a DTD file (the copy provided with the Novell exteNd Director development environment).
3. Read the comments in the DTD file (if you need additional DTD documentation, see the Sun J2EE specifications).

Support for deployment descriptors To learn about development environment support for writing and using deployment descriptors, see:

- Chapter 15, “Archive Deployment”
- Chapter 16, “Deployment Descriptor Editor”

19 exteNd Application Server Deployment Plan DTDs

Deployment plan DTD files are used when you deploy J2EE archives to the Novell exteNd Application Server or the SilverStream eXtend Application Server. The DTDs (XML document type definitions) describe the structure you must follow when writing deployment plans (XML files) for particular archives.

This chapter includes these topics:

DTD location

The DTD files are located in the Novell exteNd Common\Resources\DTDCatalog directory. You can learn about the DTDs by looking at:

- The DTD files themselves (for comments about the elements they define)
- The chapter on deployment plan DTDs in the application server's help system (for more detailed reference documentation about each element and how to structure them in your XML files)

To learn about support for writing and using deployment plans in the exteNd Director development environment, see:

- Chapter 15, "Archive Deployment"
- Chapter 17, "Deployment Plan Editor"

DTD files

For each type of J2EE archive, there's a corresponding deployment plan DTD. The specifics of the DTD depend on which server and version you are deploying to. This section includes:

Novell exteNd Application Server DTD files

They are as follows:

Server version	Deployment plan type	DTD file	
5.x	EAR	deploy-ear_1_3.dtd	view
	WAR	deploy-war_2_3.dtd	view
	EJB JAR	deploy-ejb_2_0.dtd	view
	Client JAR	deploy-car_1_3.dtd	view
	RAR	deploy_rar_1_0.dtd	view

SilverStream eXtend Application Server DTD files

They are as follows:

Server version	Deployment plan type	DTD file	
4.x	EAR	deploy-ear_1_3.dtd	view
	WAR	deploy-war_2_3.dtd	view
	EJB JAR	deploy-ejb_2_0.dtd	view
	Client JAR	deploy-car_1_3.dtd	view
	RAR	deploy_rar_1_0.dtd	view

DOCTYPE statements

The deployment plan must reference the correct DOCTYPE.

Authoring the plan with the Deployment Plan Editor When you author the deployment plan using the Deployment Plan Editor in the exteNd Director development environment, the editor inserts the correct DOCTYPE statement based on the project's server profile. If you change the server profile, the Deployment Plan Editor will ask if it should update the deployment plan's DOCTYPE statement.

Authoring the plan with your own editor If you author the deployment plan in a different editor, make sure it references the correct DOCTYPE. The DTD files supplied with the exteNd Director development environment include the DOCTYPE statement as a comment at the beginning of the file. This DOCTYPE is for use with the Novell exteNd Application Server. If you deploy to the SilverStream eXtend Application Server, see the section below for the correct statement to use.

SilverStream eXtend Application Server DOCTYPE statements

These are the DOCTYPE statements for SilverStream eXtend Application Server 4.x:

Deployment plan type	DOCTYPE statement
Application client	`<!DOCTYPE carJarOptions PUBLIC "-//SilverStream Software,Inc. //DTD J2EE CAR Deployment Plan 1.3//EN" "deploy-car_1_3.dtd">`
EAR	`<!DOCTYPE earJarOptions PUBLIC "-//SilverStream Software,Inc. //DTD J2EE EAR Deployment Plan 1.3//EN" "deploy-ear_1_3.dtd">`
EJB	`<!DOCTYPE ejbJarOptions PUBLIC "-//SilverStream Software,Inc. //DTD J2EE EJB Deployment Plan 2.0//EN" "deploy-ejb_2_0.dtd">`
RAR	`<!DOCTYPE rarJarOptions PUBLIC "-//SilverStream Software,Inc. //DTD J2EE RAR Deployment Plan 1.0//EN" "deploy-rar_1_0.dtd">`
WAR	`<!DOCTYPE warJarOptions PUBLIC "-//SilverStream Software,Inc. //DTD J2EE WAR Deployment Plan 2.3//EN" "deploy-war_2_3.dtd">`

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X