Oracle® Fusion Middleware

User's Guide for Oracle JDeveloper

11*g* Release 2 (11.1.2.1.0)

E17455-02

September 2011

ORACLE®

Oracle Fusion Middleware User's Guide for Oracle JDeveloper 11*g* Release 2 (11.1.2.1.0)

E17455-02

Primary Author: Catherine Pickersgill, David Mathews, Deepanjan Dey, Penny Anderson, Scott Fisher, Elizabeth Lynch, Ben Gelernter, Robin Merrin

Contents

3 Working with Oracle JDeveloper

6 Versioning Applications with Source Control

7 Building, Running and Debugging Applications

8 Auditing and Profiling Applications

9 Deploying Applications

Part III Developing Java EE Applications

10 Getting Started with Developing Java EE Applications

11 Developing Applications Using Web Page Tools

12 Developing with EJB and JPA Components

13 Developing TopLink Mappings

14 Developing Secure Applications

15 Developing Applications Using XML

16 Developing Applications Using Web Services

Part IV Developing Java Applications

17 Getting Started with Developing Java Applications

18 Programming in Java

19 Running and Debugging Java Programs

20 Implementing Java Swing User Interfaces

23 Developing Java EE and Java Applications Using Modeling

27 Designing Databases Within Oracle JDeveloper

28 Using Java in the Database

29 Running and Debugging PL/SQL and Java Stored Procedures

Preface

Welcome to the *User's Guide for Oracle JDeveloper*.

Audience

This document is intended for developers that use Oracle JDeveloper and provides detailed information on the functionality available in IDE.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

- *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*
- *Oracle Fusion Middleware Developer's Guide for Oracle JDeveloper Extensions*
- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*
- Oracle JDeveloper *11g Online Help*
- Oracle JDeveloper *11g Release Notes*, link included with your Oracle JDeveloper 11*g* installation, and on Oracle Technology Network

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide in Release 11.1.2.1.0

For Release 11.1.2.1.0 the *Oracle Fusion Middleware User Guide for Oracle JDeveloper* replaces the non-context sensitive online help that was available in previous releases of Oracle JDeveloper.

While this guide is applicable to both the Studio and Java editions of Oracle JDeveloper, the following chapters only apply to the Studio edition of JDeveloper:

- Chapter 9, "Deploying Applications"
- Chapter 11, "Developing Applications Using Web Page Tools"
- Chapter 12, "Developing with EJB and JPA Components"
- Chapter 13, "Developing TopLink Mappings"
- Chapter 14, "Developing Secure Applications"
- All chapters in Part V, "Developing Applications Using Modeling"
- All chapters in Part VI, "Working with Databases"

For changes made to Oracle JDeveloper and Oracle Application Development Framework (Oracle ADF) for this release, see the What's New page on the Oracle Technology Network at `http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html`.

Part I

Getting Started with Oracle JDeveloper

This part describes the general concepts of working with Oracle JDeveloper and contains the following chapters:

- Chapter 1, "Introduction to Oracle JDeveloper"

  This chapter provides an introduction to JDeveloper, including resources available for you to learn Oracle JDeveloper and understand its features.

- Chapter 2, "Oracle JDeveloper Accessibility Information"

  This chapter provides information on the accessibility features of JDeveloper.

- Chapter 3, "Working with Oracle JDeveloper"

  This chapter provides information on working in the JDeveloper IDE.

1 Introduction to Oracle JDeveloper

This chapter provides an overview of Oracle JDeveloper.

This chapter includes the following sections:

1.1 About Oracle JDeveloper

JDeveloper is an integrated development environment (IDE) for building applications using the latest standards for Java, XML, Web services, and SQL. It supports the complete development lifecycle with integrated features for modeling, coding, debugging, testing, profiling, tuning, and deploying applications. JDeveloper is the main development platform for the Oracle Fusion Middleware suite of products. It is a cross-platform IDE that runs on Windows, Linux, Mac OS X, and other UNIX-based systems.

Oracle JDeveloper provides a visual and declarative development approach and works together with the Oracle ADF to simplify development.

Key features of JDeveloper include:

- A consistent development environment that can be used for various technology stacks including Java, SOA, Oracle WebCenter Portal, SQL and PL/SQL, HTML, and JavaScript.
- XML-based application development.
- A full development and modeling environment for building database objects and stored procedures.
- A wide range of application deployment options, including Integrated Oracle WebLogic Server, an integrated run time service for running and testing applications before deploying to a production environment.
- Extension capabilities that enable customization of the IDE based on development needs and add additional functionality.

JDeveloper is available in two editions: Oracle JDeveloper Studio and Oracle JDeveloper Java. The Studio edition is the complete version of JDeveloper and includes all features. The Java edition contains only the core Java and XML features, and offers shorter download times. This guide is applicable to both editions of JDeveloper.

1.2 Oracle JDeveloper Information Resources

This section provides resources designed to get you up and running quickly on Oracle JDeveloper. You can learn about Oracle JDeveloper using various methods in addition to this guide, including online demonstrations, tutorials, and the Oracle Technology Network (OTN) forum. For more information, see Table 1–1, " Supporting Oracle JDeveloper Resources".

Table 1–1 Supporting Oracle JDeveloper Resources

Resource	Description
Online demonstrations	Online demonstrations provide visual instructions for completing common tasks. All you need to watch the demos is your web browser with flash plug-in and a sound card. You can use the playback bar at the bottom of each demo to control the speed and flow of the demo. The demos are located at: http://www.oracle.com/technetwork/developer-tools/jdev/overview/index-100269.html
Oracle JDeveloper **Tutorials**	The tutorials provide step-by-step instructions to accomplish specific tasks in Oracle JDeveloper. The tutorials are located at: http://www.oracle.com/technetwork/developer-tools/jdev/overview/index-100269.html
Oracle Fusion Order Demo Sample Application	The Fusion Order Demo (FOD) is an end-to-end application sample application developed with the purpose of demonstrating common use cases in Fusion Middleware applications, including the integration between different components of the Fusion technology stack, (ADF, BPEL, and WebCenter Portal). The demo contains several applications that make up various parts of functionality. The FOD is located at: http://www.oracle.com/technetwork/developer-tools/jdev/index-095536.html
OTN Oracle JDeveloper **Forum**	You can use the Oracle JDeveloper page on the OTN forum to ask a question, contribute to a discussion, or interact with other users. The Oracle JDeveloper page on the OTN forum is located at: http://forums.oracle.com/forums/forum.jspa?forumID=83

1.3 Migrating to Oracle JDeveloper 11*g*

For complete information on supported migration paths, on how to migrate applications and projects or information about importing preferences and settings from an earlier version of Oracle JDeveloper to Oracle JDeveloper 11*g*, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

2

Oracle JDeveloper Accessibility Information

This chapter provides information on the accessibility features of Oracle JDeveloper.

This chapter includes the following sections:

2.1 About Oracle JDeveloper Accessibility

It is our goal to make Oracle Products, Services, and supporting documentation accessible to the disabled community. Oracle JDeveloper supports accessibility features.

For additional accessibility information for Oracle products, see the Oracle Accessibility Program page at:

`http://www.oracle.com/accessibility/`

2.2 Using a Screen Reader and Java Access Bridge with Oracle JDeveloper

In order for assistive technologies, like screen readers, to work with Java-based applications and applets, the Windows-based computer must also have Sun's Java Access Bridge installed. Please refer to the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper* for the screen reader setup procedure, and for the recommended minimum technology stack.

2.3 Oracle JDeveloper Features That Support Accessibility

Oracle JDeveloper provides features that are designed to support accessibility.

2.3.1 Keyboard Access

Oracle JDeveloper features support keyboard access to JDeveloper functionality; a summary is provided below. The mnemonic keys used to open menus and choose commands are included in all procedural topics. Please refer to the keyboard

navigation topics for a summary of how keys are assigned within JDeveloper and the lists of accelerator keys provided for commands.

The following menu and toolbar functionality is provided through keyboard access:

- Users can navigate to and invoke all menu items.
- All toolbar functions are accessible through menu items.
- All menus and menu items have unique and functioning mnemonic keys.
- All context menus within the navigators and source editor can be invoked.
- Frequently used menu items have unique accelerator keys.

The following functionality is available in JDeveloper IDE windows, which include the Application Navigator, Structure window, source editor, Property Inspector, Constraints, Profilers, Debugger windows, Help windows, Log windows and BC4J Tester. Users can:

- Navigate between all open windows, to all nodes within a window or pane, and between tabs in a window.
- Set focus in a window or pane.
- Invoke all controls within a window or pane, and perform basic operations.
- Navigate and update properties in the Property Inspector.
- Use Code Insight and Code Templates in the source editor.
- Invoke context sensitive help topics, navigate to and open all help topics, and navigate between the navigation and viewer tabs.
- Open, close, dock, undock, minimize, restore and maximize the applicable JDeveloper window.

Tips:
- You can press Escape to move the focus from the current dockable window to the last active editor. Press Shift+Escape to move the focus and also close the current window.
- You can press Shift+F10 to open the context menu for any window. Use the Down Arrow and Up arrow keys to select a command and press Enter, or use the accelerators to invoke a command on the context menu.

The following functionality is available in Oracle JDeveloper dialogs and wizards:

- Users can navigate to and invoke all controls within all wizards and dialogs.
- The order in which the Tab key causes focus to flow is consistent and logical.
- Mnemonic keys are provided for controls where appropriate.

Navigation and controls are available with runtime applications, which include all runnable files that are produced with Oracle JDeveloper, including Java applications, HTML applications, applets, JSF (Faces) applications, JSPs, and Servlets. With runtime applications, users can:

- Navigate to all controls within all runtime applications.
- Invoke all controls within all runtime applications.

2.3.2 Screen Reader Readability

Here is a summary of screen readability in JDeveloper, when it is used with a screen reader.

When used with menus and toolbars:

- All menus and menu items are read.
- All toolbar items, including the Navigator toolbar items, are read.
- The hint text on all toolbar items is read.

When used with JDeveloper IDE windows:

- All open windows are read.
- All components within each window, including tabs, are read.
- Status text at the bottom of the IDE, and within the source editor, is read.

When used with dialogs and wizards:

- All controls within all wizards and dialogs are read.
- Hint text is read.

When used with runtime applications:

- All controls within all runtime applications are read.

2.3.3 Flexibility in Font and Color Choices

The user interface in JDeveloper improves usability for people who are visually impaired by offering flexibility in color and font choices. The following font and color features are included:

- Users can specify both the font and the size in which the font displays for editors.
- All features of the product have black text on a white or gray background.
- Colored text, underlining or images are never used as the only method of conveying information.

2.3.4 No Audio-only Feedback

In JDeveloper, there is no situation in which the only feedback a user receives is audible feedback. All audible feedback is accompanied by a visual indicator. For example, a prompt accompanies the bell sound that occurs when an error or illegal action has taken place.

2.3.5 No Dependency on Blinking Cursor and Animation

JDeveloper makes minimal use of a blinking cursor and animation:

- No features in JDeveloper use blinking indicators, with the exception of the cursor in the source editor.
- No features rely on animated sequences.

2.3.6 Screen Magnifier Usability

The JDeveloper user interface works well with screen magnifiers. All features of the product can be magnified by a screen magnifier.

2.3.7 How to Change the Editor or Tabbed View of a File

When you press Enter on a node in the Application Navigator, you open the default editor for that file. To switch to the different editors and views available for a document; for example, to display a JSP file in source view or history view instead of design view, you can use the Alt+Page Up and Alt+Page Down accelerators to invoke the **Window > Go to > Right Editor** and **Window > Go to > Left Editor** menu commands, respectively.

2.3.8 How to Read Text in a Multi-line Edit Field

To have the text in a multi-line edit field read by a screen reader, you can select text by holding down the Shift key while moving the cursor either up or down with the Arrow keys, depending on the initial cursor position.

2.3.9 How to Read the Line Number in the Source Editor

To have the line number read by a screen reader while you are editing a file in the source editor, you can press Ctrl+G.

2.3.10 How to Access Exception Stack HTML Links and Generated Javadoc Links in the Log Window

After generating exception stack HTML links or Javadoc links in the Log window, they will not be recognized as links, but read as plain text by a screen reader. To access the links, set the cursor focus to the Log window. Right-click or press Shift+F1 and select **Save As** from the context menu. Save the contents of the Log window as an HTML file. Add the saved HTML file to a project or application workspace as a resource. Open the file from the Application Navigator in order to invoke the Oracle JDeveloper HTML/JSP visual editor, which will display the links correctly. Navigate the file and access the links from the HTML/JSP visual editor.

2.4 Recommendations for Customizing Oracle JDeveloper

JDeveloper provides a number of customization features that enable users to specify their requirements for keyboard usage, display attributes of the IDE, and timing where appropriate. All customization features are organized within the Preferences dialog. For maximum usability and to accommodate your needs, you should consider changing any of the following from the defaults to a more usable customized setting.

2.4.1 How to Customize the Accelerators Keys

You can add and change the default accelerator keys for Oracle JDeveloper in the **Tools > Preferences > Shortcut Keys** page. You can also load preset keymaps that you are accustomed to using.

2.4.2 How to Pass a Conflicting Accelerator Key to Oracle JDeveloper

In addition to changing the mapped accelerator keys, you can pass a conflicting accelerator key to JAWS by preceding the accelerator key combination with Insert+F3.

2.4.3 How to Change the Look and Feel of the IDE

You can change the default look and feel for Oracle JDeveloper in the **Tools > Preferences > Environment** page. The look and feel determines the display colors and shapes of objects like menus and buttons.

2.4.4 How to Customize the Fonts in Editors

You can change the font and font size that display in editors in the **Tools > Preferences > Code Editor > Fonts** page.

2.4.5 How to Customize Syntax Highlighting

You can change the font style, as well as the foreground and background colors used in syntax highlighting within the source editor in the **Tools > Preferences > Code Editor > Syntax Colors** page.

2.4.6 How to Display Line Numbers in Editors

You can display or hide line numbers in the source editor in the **Tools > Preferences > Code Editor > Line Gutter** page.

2.4.7 How to Change the Timing for Code Insight

You can specify the number of seconds that Code Insight is delayed, or disable Code Insight in the **Tools > Preferences > Code Editor > Code Insight** page.

2.4.8 How to Specify the Columns in the Debugger

You can choose the columns and types of information that display in the Debugger in the **Tools > Preferences > Debugger** pages.

2.5 Highly Visual Features of Oracle JDeveloper

Oracle JDeveloper includes features that are highly visual, and these features have equivalent functionality that is available to people who are blind or visually impaired:

- The UI and visual editors. The source editor provides equivalent functionality, as pages and UI elements can be completely designed and coded in the source editor.
- The Component Palette. The source editor provides equivalent functionality, as elements and tags that can be selected from the Component Palette can also be entered in the source editor.

You can add a component from the Component Palette to the UI or visual editor using keystrokes.

Oracle JDeveloper also includes modeling features. It is possible to create, edit and move elements on a diagram using only keystrokes.

3

Working with Oracle JDeveloper

This chapter is designed to get you up and running quickly on Oracle JDeveloper. Find information about working with the general development environment, source files, connections, using the online help, and common development tools.

This chapter includes the following sections:

3.1 About Working with Oracle JDeveloper

JDeveloper is the main development platform for the Oracle Fusion Middleware suite of products. It is a cross-platform IDE that runs on Windows, Linux, Mac OS X, and other UNIX-based systems.

3.2 Working with JDeveloper Roles

Roles enable you to tailor the JDeveloper environment. The modified environment removes items that you do not need from JDeveloper, including menus, preferences, New Gallery, and even individual fields on dialogs. The role you select determines which features and options are available to you as you work in JDeveloper.

The roles available are:

- **Default Role**. This role allows you to access all JDeveloper features. The other roles provide subsets of these features.

- **Customization** JDeveloper. This role allows you to create customizable applications, using the Oracle Metadata Services (MDS) framework.
- **Database Edition**. This gives you access to just the core database development tools.
- **Java EE Edition**. This includes only features for core Java EE development.
- **Java Edition**. This includes only features for core Java development.

> **Note:** The full set of online help is always available regardless of the role you have chosen for JDeveloper.

3.2.1 How to Change the JDeveloper Role

JDeveloper prompts you to select a role the first time it is run. You can also change the role while JDeveloper is running.

To change the JDeveloper role:

1. From the main menu, select **Tools > Switch Roles**.
2. The current role contains a bullet next to it. In the Switch Roles menu, select the role you want to switch to.

3.3 How to Manage JDeveloper Features

To optimize performance and user experience, JDeveloper allows you to just load the features you need for your project. Managing features enables you to see only those components of the IDE that are most relevant to your work. Managing features has no affect on the data in a project itself.

For example, assume two projects used to create two different views into an application. The first project might have Java features loaded, which informs JDeveloper that the IDE should reflect the Java technology stack. Such filtering eliminates clutter from individual projects. The second project might have a features loaded for Swing/AWT, informing JDeveloper to reflect IDE components required for Swing/AWT development.

To add or remove features in JDeveloper:

1. From the main menu, select Tools > Features. The Manage Features for *role* dialog opens. This dialog displays the features available in the current JDeveloper role. These features are checked by default.
2. Search for the feature you want to add or remove by entering it in the **Search** field, or scroll in the list of **Available Features**. Click a feature or feature category and view its description on the right.
3. Check the features you want to add, and uncheck the features you want to remove. Click the Check for Updates icon to open the Check For Updates wizard which allows you to load features from an extension.
4. Optionally, to clear previously loaded features from the cache, click **Clear Feature Loading Cache**.
5. Click OK when you are done.

3.4 Working With Windows In the IDE

JDeveloper allows you to arrange the windows according to your convenience. JDeveloper uses two kinds of windows in the IDE:

- Dockable windows that can be placed anywhere in the IDE.
- Tabbed editor windows that are fixed in the center of the IDE.

3.4.1 How to Maximize Windows

Double-click the title bar of any JDeveloper window to quickly maximize to full screen view. Double-click the title bar again to return the window to its former position in the IDE.

3.4.2 How to Minimize and Restore Dockable Windows in the IDE

You can minimize any dockable window in JDeveloper, or set it to remain open in place. The default state is set to remain open.

When a window is set to stay open, its position is static. It remains always visible, in whichever position you have docked it.

When a window is set to minimize, its behavior is more fluid. When you give it focus, it opens fully in the general area (top, bottom, left, right) where you last left it docked. When you move the focus elsewhere, the minimized window collapses into the margin. Whether open or closed, any minimized window's status set to minimize is identified by a named button in the margin.

To minimize any dockable window:

- Click the **Minimize** icon in the far right-hand corner of the window set to be kept open.

  If the window currently has focus, it now expands to full height and remains in place. If the window does not have focus, it collapses into the margin.

  When you minimize a window, a button bearing that window's name appears in the margin. You can toggle the minimized window open and closed with this button.

> **Note:** When you minimize a window that exists in a docking zone that also contains other windows, all windows in the docking zone are minimized.

3.4.3 How to Dock Windows in the IDE

All of the tools available under the View menu—the Application Navigator, Structure window, Property Inspector, and so on—can be arranged however you like. You can dock them singly or in groups. You can also tab windows together in one location, either as docked or floating windows.

The following table provides information on how to move dockable windows.

Requirement	Action
Move a solitary docked window	Grab its title bar and drag
Decouple a docked window from a group	Grab its title bar and drag

Requirement	Action
Move a group of docked, tabbed, or docked and tabbed windows	Grab the title bar for the group—the topmost horizontal title bar, empty but for the close box—and drag.
To decouple one tabbed window from a group	Grab the window's tab and drag.

Note: The title bars for docked windows sometimes appear vertically, on the side of the window.

The following table provides information on ways to reposition dock windows:

Requirement	Action
Dock a window (or window group) against another edge of the development area	Drag the window (or window group) to the destination edge
Dock a window (or window group) alongside another window	Drag the window (or window group) to the top, bottom, or side edge of the docked window
Tab one window with another	Drag the window to be tabbed into the center of the destination window (or window group) and release

3.4.4 About Dockable Windows in the IDE

You can float any window that's normally docked—the Application Navigator, any custom navigator, the Log window, the Property Inspector, the Component Palette. You can also resize and position it wherever you would like within JDeveloper.

Generally, floating windows are best suited for a large screen with enough room for displaying both the information windows and your source code. If you are using floating palettes on a smaller screen they can sometimes be hidden by other information windows as you work.

3.4.5 How to Close and Reopen Dockable Windows in the IDE

You can easily open and close the main elements of the JDeveloper IDE, which include the navigators, Structure window, Property Inspector, Component Palette, Resource Palette and Log window.

To open a closed window:

- In the **View** menu, choose the name of the window.

You can close an open window in one of these ways:

- Click the **Close** icon which appears on the tab window's name.
- With the focus in the window, press Shift+Escape or Ctrl+Click.

3.4.6 How to Restore Window Layout to Factory Settings

To restore the layout of dockable windows in JDeveloper, go to the **Window** menu and select **Reset Layout to Factory Settings**.

3.5 Navigating The IDE

You can accomplish any task in JDeveloper using the keyboard as you use the mouse.

3.5.1 How to Work With Shortcut Keys In The IDE

JDeveloper comes with several predefined keyboard schemes. You can choose to use one of these, or customize an existing set to suit your own coding style by changing which keyboard shortcuts map to which actions.

To load preset keyboard schemes:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the preferences dialog, select the **Shortcut Keys** node. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
3. On the shortcut keys page, click **More Actions** and then select **Load Keyboard Scheme**. The Load Keyboard Scheme dialog appears, with the currently loaded keyboard scheme highlighted.
4. In the Load Keyboard Scheme dialog, select the scheme you wish to load and click **Ok**.
5. On the Shortcut Keys page, if you have finished, click **Ok**.

To view JDeveloper commands and their associated keyboard shortcuts (if assigned):

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, select the **Shortcut Keys** node.
3. On the Shortcut Keys page, under **Available Commands**, you can view the complete set of JDeveloper commands, and what keyboards shortcuts (if any) are assigned to each. If you are looking for a particular command or shortcut, or want to look at shortcuts for a particular category of commands only, enter a filtering expression in the **Search** field.
4. You can also define new shortcuts, or change existing ones.

To define a new keyboard shortcut for a command within a given keyboard scheme:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, select the **Shortcut Keys** node. For more information at any time, press F1 or click **Help** from within the preferences dialog.
3. On the Shortcut Keys page, under **Available Commands**, select the command that you wish to define a new shortcut for.
4. To define a new shortcut for this action, place focus on the **New Shortcut** field, and then press the key combination on the keyboard.

   If this proposed shortcut already has a command associated with it, that command will now appear in the **Conflicts** field. Any conflicting shortcuts are overwritten when a new shortcut is assigned.
5. To assign this shortcut to the action selected, click **Assign**. If you want to delete an already-assigned shortcut, click the **Delete** button in the toolbar.

If you want to assign more than one shortcut to a command, select the command and click the **Duplicate** button. Then, type the shortcut key in the **New Shortcut** field and click **Assign**.

6. When you are finished, click **Ok**.

To import or export keyboard schemes:

1. From the main menu, select **Tools > Preferences** to open the Preferences dialog.
2. Click **More Actions > Export** or **Import**. Keyboard schemes are stored as XML files.

3.5.2 Keyboard Navigation In JDeveloper

For any action that can be accomplished with a mouse, including selection, there is a way to accomplish the action solely from the keyboard. You can accomplish any task in JDeveloper using the keyboard as you can using the mouse.

The shortcut keys defined in the Java Look and Feel guidelines provide the base set for JDeveloper. The various predefined keyboard schemes available in JDeveloper are then overlaid upon this base set. If the same shortcut key exists in both the look and feel guidelines and the JDeveloper keyboard scheme, the JDeveloper scheme prevails. If a shortcut key defined by the look and feel guidelines does not appear in a JDeveloper scheme, then it is the original look and feel definition that remains in effect when the scheme in question is enabled.

At any given time, then, the shortcut keys enabled in JDeveloper depend upon the interaction of the currently enabled scheme with the Java look and feel guidelines. When you first open JDeveloper, the default scheme is enabled. You can change this scheme whenever you wish, and within each scheme, you can customize any of the shortcut key assignments that you would like. Note that any customized shortcuts you create in a scheme are not retained when another predefined keyboard scheme is activated (or even if the same scheme is reloaded).

To load predefined keyboard schemes, view current shortcut assignments within a scheme, and customize those assignments, you will need to open the preferences dialog. To open the dialog, choose **Tools** > **Preferences** (or on the keyboard, press Alt+T+P) from the main menu and then, using the arrow keys in the left-hand pane, navigate to the **Shortcut Keys** node. For details on working with the dialog, with the page displayed, click **Help** (or on the keyboard press H).

3.5.2.1 Common Navigation Keys

The following table describes the common methods of moving the cursor in JDeveloper:

Table 3–1 Common Methods of Moving the Cursor

Key	Cursor Movement	Ctrl+cursor Movement
Left Arrow	Left one unit (e.g., a single character)	Left one proportionally larger unit (e.g., a whole word)
Right Arrow	Right one unit	Right one proportionally larger unit
Up Arrow	Up one unit or line	Up one proportionally larger unit
Down Arrow	Down one unit or line	Down one proportionally larger unit

Table 3–1 (Cont.) Common Methods of Moving the Cursor

Key	Cursor Movement	Ctrl+cursor Movement
Home	Beginning of the line	To the beginning of the data (top-most position)
End	End of the line	To the end of the data (bottom-most position)
Tab	Next field or control, except when in a text area or field. In this case, press Ctrl+Tab to navigate out of the control. Where there are fields and controls ordered horizontally as well as vertically, pressing Tab moves the cursor first horizontally to the right, then at the end of the line, down to the left of the next line.	To the next pane which may be a navigator, an editor, or a palette, except when in a text area or field. In this case, press Ctrl+Tab to navigate out of the control
Shift+Tab	Previous field	To previous tab position. In property sheets, this moves the cursor to the next page
Enter	Selects and highlights the default button, except when in a combo box, shuttle button, or similar control. **Note**: The default button changes as you navigate through controls.	n/a

3.5.2.2 Navigation In Standard Components

This section describes keyboard navigation in standard JDeveloper components.

Buttons

The following table describes the keyboard actions to perform navigation tasks involving buttons.

Table 3–2 Keyboard Navigation for Buttons

Navigation	Keys
Navigate forward to or from button	Tab
Navigate backward to or from button	Shift+Tab
Activate the default button (when the focus is not on a button)	Enter
Activate any button while it has focus	Enter, Spacebar, or keyboard shortcut (if one has been defined)
Activate **Cancel** or **Close** buttons on a dialog	Esc

Checkboxes

The following table describes the keyboard actions to perform navigation tasks involving checkboxes.

Table 3–3 Keyboard Navigation for Checkboxes

Navigation	Keys
Navigate forward to or from checkbox	Tab

Table 3–3 (Cont.) Keyboard Navigation for Checkboxes

Navigation	Keys
Navigate backward to or from checkbox	Shift+Tab
Select or deselect (when the focus is on the checkbox)	Spacebar or keyboard shortcut (if one has been defined)
Navigate to checkbox and select or deselect (when the focus is not on the checkbox)	Keyboard shortcut (if one has been defined)

Dropdown Lists And Combo Boxes

The following table describes the keyboard actions to perform navigation tasks involving dropdown lists and combo boxes.

Table 3–4 Keyboard Navigation for Dropdown Lists and Combo Boxes

Navigation	Keys
Navigate forward to or from a combo box or dropdown list	Tab or keyboard shortcut (if one has been defined)
Navigate backward to or from a combo box or dropdown list	Shift+Tab
Toggle list open and closed	Spacebar (the current selection receives the focus)
Open a list	Down Arrow to open (first item on list receives focus)
Move up or down within list	Up and Down Arrow keys (highlighted value has focus)
Move right and left within the initial entry on a combo box	Right and Left Arrow keys
Select list item	Enter **Note**: The first time you press Enter, the item in the list is selected. The second time you press Enter, the default button is activated.
Close list (with the highlighted value selected)	Esc

List Boxes

The following table describes the keyboard actions to perform navigation tasks involving list boxes.

Table 3–5 Keyboard Navigation for List Boxes

Navigation	Keys
Navigate forward into or out of a list	Tab
Navigate backward into or out of list	Shift+Tab

Table 3–5 (Cont.) Keyboard Navigation for List Boxes

Navigation	Keys
Make a selection	Up Arrow, Down Arrow, Spacebar, or Enter **Note**: The first time you press Enter, the highlighted item in the list is selected. The second time you press Enter, the default button is activated.
Move within list	Up Arrow or Down Arrow
Move to beginning of list	Home or Ctrl+Home
Move to end of list	End or Ctrl+End
Select all entries	Ctrl+A
Toggle (select or deselect) an item	Spacebar or Ctrl+Spacebar
Select next item up in list without deselecting item with current focus	Shift+Up Arrow Key
Select next item down in list without deselecting item with current focus	Shift+Down Arrow Key
Select current item and all items up to the top of the list	Shift+Home
Select current item and all items up to the bottom of the list	Shift+End
Select current item and all items visible above that item	Shift+Page Up
Select current item and all items visible below that item	Shift+Page Down
Select item with current focus without deselecting other items (to select items that are not adjacent)	Ctrl+Spacebar
Navigate through list without deselecting item with current focus.	Ctrl+Up Arrow or Ctrl+Down Arrow

Radio Buttons

Table 3–6 Keyboard Navigation for Radio Buttons

Navigation	Keys
Navigate forward to or from radio button	Tab
Navigate backward to or from radio button	Shift+Tab
Navigate forward from radio button	Arrow Keys
Navigate backward from radio button	Shift+Arrow Keys

Table 3–6 (Cont.) Keyboard Navigation for Radio Buttons

Navigation	Keys
Select radio button	Arrow key (navigating to a radio button via arrows selects it) or keyboard shortcut (if one has been defined)
Deselect radio button	Select a different radio button in the group using one of the commands above

Shuttles

The following table describes the keyboard actions to perform navigation tasks involving shuttles.

Table 3–7 Keyboard Navigation for Shuttles

Navigation	Keys
Navigate forward into or out of a list	Tab
Navigate backward into or out of list	Shift+Tab
Make a selection	Up Arrow or Down Arrow
Move within list	Up Arrow or Down Arrow
Move to beginning of list	Home or Ctrl+Home
Move to end of list	End or Ctrl+End
Select all entries	Ctrl+A
Toggle (select or deselect) an item	Spacebar or Ctrl+Spacebar
Select next item up in list without deselecting item with current focus	Select next item up in list without deselecting item with current focus
Select next item down in list without deselecting item with focus	Shift+Down Arrow Key
Select current item and all items up to the top of the list	Shift+Home
Select current item and all items up to the bottom of the list	Shift+End
Select current item and all items visible above that item	Shift+Page Up
Select current item and all items visible below that item	Shift+Page Down
Select item with current focus without deselecting other items (to select items that are not adjacent)	Ctrl+Spacebar
Navigate through list without deselecting item with current focus.	Ctrl+Up Arrow or Ctrl+Down Arrow

Sliders

The following table describes the keyboard actions to perform navigation tasks involving sliders.

Table 3–8 Keyboard Navigation for Sliders

Navigation	Keys
Navigate forward to or from slider	Tab
Navigate backward to or from slider	Shift+Tab
Increase value	Up Arrow or Right Arrow
Decrease value	Left Arrow or Down Arrow
Minimum value	Home
Maximum value	End

Spin Controls

The following table describes the keyboard actions to perform navigation tasks involving spin controls.

Table 3–9 Keyboard Navigation for Spin Controls

Navigation	Keys
Navigate forward to or from spin control	Tab
Navigate backward to or from spin control	Shift+Tab
Increase value	Up Arrow or Right Arrow, or type the value you want
Decrease value	Left Arrow or Down Arrow, or type the value you want
Minimum value	Home
Maximum value	End

Text Fields

The following table describes the keyboard actions to perform navigation tasks involving text fields.

Table 3–10 Keyboard Navigation for Text Fields

Navigation	Keys
Navigate forward into or out of text box	Tab or keyboard shortcut (if one has been defined)
Navigate backward into or out of text box	Shift+Tab
Move to previous/next character within text box	Left Arrow/Right Arrow
Move to start/end of box	Home/End
Select all text	Ctrl+A
Deselect all text	Left Arrow or Right Arrow

Table 3–10 (Cont.) Keyboard Navigation for Text Fields

Navigation	Keys
Select current item and all items up to the Left/Right	Shift+Left Arrow, Shift+Right Arrow
Select current item and all items up to the Start/End	Shift+Home, Shift+End
Select current item and all items up to the previous/next word	Ctrl+Shift+Left Arrow, Ctrl+Shift+Right Arrow
Copy selection	Ctrl+C
Cut selection	Ctrl+X
Paste from clipboard	Ctrl+V
Delete next character	Delete
Delete previous character	Backspace

3.5.2.3 Navigating Complex Controls

This section contains information about keyboard shortcuts for complex UI components.

Dockable Windows

The following table describes the keyboard actions to perform navigation tasks involving dockable windows.

Table 3–11 Keyboard Navigation for Dockable Windows

Navigation	Keys
Navigate forward in or out of dockable window	Ctrl+Tab
Navigate backward in or out of dockable window	Ctrl+Shift+Tab
Display context menu	Shift+F10
Navigate between tabs within a dockable window	Alt+Page Down, Alt+Page Up
Move between elements including dropdown lists, search fields, panels, tree structure (but not individual elements in a tree), individual component buttons	Tab
Move up/down through dockable window contents (scrollbar)	Up Arrow, Down Arrow This scrolls the window contents if the focus moves beyond visible area of canvas.
Move left/right (scrollbar)	Up Arrow, Down Arrow This scrolls the pane contents if focus moves beyond visible area of canvas.
Move to start/end of data (component buttons)	Ctrl+Home, Ctrl+End
Select an element	Enter or Spacebar

Table 3–11 (Cont.) Keyboard Navigation for Dockable Windows

Navigation	Keys
Scroll left/right within the canvas area (without moving through the window contents)	Ctrl+Left/Ctrl+Right
Scroll Up/Down within the canvas area (without moving through the window contents)	Ctrl+Up/Ctrl+Down

Menus

Context menus are accessed using Shift+F10. Menus from the main menu bar are accessed using the keyboard shortcut for the menu.

The following table describes the keyboard actions to perform navigation tasks involving the menu bar.

Table 3–12 Keyboard Navigation for Menus

Navigation	Keys
Navigate to menu bar	F10
Navigate out of menu bar	Esc
Navigate between menus in menu bar	Right Arrow, Left Arrow
Navigate to menu item	Up Arrow, Down Arrow
Navigate from menu item	Up Arrow, Down Arrow
Activate item	Enter, Spacebar, or keyboard shortcut (if one has been defined)
Open submenu	Right Arrow
Retract submenu	Left Arrow or Esc

Panels

The following table describes the keyboard actions to perform navigation tasks involving panels.

Table 3–13 Keyboard Navigation for Panels

Navigation	Keys
Navigate in/out forward	Tab
Navigate in/out backward	Shift+Tab
Expand panel (when focus on header)	Right Arrow
Collapse panel (when focus on header)	Left Arrow
Navigate within panel	Up Arrow, Down Arrow
Navigate to panel header from contents (when focus is on top item in list)	Up Arrow

Table 3–13 (Cont.) Keyboard Navigation for Panels

Navigation	Keys
Navigate to panel contents from header (when focus is on header)	Down Arrow

Tables

Arrow keys move focus in the direction of the arrow, except when a web widget has focus; in that case, the down arrow or enter key initiates the widget control action, such as opening a choice list. tab moves the focus right, shift+tab moves the focus left.

The following table describes the keyboard actions to perform navigation tasks involving tables.

Table 3–14 Keyboard Navigation for Tables

Navigation	Keys
Navigate forward in or out of table	Ctrl+Tab
Navigate backward in or out of table	Shift+Ctrl+Tab
Move to next cell (wrap to next row if in last cell)	Tab Arrow or Right Arrow
Move to previous cell (wrap to previous row if in first cell)	Shift+Tab or Left Arrow
Controls in cells open	Down Arrow or Enter
Block move left	Ctrl+Page Up
Block move right	Ctrl+Page Down
Block move up	Page Up
Block move down	Page Down
Move to first cell in row	Home
Move to last cell in row	End
Move to first cell in table	Ctrl+Home
Move to last cell in table	Ctrl+End
Select all cells	Ctrl+A
Deselect current selection (and select alternative)	Any navigation key
Extend selection on row	Shift+Up Arrow
Extend selection one column	Shift+Down Arrow
Extend selection to beginning of row	Shift+Home
Extend selection to end of row	Shift+End
Extend selection to beginning of column	Ctrl+Shift+Home

Table 3–14 (Cont.) Keyboard Navigation for Tables

Navigation	Keys
Extend selection to end of column	Ctrl+Shift+End
Edit cell without overriding current contents, or show dropdown list in combo box	F2
Reset cell content prior to editing	Esc

Tabs

This section refers to the tabs that appear within a dockable window, view or dialog. The following table describes the keyboard actions to perform navigation tasks involving tabs in dockable windows, views and dialogs.

Table 3–15 Keyboard Navigation for Tabs

Navigation	Keys
Navigate forward into or out of tab control	Tab
Navigate backward into or out of tab control	Ctrl+Tab
Move to tab (within control) left/right	Left Arrow/Right Arrow
Move to tab (within control) above/below	Up Arrow/Down Arrow
Move from tab to page	Ctrl+Down
Move from page to tab	Ctrl+Up
Move from page to previous page (while focus is within page)	Ctrl+Page Up
Move from page to next page (while focus is within page)	Ctrl+Page Down

Trees

The following table describes the keyboard actions to perform navigation tasks involving trees.

Table 3–16 Table Navigation for Trees

Navigation	Keys
Navigate forward into or out of tree control	Tab
Navigate backward into or out of tree control	Shift+Tab
Expand (if item contains children)	Right Arrow
Collapse (if item contains children)	Left Arrow

Table 3–16 (Cont.) Table Navigation for Trees

Navigation	Keys
Move to parent from child (if expanded)	Left Arrow
Move to child from parent (if already expanded)	Right Arrow
Move up/down one item	Up Arrow, Down Arrow
Move to first item	Home
Move to last entry	End
Select all children of selected parent	Ctrl+A
Select next item down in list without deselecting that item that currently has focus	Shift+Down Arrow
Select next item up in list without deselecting that item that currently has focus	Shift+Up Arrow
Select current item and all items up to the top of the list	Shift+Home
Select current item and all items up to the bottom of the list	Shift+End
Select the item with current focus without deselecting other items (to select items that are not adjacent)	Ctrl+Spacebar
Navigate through list without deselecting item with current focus	Ctrl+Up/Down Arrow

Wizards

The Following Table Describes The Keyboard Actions To Perform Navigation Tasks Involving Wizards.

Table 3–17 Keyboard Navigation for Wizards

Navigation	Keys
Navigate between stops on the roadmap or between pages	Up Arrow, Down Arrow (these do not wrap)
Navigate forward between components on wizard panel, wizard navigation bar buttons, and navigation panel	Tab
Navigate backward between components on wizard panel, wizard navigation bar buttons, and navigation panel	Shift+Tab

Table 3–17 (Cont.) Keyboard Navigation for Wizards

Navigation	Keys
Navigate between buttons on Navigation Bar	Right and Left Arrow Key (does not wrap)
Navigate between stops on Roadmap/between wizard pages	Ctrl Page Up and Ctrl Page Down

3.5.2.4 Navigation in Specific Components

This section contains information about keyboard shortcuts for JDeveloper-specific UI components.

Dialogs

The following table describes the keyboard actions to perform navigation tasks involving dialogs.

Table 3–18 Keyboard Navigation for Dialogs

Navigation	Keys
Close dialog without making any selections or changes	Esc
Activate the default button (if one is defined)	Enter

Overview Editor (Form + Mapping)

The following table describes the keyboard actions to perform navigation tasks involving overview editors.

Table 3–19 Keyboard Navigation for the Overview Editor

Navigation	Keys
Navigate into or out of overview editor from other pages in editor (for example Source or History)	Alt+Tab
Navigate from the tab group to next control in editor)	Tab or Ctrl+Down Arrow
Navigate forward or backwards between controls on overview editor	Tab or Alt+Tab
Move between tabs in the side tab control (when the focus in the tab group)	Up Arrow, Down Arrow
Move between tabs in side tab control (when focus on Page)	Ctrl+Page Up/Ctrl+Page Down
Move from page to tab group (from next control in editor)	Ctrl+Tab
Move from page to tab group (from any control in editor)	Ctrl+Up Arrow

Table 3–19 (Cont.) Keyboard Navigation for the Overview Editor

Navigation	Keys
Open and close Sections (when focus is on a section header)	Enter, Spacebar, Right Arrow/Left Arrow

Component and Resource Palettes

The following table describes the keyboard actions to perform navigation tasks involving palettes.

Table 3–20 Keyboard Navigation for Component and Resource Palettes

Navigation	Keys
Navigate forward in or out of palette	Ctrl+Tab This moves you into first item within the pane.
Navigate backward in or out of palette	Ctrl+Shift+Tab
Move between elements including dropdown lists, search fields, panels, tree structure (but not individual elements in a tree), individual component buttons	Tab, Shift+Tab
Move up/down elements in a list or tree	Up Arrow/Down Arrow
Move left/right elements in a list or tree	Left Arrow/Right Arrow
Move to start/end of data (component buttons)	Ctrl+Home/Ctrl+End
Select a component button	Enter

Navigators

The following table describes the keyboard actions to perform navigation tasks involving navigators.

Table 3–21 Keyboard Navigation for Navigators

Navigation	Keys
Navigate forward in or out of navigator	Ctrl+Tab This moves you into first item within the pane.
Navigate backward in or out of navigator	Ctrl+Shift+Tab
Move between elements including dropdown lists, search fields, panels, tree structure (but not individual elements in a tree), individual component buttons	Tab
Move up/down elements in a list or tree	Up Arrow/Down

Table 3–21 (Cont.) Keyboard Navigation for Navigators

Navigation	Keys
Move left/right elements in a list or tree	Left Arrow/Right Arrow
Move to start/end of data (component buttons)	Ctrl+Home/Ctrl+End
Select a component button	Enter
Select an element	Enter

Property Inspector

The following table describes the keyboard actions to perform navigation tasks involving the Property Inspector.

Table 3–22 Keyboard Navigation for the Property Inspector

Navigation	Keys
Navigate forward into or out of Property Inspector	Ctrl+Tab
Navigate backward into or out of Property Inspector	Ctrl+Shift+Tab
Navigate from side tab group to page	Tab
Navigate backward and forwards between elements on page	Tab, Shift+Tab
Move to tab above/below (when focus is on the side tab)	Up Arrow, Down Arrow
Move to tab right or left, above or below (when focus is on the internal tab group)	Up Arrow, Down Arrow, Right Arrow, Left Arrow
Move from side tab group to page	Ctrl+Down Arrow
Move from page to side tab group	Ctrl+Up Arrow
Move to side tab above (previous) when focus on page	Ctrl+Page Up
Move to side tab below (next) when focus on page	Move to side tab below (next) when focus on page
Open and Close sections (when focus is on a section header)	Enter

Text Editors

The following table describes the keyboard actions to perform navigation tasks involving the pane elements of text editors.

Table 3–23 Keyboard Navigation for Text Editors

Navigation	Keys
Navigate forward in or out of editor	Ctrl+Tab
Navigate backward in or out of editor	Ctrl+Shift+Tab
Move from page to previous page	Alt+Page Up
Move from page to next page	Alt+Page Down

The following table describes the keyboard actions to perform navigation tasks involving the text or canvas areas of text editors.

Table 3–24 Keyboard Navigation for Canvas Areas of Text Editors

Navigation	Keys
Move up/down one line	Up Arrow, Down Arrow
Move left/right one character	Left Arrow, Right Arrow
Move to start/end of line	Home, End
Move to previous/next word	Ctrl+Left Arrow, Ctrl+Right Arrow
Move to start/end of text area	Ctrl+Home/Ctrl+End
Move to beginning/end of data	Ctrl+Home/Ctrl+End
Move up/down one vertical block	Page Up/Page Down
Block move left	Ctrl+Page Up
Block move right	Ctrl+Page Down
Block extend up	Shift+Page Up
Block extend down	Shift+Page Down
Block extend left	Ctrl+Shift+Page Up
Block extend right	Ctrl+Shift+Page Down
Select all	Ctrl+A
Deselect all	Up Arrow, Down Arrow, Left Arrow, Right Arrow
Extend selection up/down one line	Shift+Up Arrow/Shift+Down Arrow
Extend selection left/right one component or char	Shift+Left Arrow/Shift+Right Arrow
Extend selection to start/end of line	Shift+Home/Shift+End
Extend selection to start/end of data	Ctrl+Shift+Home/Ctrl+Shift+End
Extend selection up/down one vertical block	Shift+Page Up/Shift+Page Down

Table 3–24 (Cont.) Keyboard Navigation for Canvas Areas of Text Editors

Navigation	Keys
Extend selection to previous/next word	Ctrl+Shift+Left Arrow /Ctrl+Shift+Right Arrow
Extend selection left/right one block	Ctrl+Shift+Page Up/Ctrl+Shift+Page Down
Copy selection	Ctrl-C
Cut selection	Ctrl-X
Paste selected text	Ctrl-V

Graphical Editors

The following table describes the keyboard actions to perform navigation tasks involving graphical editors.

Table 3–25 Keyboard Navigation for Graphical Editors

Navigation	Keys
Navigate forward in or out of editor	Ctrl-Tab
Navigate backward in or out of editor	Ctrl+Shift+Tab
Move from page to previous page	Alt+Page Up
Move from page to next page	Alt+Page Down

The following table describes the keyboard actions to perform navigation tasks involving the canvas areas of graphical editors.

Table 3–26 Keyboard Navigation for Canvas Areas of Graphical Editors

Navigation	Keys
Move to the next focusable element within editor area	Up Arrow, Down Arrow, Left Arrow, Right Arrow
Select element	Spacebar
Activate context menu	Shift+F10

3.6 Customizing the IDE

You can alter the appearance and functionality of a wide variety of JDeveloper features.

3.6.1 How to Change the Look and Feel of the IDE

You can alter the appearance of JDeveloper using pre-defined settings.

To change the look and feel of the IDE:

1. From the main menu, choose **Tools** > **Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Environment** node if it is not already selected.

3. On the Environment page, select a different look and feel from the **Look and Feel** dropdown list.
4. Click **OK**.
5. Restart JDeveloper.

> **Note:** The key bindings in Motif are different from key bindings in Windows. Under Motif, the arrow keys do not change the selection. Instead they change the lead focus cell. You must press Ctrl + Space to select an item. This is expected behavior.

3.6.2 How to Customize the General Environment for the IDE

You can customize the default display options (such as whether or not the splash screen is displayed at start up, or whether dockable windows are always on top), as well as other general behavior, such as whether JDeveloper will automatically reload externally modified files and whether output to the Log window is automatically saved to a file.

To change the general environment settings for the IDE:

1. From the main menu, choose **Tools** > **Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Environment** node if it is not already selected.
3. On the Environment page, select the options and set the fields as appropriate.
4. Click **OK**.
5. Restart JDeveloper.

3.6.3 How to Customize Dockable Windows in the IDE

You can customize the layout for dockable windows in their docked position. You can also set dockable windows to remain on top of other GUI elements, or not, when those windows are moved.

To change the shape of one or more of the four docking areas:

1. From the main menu, choose **Tools** > **Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Environment** node select **Dockable Windows**.
3. On the Dockable Windows page, click the corner arrows to lengthen or shorten each docking area's shape.
4. Click **OK**.

To change whether dockable windows remain on top or not when moved:

1. From the main menu, choose **Tools** > **Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Environment** node select **Dockable Windows**.
3. On the Dockable Windows page, select or deselect **Dockable Windows Always on Top** as appropriate.
4. Click **OK**.

3.6.4 How to Customize the Compare Window in the IDE

You can customize the display of the Compare window.

To customize the options for comparing files:

1. From the main menu, choose **Tools** > **Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select **Compare and Merge**.
3. On the Compare page, set the options available for the display of two files being compared.
4. Click **OK**.

3.6.5 How to Customize the Component Palette

The Component Palette offers you a quick method for inserting components into files open in the editor.

3.6.5.1 How to Add a Page to the Palette

You can add pages to the Component Palette, within which to group additional components, or you can add components to existing pages.

To add a page to the Palette:

1. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog. For more information at any time, press F1 or click **Help** from within the Configure Component Palette dialog.
2. Optionally, in the Configure Component Palette dialog, for **Page Type** select the appropriate type to limit the display in the **Pages** list.
3. In the Configure Component Palette dialog, underneath the **Pages** list box, click **Add**.
4. In the Create Palette Page dialog, enter the name of the new page and select a type from the dropdown list. If you selected a page type in Step 2, that type is reflected now in this dialog.
5. Click OK to return to the Configure Component Palette dialog.
6. If finished, click **OK**. The new page is now added to the dropdown list in the Component Palette. It also appears in the Pages list of the Configure Component Palette dialog.

Alternately, right-click in the Component Palette and choose **Add Page**.

3.6.5.2 How to Add a JavaBeans Component to the Palette

You can add pages to the Component Palette to group your JavaBeans components, or you can add components to existing pages. Once you add JavaBeans to the Palette, you can insert these beans into any file you have open in the Java Visual Editor by selecting them from the Palette.

To add a JavaBeans component to the Palette:

1. If the bean is not already referenced by a library, create a user library (outside the project) for the bean.

In the **Class Path** field, set the location of the bean class. If the bean is in an archive, use the archive. If the bean is contained in a project, use the output directory of that project.

Note that when you are creating your own JavaBeans for later deployment, it can be useful to defer putting them into an archive until you have finished development.

2. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog. For more information at any time, press F1 or click **Help** from within the Configure Component Palette dialog.
3. Optionally, in the Configure Component Palette dialog, for **Page Type** select **Java** to view only those pages containing JavaBeans.

   Skip to Step 6 if you do not want to add a new page.
4. Underneath the **Pages** list box, click **Add**.
5. In the Create Palette Page dialog, enter the name of the new page, ensure that **Java** is selected from the dropdown list, and click **OK**.

   Your new page name is added to the bottom of the **Pages** list in the Configure Component Palette dialog.
6. In the **Pages** list, select the page to which you wish to add the JavaBeans component.
7. Underneath the **Components** list box, click **Add**.
8. In the Add JavaBeans dialog, fill in the appropriate details for the new component.
9. Click **OK** to return to the Configure Component Palette dialog.
10. If finished, click **OK**.

The new beans component now appears in the Component Palette when the appropriate page is selected. It also appears in the **Components** list of the Configure Component Palette dialog when the page it is associated with is selected in the **Pages** list.

3.6.5.3 How to Add a Code Snippet to the Palette

You can add pages to the Component Palette to group your snippets, or you can add snippets to the existing Code Snippets page. Once you add snippets to the Palette, you can insert this code into any file you have open in the source editor by selecting the snippet from the Palette.

To add a Code Snippet to the Palette:

1. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog. For more information at any time, press F1 or click **Help** from within the Configure Component Palette dialog.
2. In the Configure Component Palette dialog, for Page Type select snippet to view only those pages containing snippets.

   Skip to step 5 if you do not want to add a new page.
3. Underneath the **Pages** list box, click **Add**.
4. In the Create Palette Page dialog, enter the name of the new page, ensure that snippet is selected from the dropdown list, and click **OK**.

Your new page name is added to the bottom of the Pages list in the Configure Component Palette dialog.

5. In the **Pages** list, select the predefined **Code Snippets** page, or any of your own snippets pages.
6. Underneath the **Components** list box, click **Add**.
7. In the Add Snippets dialog, enter a name for the snippet and then the code itself.
8. Click **OK** to return to the Configure Component Palette dialog.
9. If finished, click **OK**.

The new snippet now appears in the Component Palette when the appropriate page is selected. It also appears in the **Components** list of the Configure Component Palette dialog when the page it is associated with is selected in the **Pages** list

3.6.5.4 How to Remove a Page from the Palette

Note that if you remove a page supplied by JDeveloper, the only way to recover it again is to restore the default setting for the Component Palette by moving the `palette.xml` file from *`jdev_install`*`/system` to *`jdev_install`*`/system/release_and_build_`number, where *`jdev_install`* is the root directory in which JDeveloper is installed.

To remove a page from the Palette:

1. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog. For more information at any time, press F1 or click **Help** from within the Configure Component Palette dialog.
2. Optionally, in the Configure Component Palette dialog, for **Page Type** select the appropriate type to limit the display in the **Pages** list.
3. In the **Pages** list, select the page to be removed.
4. Underneath the **Pages** list box, click **Remove**.

   If the page cannot be removed, the Illegal Request dialog appears.
5. To confirm removal, in the Confirm Remove Page dialog, click **Yes**.
6. In the Configure Component Palette dialog, click **OK**.

   The page no longer appears in the Component Palette dropdown list. It has also been removed from the **Pages** list of the Configure Component Palette dialog.

Alternately, with the page selected in the Component Palette, right-click in the Palette and choose **Remove Page**.

3.6.5.5 How to Remove a Component from the Palette

Note that if you remove a component supplied by JDeveloper, the only way to recover it again is to restore the default setting for the Component Palette by moving the `palette.xml` file from *`jdev_install`*`/system` to *`jdev_install`*`/system/release_and_build_`number, where *`jdev_install`* is the root directory in which JDeveloper is installed.

To remove a component from the Palette:

1. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog. For more information at any time, press F1 or click **Help** from within the Configure Component Palette dialog.

2. Optionally, in the Configure Component Palette dialog, for **Page Type** select the appropriate type to limit the display in the **Pages** list.
3. In the **Pages** list, select the page you want to remove the component from.
4. In the **Components** list box, click **Remove**.

   If the component cannot be removed, the Illegal Request dialog appears.
5. To confirm removal, in the confirmation dialog, click **Yes**.
6. In the Configure Component Palette dialog, click **OK**.

   The component no longer appears in the Component Palette dropdown list. It has also been removed from the **Components** list of the Configure Component Palette dialog.

You cannot remove a component using the Component Palette context menu. You must work through the Configure Component Palette dialog.

3.6.6 How to Change Roles in JDeveloper

You can change the roles that are used to shape JDeveloper. Shaping tailors the JDeveloper environment based on the role of the user.

When you change to a new role, it is only available after you restart JDeveloper.

To change the role for JDeveloper:

- From the main menu, choose **Tools** > **Switch Roles** and select the role of your choice.

3.6.7 How to Associate File Types with JDeveloper

You can associate commonly used file types with JDeveloper. Once a file type has been associated with JDeveloper, opening a file of that type automatically launches JDeveloper. (This feature is supported only in Windows systems.)

To associate a file type with JDeveloper:

1. From the main menu, choose **Tools** > **Preferences** and open the **File Types** pane. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the list of file types, select a file type to be associated with JDeveloper.
3. In the **Details for** area, check **Open with** JDeveloper.

3.7 Working with the Resource Palette

When designing and building applications, you may need to find and use many software assets. You may know what you want to find, but you may not certain where to find it or even what the artifact of interest is called. Even if you think you know where to find the artifact, and what it is called, you might not know how to establish a connection to the source repository. Consider the following:

- An application developer needs to find and incorporate shared model, view and controller objects created by other members of her team, and by other product teams.

- A UI designer needs access to a corporate catalog of images, style sheets, templates and sample designs to facilitate rapid creation of standards-compliant pages.
- An application integrator needs easy access to a variety of web services of interest to a particular domain.
- An end user needs to find relevant content (for example, portlets and UI components) for use while personalizing a page.
- In each of these cases, the user has a simple goal: find the resource(s) needed for the task at hand. The process of discovering and accessing the assets should be as effortless as possible.

The Resource Palette in JDeveloper addresses this. It lets you:

- Locate resources stored in a wide variety of underlying repositories through IDE connections
- Locate resources by browsing a hierarchical structure in catalogs
- Search for resources and save searches
- Filter resources to reduce the visible set when browsing
- Use a resource you have found in an application you are building
- Facilitate resource discovery and reuse by sharing catalog definitions

3.7.1 How to Open the Resource Palette

The resource palette allows you to create connections to a number of different resources, such as application servers, databases, WebDAV servers, from where you can use them in different applications and share them with other users.

To open the Resource Palette:

In the main menu, choose **View > Resource Palette**.

By default, the Resource Palette is displayed to the right of the JDeveloper window.

3.7.2 How to Work With IDE Connections

The connections defined in the Resource Palette are listed in the IDE Connections panel of the Resource Palette.

When you create a connection in JDeveloper, you can create it in the context of the Resource Palette as an IDE connection that can be reused in different applications, or shared between users, or as application connections.

Some types of connections may appear in special connection-type navigators. For example, database connections display in the Database Navigator under the IDE Connection node (for IDE connections), or the *application-name* node (for application resource connections). The Database Navigator is where you edit objects through the database connection.

The different types of connection that can be made depends on the technologies and extensions available to you. To see what you can create a connection to, choose New Connection from the **New button** in the Resource Palette.

3.7.2.1 Resource Palette Connection Descriptor Properties Location

The file system location for the Resource Palette connection descriptor definition information is

system-dir/jdeveloper/system11.1.2.*n.nn.nn.nn*/o.jdeveloper.rescat2.model/connections/connections.xml

3.7.2.2 Defining the Scope of a Connection

In JDeveloper 11*g* you have two ways of creating and managing connections. You can define a connection to be used in the context of an application (called an Application Resource connection), or for the IDE as a whole (called an IDE connection). You use the same dialog to define both of these, but their scope within JDeveloper is different.

When you first create a connection you choose the connection scope. You cannot subsequently change the connection scope.

3.7.2.2.1 Application Resource Connections These connections are locally scoped and just available within the application. Connections in application resources are artifacts of the application and are deployed within the application. These types of connection are listed in the Application Resources panel of the Application Navigator, under the Connections node.

The file system location for the connection descriptor definition information is `application-folder/.adf/META-INF/connections.xml`, where `application-folder` is the path for the selected application.

3.7.2.2.2 IDE Connections These are globally defined connections available for reuse, and they are listed in the IDE Connections panel of the Resource Palette. You can copy IDE connections to the application navigator to use them within an application.

3.7.2.3 How to Create IDE Connections

You can create connections in the Resource Palette to resources available to JDeveloper.

The specific types of connections you can make depend on the technologies and extensions available to you.

To create an IDE connection:

1. In the Resource Palette IDE Connections panel, choose **New Connection** from the **New button**.
2. Choose the type of connection you want to create, and enter the appropriate information in the Create Connection dialog. For more information at any time, press F1 or click **Help** from within the dialog.

3.7.2.4 How to Edit IDE Connections

Once you have created a connection in the Resource Palette, you can edit details of the connection, but you cannot change the connection name.

To edit an IDE connection:

1. In the Resource Palette IDE Connections panel, choose **Properties** from the context menu of a connection.

2. The Edit connection dialog opens where you can change the connection details. For more information at any time, press F1 or click **Help** from within the Edit connection dialog.

3.7.2.5 How to Add IDE Connections to Applications

You can use connections in the Resource Palette in an application.

The connection can be added to the application currently open in JDeveloper, and it is listed in the Application Resources panel of the Application Navigator, under the Connections node.

To add a connection to an application:

In the Resource Palette IDE Connections panel, choose **Add to Application** from the context menu of a connection.

Alternatively, drag the resource from the Resource Palette and drop it onto an application page.

Alternatively, drag the connection from IDE Connections in the Resource Palette and drop it onto the Application Resources pane in the Application Navigator.

3.7.3 How to Search the Resource Palette

There are two ways of searching in the Resource Palette:

- Performing a simple search
- Performing an advanced search, where you enter parameters in a dialog

In addition, you can define a dynamic folder in a catalog where the content of the folder is defined by a query expression that is executed when the folder is opened.

The time the search takes depends on how many resources there are in the Resource Palette, and how long it takes to connect to them, and the results are displayed in the Search Results panel.

You can stop a search before it has completed by clicking the **Stop Search button**.

3.7.3.1 Performing a simple search

In this case, the search is performed across all the contents of the Resource Palette, and it may take some time because JDeveloper connects to remote resources during the search.

To perform a simple search:

1. In the Resource Palette, click the **Search Options button** to choose whether the search is performed against the Name, Type or Description of the resource. For more information at any time, press F1 or click **Help** from within the Resource Palette.
2. Enter a search string in the field. For example, if you want to find every resource that contains `dep` in the name, choose **Name** in step 1, and enter `dep`. Every resource that contains the string `dep` will be listed in the search results.
3. Click the **Start Search button** to start the search.

3.7.3.2 Performing an advanced search

You can specify a series of search criteria, and you can choose where to start the search from.

To perform an advanced search:

1. In the Resource Palette, choose **Advanced Search** from the context menu of an object in the My Catalogs panel or the IDE Connections panel. For more information at any time, press F1 or click **Help** from within the Advanced Search dialog.
2. Define where the search starts. Either select from **Search in**, or click **Show Hierarchy** which allows you choose within a hierarchical list of the Resource Palette contents.
3. Enter search criteria to return the resources you want, and click **Search**.

3.7.4 How to Reuse Resource Palette Searches

You can save a search and reuse it. There are two ways of saving a search in order to reuse it:

- As a dynamic folder, where the contents of the folder are created dynamically based on the search criteria when the folder is opened.
- As a static folder containing the results of the search.

Dynamic folders can also be created directly in a catalog.

To save a search:

1. In the Resource Palette Search Results panel, choose **Save Search** from the context menu.
2. In the Save Search dialog, choose:
   - **Save Search Criteria**, to create a dynamic folder.
   - **Save Search Results**, to create a static folder of results.

   For more information at any time, press F1 or click **Help** from within the Resource Palette.
3. Enter a name for the folder.
4. Choose the catalog to contain the folder, either from the dropdown list, or from the hierarchical list displayed when you click **Show Hierarchy**.

3.7.5 How to Filter Resource Palette Contents

Filters allow you fine-tune the contents of catalog folders.

To filter the contents of My Catalogs:

1. In the Resource Palette, choose **Filter** from the context menu of an object in the My Catalogs panel or the IDE Connections panel. For more information at any time, press F1 or click **Help** from within the Filter dialog.
2. Enter a string to define the filtering. Only entries in the folder that contain the string will be shown.

3.7.6 How to Import and Export Catalogs and Connections

Catalogs and connections are shared by importing Resource Catalog archive (.rcx) files that have been exported by another user.

To export a catalog:

> **Note:** When you select a catalog to export, any connections in the catalog are also selected. If you deselect the catalog before exporting, you must be sure to also deselect the connections that are not wanted in the archive file.

1. In the Resource Palette, choose **Export** from the context menu of an object in the My Catalogs panel or the IDE Connections panel.
2. In the Export Catalog and Connections dialog, select the catalogs and connections to be exported, and decide how errors will be handled. For more information at any time, press F1 or click **Help** from within the Export Catalog and Connections dialog.

To import a catalog:

1. In the Resource Palette, choose Import from (New).
2. In the Import Catalog and Connections dialog, specify or browse to the path and name of the Resource Catalog archive file (.rcx). For more information at any time, press F1 or click Help from within the Import Catalog and Connections dialog.
3. Choose the catalogs and connections you want to import, and determine how to handle errors.

3.7.7 How to Refresh the Resource Palette

You can refresh the contents of the Resource Palette.

To refresh the Resource Palette:

1. In the Resource Palette, choose **Refresh** from the context menu of an object in the My Catalogs panel or the IDE Connections panel.

3.7.8 How to Work With Resource Palette Catalogs

A catalog is a user-defined construct for organizing resources from multiple underlying repositories. The contents of a catalog and its associated folder structure can be designed to be used by an individual developer, or they can be targeted towards specific groups of users such as the UI designers for a development project.

Catalog folders organize resources in a catalog. You use catalog folders in the same way you would to organize files in a file system or bookmarks in a Web browser. Each catalog folder can contain any combination of:

- Folders.
- Dynamic folders, which are populated using a query.
- Filters, which are used to fine-tune the content of a folder or subtree.

3.7.8.1 How to Create Catalogs

You can organize the information in the resource palette in catalogs.

To create a catalog:

1. In the Resource Palette, choose **New Catalog** from the New button.

2. In the Create Catalog dialog, specify a name for the catalog. For more information at any time, press F1 or click **Help** from within the Create Catalog dialog.
3. (Optional) Provide a description for the catalog, and the email of the catalog administrator.

3.7.8.2 How to Rename Catalogs

You can rename catalogs.

To rename a catalog:

1. In the Resource Palette, right-click the catalog, and choose Rename from the context menu.
2. In the Rename dialog, specify a new name for the catalog. For more information at any time, press F1 or click Help from within the Rename dialog.

3.7.9 How to Work with Catalog Folders

You can create folders to organize the contents of catalogs.

3.7.9.1 How to Create Folders

You can organize the information within catalogs in folders.

To create a catalog folder:

1. In the Resource Palette, choose **New Folder** from the context menu of a catalog in the My Catalogs panel or the IDE Connections panel. For more information at any time, press F1 or click **Help** from within the Create Folder dialog.
2. Enter a name for the folder.

3.7.9.2 How to Create Dynamic Folders

Dynamic Folders provide a powerful way to dynamically populate a catalog folder with resources. The content of the folder is defined by a query expression that is executed when the folder is opened. The results of the query appear as the contents of the folder.

To create a dynamic folder:

1. In the Resource Palette, choose **New Dynamic Folder** from the context menu of a catalog in the My Catalogs panel or the IDE Connections panel. For more information at any time, press F1 or click **Help** from within the Create Dynamic Folder dialog.
2. Define the search criteria that will be used to populate this folder when it is opened.

3.7.9.3 How to Add Resources to a Catalog

You can add a connection from the IDE Connections panel or a resource from the Search panel in the Resource Palette to a catalog in My Catalogs.

To add a resource to a catalog:

1. In the Resource Palette, right click a connection in the IDE Connections panel, or the result of a search in the Search panel and choose **Add to Catalog** from the context menu.

2. The Add to Catalog dialog opens for you to specify the name for the resource in the catalog, and the catalog to add it to. For more information at any time, press F1 or click **Help** from within the Create Connection dialog.

Alternatively, you can drag an item from under **IDE Connections** and drop it on a catalog or catalog folder.

You can reorganize a catalog by selecting an item or folder in the catalog and dragging it to another folder in the same catalog, or to another catalog.

3.8 Working with Source Files

JDeveloper includes an editor for editing source files across several technologies, including Java and XML, among others.

3.8.1 Using the Source Editor

JDeveloper includes an editor for editing source files across several technologies, including Java and XML, among others.

Depending on the type of source file you are editing, the source editor will be available in one of the following forms:

- Java Source Editor
- XML Editor
- HTML/JSP Source Editor
- JavaScript Editor

In addition to technology-specific features, the source editor also has a set of common features across all technologies that enhance the coding experience. These features include bookmarking, code insight, code templates, and several other features that enable you to code faster and better.

Use the Code Editor page in the Preferences dialog to customize the source editor to suit your coding style.

The source editor offers a set of common features across all technologies that provide intuitive support for a variety of coding tasks. Available across all forms of the editor, these features enhance your coding experience through quicker execution of coding tasks and better navigation through code.

Breadcrumb Navigation

The breadcrumb bar, located at the bottom of the editor window, shows the hierarchy of code entities from the current caret position up to the top of the file. Hovering the mouse cursor over a node pops up some information about the node, and clicking on the node navigates the caret to the node location.

A breadcrumb can be clicked to display a popup list of child breadcrumbs can be displayed (where appropriate). For example, for a Java class, you can click the breadcrumb to display the class' methods and inner classes in a list. Choosing an item on this list will navigate the editor to its location.

If block coloring has been activated and colors have been assigned, breadcrumbs are highlighted in the same color as their corresponding code blocks.

Overview Popup

The right margin of the editor provides colored overview marks that are indicators for a location in the source file. Hovering the mouse over an overview mark makes a popup appear which displays information about the item in that location of the source file, and a snippet of the relevant code.

The following overview indicators are provided:

- A square mark at the top right corner of the editor window indicates the overall health of your source file, as per its color. White indicated that the health is currently being calculated. Green indicates that there are no errors or warnings in the file. Red indicates errors, and yellow indicates warnings
- Rectangles, depending on their color, signify the occurrence of the following source editing artifacts:
  - Red: Java code error
  - Pale blue: bookmark
  - Medium blue: current execution breakpoint
  - Yellow: occurrence of searched text
  - Pale orange: Java warning
  - Bright orange: Profile Point

You can also press Ctrl anywhere in the right margin to view a popup window that displays a portion of the source code that is not currently in view. By adjusting the position of the mouse while pressing Ctrl, you can view the entire code without scrolling in the editor itself

Hovers

Hovers enable you to position the mouse cursor over certain areas of the IDE and get some information on them in a popup window that appears floating in front.

Whitespace Display

Tools menu > Preferences > Code Editor > Display > Show Whitespace Characters

This feature optionally renders spaces, new lines, carriage returns, non-breaking spaces, and tab characters as visible characters in the editor. Turned off by default, this can be enabled and disabled using the Preferences Dialog.

Duplicate Selection

Edit menu > **Duplicate Selection**

Duplicates the currently selected block of code, and places the copied code beside the original code. After duplication, the newly inserted code is selected. The clipboard is not affected by this operation.

Vertical Selection

Edit menu > Wrap Selection

This feature enables you to select code vertically. when you do not want to select text that wraps around the end of lines. This is useful for selecting tabular data, or vertically aligned code blocks.

Join Lines

Join the current line to the next, or join all lines in a selection. Any comment delimiters or extra whitespace are intelligently removed to join the lines.

Default keyboard shortcut: Ctrl+J

Cursor Position

When the source editor is in use, the status bar at the bottom displays the line and column coordinates of the current position of the cursor.

Mouse Wheel Zoom

Hold down the Ctrl key and use the mouse scroller to zoom in to or zoom out of the code editor.

3.8.1.1 Features Available From the Context Menu

The generic source editor also provides a set of features through the context menu. To use these features, in the context menu, select **Source**. Depending on the type of source file in use, items other than the ones mentioned below may be present in the context menu. For example, the Java Source Editor contributes Java-specific options to the source editor context menu.

> **Note:** These features are also available through the **Source** menu.

Completion Insight

Completion insight provides you with a list of possible completions, such as method names, and parameter types if they are applicable, at the insertion point, which you may use to auto-complete Java code you are editing. This list is generated based on the code context found at the insertion point. The contexts supported by completion insight are:

- Within package and import statements
- Within extends, implements, and throws clauses
- Within continue and break statements
- Within general code expressions

Default keyboard shortcut: Ctrl+Space

Parameter Insight

Parameter insight provides you with the types and names of the parameters of the method call you are typing. If the method is overloaded, multiple sets of parameter types and names are listed.

Default keyboard shortcut: Ctrl+Shift+Space

> **Note:** If errors for the file appear in the Structure window, Code (Completion or Parameter) Insight may not work. If the class(es) you are using are not in your project (that is, not on your classpath), Code Insight will not appear. Please note that you may need to compile your src files in order for Code Insight to have access to them.

Complete Statement

Use to auto-complete code statements where such a completion is obvious to JDeveloper; for example, semi-colon insertions at the end of a statement.

Default keyboard shortcut: Ctrl+Shift+Enter

Expand Template

Insert a code template from a list of JDeveloper's predefined code templates. The code templates offered are context sensitive. For example, templates to declare class variables are only offered when the cursor is in the appropriate place in the class file.

Default keyboard shortcut: Ctrl+Enter

Code Assist

Code Assist examines your code in the editor and provides assistance to fix common problems. A **Code Assist** icon appears in the editor margin when JDeveloper has a suggestion for a code change. To invoke Code Assist manually, press Ctrl+Alt+Enter. To select an action listed in Code Assist, press Alt+ the underlined key.

Default keyboard shortcut: Ctrl+Alt+Enter

QuickDoc

Select to view the Javadoc or Jsdoc (depending on whether you are using the Java or JavaScript editor) for the element in focus.

Default Keyboard Shortcut: Ctrl+D

Toggle Line Comments

Comments out the line currently in focus in the source editor. Running this command on a commented line uncomments the line.

Default Keyboard Shortcut: Ctrl+Slash

Indent Block

Indents the line of code currently in focus. If a block of code is selected, the entire block is indented.

Unindent Block

Unindents a line or block of code, based on code has focus in the editor.

3.8.2 How to Set Preferences for the Source Editor

You can change the default settings of many of the features of the source editor by changing the preferences.

You can also view or change shortcut keys for the source editor, by modifying the predefined keyboard schemes.

To set tabs for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Code Editor** node, then the **Code Style** page.
3. On the Code Style page, select the **Edit** button.
4. On the **Format** tab, open the **Indentation** node and select **Tab Size**.

5. Change the tab size value as required.
6. Click **OK** to close the dialogs.

To set fonts for the Source Editor

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, select the **Code Editor** node, then the **Fonts** node.
3. On the Fonts page, select a font type and size. Alter the sample text, if you wish. The sample text display reflects your font changes.

   By default, all your system fonts are loaded. To limit the fonts available on this page to fixed-width fonts, select **Display Only Fixed-Width Fonts**.
4. Click **OK**.

To set caret behavior for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Caret Behavior** node.
4. On the Caret Behavior page, set the different attributes that determine how the caret will look and behave.

   For more information, press F1 or click **Help** from within the dialog page.
5. Click **OK**.

To set the options for width and the right margin in the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Display** node.
4. On the Display page, enter the settings you wish for the right margin.
5. Enter a width for the source editor, expressed in numbers of columns.
6. Click **OK**.

To set line gutter behavior for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Line Gutter** node.
4. On the Line Gutter page, decide whether or not line numbers will appear.
5. Set the other attributes to create the line gutter behavior that you want.
6. Click **OK**.

To set the options for syntax highlighting in the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog

2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Syntax Colors** node.
4. On the Syntax Colors page, begin by selecting the appropriate category for the syntax you wish to work with.

   The display on the page changes to reflect the current settings for the first style listed in this category, which is highlighted.
5. With the category displayed above, select any individual style in the **Available Styles** list to view its current settings.
6. Select a font style and set the background and foreground color as desired. The sample text changes accordingly.
7. Click **OK**.

To set bookmark options for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Bookmarks** node.
4. On the Bookmarks page, decide how you wish to handle bookmarks once you've exited the editor or Oracle JDeveloper, how you wish to traverse bookmarks, and how you wish to handle bookmarks at the end of files for lines that may no longer exist.
5. Click **OK**.

To set the options for Code Insight in the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Code Insight** node.
4. On the Code Insight page, select the appropriate checkboxes to enable completion insight or parameter insight and use the sliding bar to set the delay time before the popup window appears.
5. Click **OK**.

3.8.3 How to Customize Code Templates for the Source Editor

Code templates assist you in writing code more quickly and efficiently while you are in the source editor. You can edit the existing templates or create your own.

To view existing code templates:

1. From the main menu, choose **Tools > Preferences**, expand the **Code Editor** node, and select **Code Templates**. For more information at any time, press F1 or click **Help** from within Preferences dialog.
2. On the Code Templates page, scroll through the shortcuts, which represent the letters you must type to evoke each template.

3. Click on any shortcut to view the associated template code on the **Code** tab. If there are any imports associated with this template, they will be shown on the Imports tab.

To edit an existing code template:

1. From the main menu, choose **Tools > Preferences**, expand the **Code Editor** node, and select **Code Templates**.
2. On the Code Templates page, make changes to the shortcut, the description, the code (including the variables used in it), and the imports, as required.
3. When you are finished, click **OK**.

To define a new code template:

1. From the main menu, choose **Tools > Preferences**, expand the **Code Editor** node, and select **Code Templates**.
2. On the Code Templates page, click **Add**. The cursor jumps to the bottom of the **Shortcut** list and a new row is added.
3. Type in the name for the new shortcut and add a description in the list next to it.
4. Select the **Code** tab and enter the code for this template. Note that cursor position is a part of the template, representing the logical insertion point for new code to be entered when the template is used. Select the **Imports** tab and enter any imports associated with this template.
5. Click **OK**.

To customize the HTML and JSP options for the source editor:

1. Choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within Preferences dialog.
2. Expand the **Code Editor** node.
3. Select the **XML and JSP/HTML** node.
4. On the XML and JSP/HTML page, select **End Tag Completion** to enable that option.
5. Click **OK**.

To set undo behavior for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Undo Behavior** node.
4. On the Undo Behavior page, use the slider bar to set the number of actions of the same type to be combined into one undo.
5. Select or deselect the options for combining insert-mode and overwrite-mode edits and for combining the deletion of next and previous characters.
6. If you wish to be able to undo navigation-only changes, select the appropriate checkbox. If you enable this setting, use the slide bar to set the number of navigation changes to be combined into one undo.
7. Click **OK**.

To set printing options for the source editor:

1. From the main menu, choose **Tools > Preferences**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Printing** node.
4. On the Printing page, set the various print options.
5. Click **OK**.

3.8.4 How to Manage Source Files in the Editor Window

Oracle JDeveloper possesses several capabilities for easier handling of files in the editor window.

3.8.4.1 Maximizing the View of a File

You can open a file to fill the maximum view available in JDeveloper. This is done by maximizing the source editor to fill JDeveloper

The same technique of double-clicking a tab can be used for any of the other windows in JDeveloper, for example, the Help Center, or the Application Navigator.

To maximize the view of a file:

- In the source editor, double-click the tab of the file. The source editor becomes the only window visible in JDeveloper, with the file you have chosen currently displayed in it.

To reduce the view of a file to its former size

- Double-click the tab of the file again. The windows within Oracle JDeveloper return to their former layout.

3.8.4.2 Navigating Between Open Files in the Editor Window

You can navigate through files visually (cycling through by tab), historically (cycling through by order of access), or numerically (cycling through based on file shortcut key assignment).

To navigate through open files by tab:

- Press Alt+Left Arrow or Alt+Right Arrow. Use Alt+Left Arrow to navigate to the left, and Alt+Right Arrow to navigate to the right.

To navigate through open files based on history:

- Press Ctrl+Tab or Ctrl+Shift+Tab

  Use Ctrl+Tab to open the last active file. Note that opening a file renders it the currently active file, such that the previously active file now becomes the last file to have been active.

  For example, given files A, B, and C (opened in the order C, B, A), where file A currently has focus, pressing Ctrl+Tab brings B to the foreground. Now B is the file with focus and A is the last active file. Pressing Ctrl+Tab again thus brings A back to the foreground.

- Press Ctrl+Tab+Tab+Tab to cycle through files by order of access without stopping. Only when you stop on a file is that file given focus. Stopping on a file is equivalent to using Ctrl+Tab on that file.

3.8.4.3 How to Display the List of All Currently Open Files

You can display all the files currently open in the editor window, or all the files currently open in a particular tab group.

To display the alphabetical list of all the files currently open in a given tab group:

Click the **File List** button in the upper right-hand corner of the editor window. Alternately, with the focus in the editor window, press (in the default keyboard scheme) Alt+0.

If the editor window is not subdivided, the list will contain all open files. If the editor window is subdivided, the list will contain all the open files in that tab group.

To display the alphabetical list of all the files currently open in the editor window, regardless of split or detached files:

- From the main menu, choose **Window > Windows**.

3.8.4.4 How to Access a Recently Opened File

Oracle JDeveloper remembers the last files you have edited.

To access a recently-edited file, irrespective of whether it is currently open or not:

1. From the main menu, choose **Navigate > Go to Recent Files** or (in the default keyboard scheme) press Ctrl+ =.
2. In the Recent Files dialog, select the file from the list or begin typing the first letters of the filename.
3. Click **OK**.

By default, only those files opened directly (through the navigator, for instance) appear in the list. Those opened indirectly (for example, as you debug code) do not automatically appear. To view files opened both directly and indirectly, select **Show All**.

3.8.4.5 How to Open Multiple Editors for a File

You can split the editor window horizontally or vertically, opening a single file in multiple views. In each view, you've the choice of changing which editor the file is opened in.

You can split a file into as many views as you like. The split views are automatically synchronized with each other.

To open a single file in multiple views:

1. From the main menu, choose **Window > Split Document**.

   The editor window is now split in two, with two identical and independent windows opened on the same file. Each window has its own set of editor tabs at the bottom.

2. In each window, select the editor tab to view the file in that editor.

   Note that some editors (such as the Java Visual Editor) permit only one view at a time on a file.

Alternately, you can split the file using the mouse, either horizontally or vertically.

To split the file horizontally, grab the splitter just above the vertical scroll bar (on the upper right-hand side of the window) and drag it downward.

To split the file vertically, grab the splitter just to the right of the horizontal scroll bar (on the lower right-hand side of the window) and drag it left.

To navigate quickly between split views:

- Press F6 to cycle forward.
- Press Shift+F6 to cycle backward.

To collapse those multiple views back into one:

- From the main menu, choose **Window > Unsplit Document**.

Alternately, you can drag the splitter past the end of the editor window.

3.8.4.6 Viewing More Than One File at a Time

You can split the editor window horizontally or vertically, opening views on more than one file at a time. Each view is independent of the others

You can split the editor window into as many different independent views as you would like.

To view more than one file at a time, in independent windows:

- From the main menu, choose **Window > New Tab Group**.

  The editor window is now split in two, with different files in each window. Each window has a set of document tabs at the top and a set of editor tabs at the bottom. Each window is known as a tab group.

  You can create as many tab groups as you like.

Alternately, you can detach a file using the mouse, by grabbing the document tab for the file and dragging it towards the area of the window where you want the file displayed.

As you drag the tab, the icon that follows the cursor changes. A split window with an arrow to the left, right, top, or bottom indicates that if you release the mouse now, the new window will be placed in that relationship to the current window.

To move a file to a different tab group:

1. Drag the document tab for the file to the center of the area occupied by the tab group you wish to attach it to.
2. When the icon that follows the cursor changes to show a miniature window with tabs, release the mouse.

To collapse those multiple views back into one:

- From the dropdown menu, choose **Window > Collapse Tab Groups**.

Alternately, you can simply grab the document tab for a detached file and drop it onto an existing tab or tab group. When the icon changes to show a miniature window with tabs, release the mouse.

3.8.4.7 How to Quickly Close Files in the Editor Window

You can close any file open in the editor window with a single click.

To close the current file, choose one of the following ways:

- From the main menu, choose **File > Close**.
- Press Ctrl+F4.
- In the editor, right-click the tab for the current file and choose **Close**.
- Hover the mouse over the tab for the current file and click the **Close** button.

To close all files, choose one of the following ways:

- From the main menu, choose **File > Close All**.
- Press Ctrl+Shift+F4.
- In the editor, right-click the tab for any file and choose **Close All**.

To close all files except one:

- In the editor, right-click the tab for the file you want to stay open and choose **Close Others**.

To close multiple files at once:

1. From the main menu, choose **Window > Windows**.
2. In the Windows dialog, select the files to be closed and click **OK**.

To selectively close files:

1. In the editor, select the corresponding tab for the file to be closed.
2. Ctrl+click the tab, or hover the mouse over the tab and click the **Close** button.

3.8.5 Working with Mouseover Popups

Mouseover Popups enable you to position the mouse cursor over certain areas of the IDE and get some information on them in a popup window that appears floating in front. Information is available on the following:

- Javadoc
- Source code
- Data values while debugging
- Breakpoints

The popup window appears when you move the mouse over and optionally press the key that you assign for the feature. The following are some of the areas of the IDE that mouseover popups are available for:

- Structure window
- Text in an editor

Smart-Popup

The Smart-Popup feature shows the most appropriate popup for a given situation, depending on the order of popups specified in the Mouseover Popups page of the Preferences dialog. Smart-Popup is activated by a keystroke which you can specify on the Mouseover Popups page of the Preferences Dialog.

For example, you may have the following popup configuration (set using the Mouseover Popups page of the Preferences dialog)

- Smart-Popup is enabled and configured on the Control key.
- The Data Values, Documentation, and Source popups all have Smart-Popup enabled and are ordered in the following way: Data Values, Documentation, Source Code in the Mouseover Popups table.

With this configuration, if you hover the mouse over a variable in the source editor and press Control, then:

- The Data Values popup is considered first. If you are debugging and the mouse hovers over a variable with a value, the Data Value popup is displayed.
- If no popup is displayed for the previous step, then the Documentation popup is considered next. If the variable has any documentation, it is displayed in a popup window.
- If no popup is displayed for the previous step, then the Source popup is considered next, and the source code for the variable (if available) is displayed in a popup window.

With Smart-Popup, you only need to use the Smart-Popup activation keystroke for the IDE to display the most appropriate popup

---

Note: Even with Smart-Popup enabled, the individual popups for Data Values, Documentation, and Source Code can still be activated by their respective activation keys.

3.8.6 How to Locate a Source Node in the Navigator

You can quickly locate the source node in the Application Navigator for any file opened for editing, whether or not that node is in the current project.

To locate the node for any file opened in the editor:

1. Make sure that the focus in the editor is on the file you wish to locate.
2. From the context menu, choose Select in Navigator.

3.8.7 How to Set Bookmarks in Source Files

You can use bookmarks in your source files to help you quickly locate relevant code. You can use the Bookmarks Window to navigate to bookmarked material.

To set or remove a bookmark in a source file:

1. Within the file, place the cursor in the gutter of the line you would like bookmarked.
2. Right-click and choose **Toggle Bookmark**.

3.8.8 How to Edit Source Files

Oracle JDeveloper provides several features for editing source files.

3.8.8.1 How to Open Source Files in the Source Editor

JDeveloper provides a powerful source editor that will help you write different kinds of code quickly and efficiently.

You can set preferences for the specific editor for each file type.

To open your source code in its default editor:

- In the Application Navigator, double-click the file or right-click and choose Open.

  The default editor associated with that file type appears in the content area. If the editor is already open on that file, the editor comes to the foreground.

To open your source code in a specific editor or viewer:

1. In the Application Navigator, double-click the file or right-click and choose Open.
2. In the editor window, select the appropriate editor tab.

Changes made in the source will be immediately reflected in other views of that file.

You can also generate Java source code from modeled Java classes.

3.8.8.2 How to Edit Source Code with an External Editor

It is possible to edit source code that you have opened in JDeveloper with an outside editor, should you wish to do so. When you return to the JDeveloper IDE, it will detect the changes you have made.

Before you edit a file externally, you should first save any changes made in JDeveloper. If you do not, when you return to JDeveloper, you will be asked whether to reload those files or not. If you reload the externally modified files, you will lose the unsaved changes made in JDeveloper. If you do not reload them, you will lose the changes made outside JDeveloper once you save the file in JDeveloper.

To edit source code with an external editor, with the file open in JDeveloper:

1. Save any changes made to the file open in JDeveloper.
2. Edit your file externally and save your changes to the disk.
3. Return to JDeveloper and to the file open in the source editor.

   By default, the file is reloaded in JDeveloper without a confirmation beforehand. To receive a confirmation dialog, deselect the **Silently Reload When Buffer Is Unmodified** option on the Environment page of the Preferences dialog.

3.8.8.3 How to Insert a Code Snippet from the Component Palette into Source Files

Once you have added code snippets to the Component Palette, you can add them to files open in the editor.

Alternatively, you can use code templates to assist you in writing code more quickly and efficiently while you are in the source editor.

To insert a code snippet from the Palette into a source file:

1. Open the file in the source editor.
2. If the Component Palette is not visible, open it by choosing **View > Component Palette**.
3. In the Palette dropdown list, select Code Snippets or the snippets page you have defined.

   The snippets defined for that page appear listed to the right. Toggle between list and icon views by right-clicking and choosing the view you want from the context menu.
4. Position your cursor in the file at the point where the snippet is to be inserted.
5. In the Palette, click the snippet name or icon.

The code snippet appears in the file.

3.8.8.4 How to Record and Play Back Macros in Source Files

You can record, and play back, keystroke sequences in files open in the source editor.

To define shortcut keys for recording and playing back:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select the **Shortcut Keys** node.
3. On the Shortcut Keys page, in the **Search** field, enter `Macro Toggle Recording`.
4. You will see the **Macro Toggle Recording** action selected under **Available Commands**.
5. To assign a shortcut, place focus in the **New Shortcut** field, and enter a shortcut by pressing the key combination on the keyboard.

   If this proposed shortcut already has an command associated with it, that command will appear in the **Conflicts** field.
6. To assign the shortcut you have specified, click **Assign**.
7. Now, in the **Search** field, enter `Macro Playback`.
8. Repeat steps 5 and 6 to assign a shortcut for playing back the macro.
9. Click **OK**.

To record a macro:

1. Open the source file in an editor.
2. To begin recording, press the key combination you have defined for recording macros.
3. Now enter the keystroke sequence you wish to record.
4. To end recording, again press the key combination you have defined for recording macros.

To play back a macro:

1. Open the source file in an editor.
2. Position your cursor in the open file.
3. Press the key combination you have defined for playing back macros.

3.8.8.5 How to Create Tasks

You can create tasks that are directly related to lines in files of source code, or tasks that are associated with applications, projects or general files. Oracle JDeveloper comes with the tags `TODO`, `TASK`, and `FIXME` and the priorities `HIGH`, `MEDIUM`, `LOW` and `NONE` preconfigured, and you can add your own task tags and priorities in the Tasks page of the Preferences dialog.

To add your own task priorities and task tags:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select the **Tasks** node.
3. On the Tasks page, alter the priorities and source tags to suit your requirements.

For more information, press F1 or click **Help** from within the dialog page.

4. Click **OK**.

To create a task associated with a comment line in source code:

1. Open the tasks window by choosing **View > Tasks Window**.
2. Within the source code file, create a comment line starting with `//` and one of the task tags, for example `//TODO`.
3. Continue to type the comment, which will at the same time appear as the description of the task in the tasks window.
4. Set the other options in the task window as required. For help while using the tasks window, press F1.

To create a task associated with an application, project or file:

1. In the navigator, select the object for which you wish to create a task.
2. Open the tasks window by choosing **View > Tasks Window**.
3. Select the **Add Tasks** button, which will open the Add Task dialog.
4. Complete the dialog for the task that you want to create. For help while using the Add Task dialog, press F1.

3.8.9 How to Compare Source Files

You can compare source files either belonging to the same project, or outside.

To compare a file currently being edited with its saved version:

1. Place the focus on the current version open in the editor.
2. Select the History tab in the editor window.

The saved file opens side by side with the file in the editor buffer.

To compare one file with another file outside the project:

1. Place the focus on the file in the editor to be compared.
2. From the main menu, choose **File > Compare With Other File**.
3. In the Select File to Compare With dialog, navigate to the file and click **Open**.

The two files open side by side, under a tab labeled **Compare**.

To compare any two files within the same project:

1. In the navigator, select the two files to be compared.
2. From the main menu, choose **File > Compare With > Each Other**.

The two files open side by side, under a tab labeled **Compare**.

3.8.10 How to Revert to the Last Saved Version of a File

While you are in the process of making changes to a file, at any time you can revert to the last saved version of the file.

To revert to the last saved version of a file:

1. While the changed file has focus in the editor, from the main menu choose **File > Replace With > File On Disk**.
2. In the Confirm Replace dialog, click **Yes**.

Any changes you have made since the last save are now undone.

3.8.11 How to Search Source Files

Oracle JDeveloper provides a powerful source editor that will help you write different kinds of code quickly and efficiently.

To search a source file currently open in the source editor, with the option to replace text:

1. With the file open in the editor, ensure that the editor has focus.
2. Optionally, if an instance of the text you want to search for is easily found, you can highlight it now.
3. From the main menu, choose **Search > Find**. Alternatively, press Ctrl+F.
4. In the Find Text Dialog, enter or select the text to locate.

   Text previously searched for in this session of JDeveloper appears in the **Text to Search For** dropdown list.
5. Select other search parameters accordingly.

   For more information, press F1 or click **Help** from within the dialog.
6. Click **OK**.

To do a simple search in the open source file for a single text string:

1. With the file open in the editor, ensure that the editor has focus.
2. Place the cursor in the file at the point you wish to search from.
3. From the main menu, choose **Search > Incremental Find Forward** or **Search > Incremental Find Backwards**.
4. In the dialog, enter the search text.

   As you type, the cursor jumps to the next instance of the group of letters displayed.

Alternatively, enter the text string in the search box. As you type, the cursor jumps to the next instance of the group of letters displayed. Use the **Previous** or **Next** buttons to search up and down the file. Click in the search box to set **Match Case**, **Whole Word**, or **Highlight Occurrences**.

To search all files in a project or an application:

1. From the main menu, choose **Search > Find in Files**.
2. In the Find in Files dialog, enter or select the text to locate.

   Text previously searched for in this session of Oracle JDeveloper appears in the **Search Text** dropdown list. By default, if you opened this dialog with text selected in the source editor, that text appears as the first entry.
3. If you want to choose the file types that are included in the search, click the **File Types** button to open the File Types To Include dialog. By default, all file types will be searched.

4. Select other search parameters as required.

   For more information, press F1 or click **Help** from within the dialog.

5. Click **OK**.

3.8.12 How to Print Source Files

Oracle JDeveloper enables you to print source files.

To print a source file:

1. Display the file to be printed in an editor, or select its filename in the navigator.
2. From the main menu, choose **File > Print**.
3. In the Print dialog, select your print options.
4. Click **OK**.

3.8.13 Reference: Regular Search Expressions

Regular expressions are characters that customize a search string through pattern matching. You can match a string against a pattern or extract parts of the match.

JDeveloper uses the standard Sun regular expressions package, java.util.regex. For more information, see "Regular Expressions and the Java Programming Language" at http://download-llnw.oracle.com/javase/tutorial/essential/regex/ .

3.9 Working with Extensions

Extensions are components that are loaded and integrated with JDeveloper after it is started. Extensions can access the IDE and perform many useful tasks. In fact, much of JDeveloper itself is composed of extensions. Most of the basic functionality in JDeveloper is implemented as extensions—software packages which add features and capabilities to the basic JDeveloper IDE. You can add existing extensions into JDeveloper, or create your own.

This section contains information on finding, installing, and enabling or disabling JDeveloper extensions. The simplest way to find and download JDeveloper extensions is through the Check for Updates wizard.

If you need additional capabilities from the IDE (such as integration with a version control system or a special editor or debugger), you can add external tools to JDeveloper. See Section 3.12, "Adding External Tools to JDeveloper" for more information. In addition, you can obtain additional extension development tools and functionality in the Extension Software Development Kit (SDK). You can download the Extension SDK via the Check for Updates wizard.

You can also download the Extension SDK from the Oracle Technology Network Web page.

Note: Any time an extension is added or upgraded, the migration dialog appears at startup in case you need to migrate any previous settings related to that extension.

3.9.1 How to Install Extensions with Check for Updates

The easiest way to find and install extensions is to use the Check for Updates wizard.

To install extensions using the Check for Updates wizard:

1. From the **Help** menu, select **Check for Updates**.
2. Follow the steps in the wizard to browse, download, and install patches and extensions.

You can also access the Check for Updates wizard from the **Tools > Features** page.

3.9.2 How to Install Extensions from the Provider's Web Site

Some extension providers prefer to have you install directly from their Web site, so that among other things they can contact you when there are updates to the extension. In this case, the Check for Updates wizard will inform you of the provider's preference, and will then open your default Web browser so that you can conduct the download and installation from the provider's Web site.

To download and install from the provider's Web site:

- Follow the instructions on the provider's Web site for downloading and installing the extension. Be sure to note any comments or instructions on registration, configuration, or other setup requirements.

3.9.3 How to Install Extensions Directly from OTN

You can find and download extensions from the JDeveloper Extensions Exchange website on OTN. The page is located here:

`http://www.oracle.com/technetwork/developer-tools/jdev/index-099997.html`

The available extensions include:

- JUnit Extension, an extension you can use to create and run test cases, test suites, and test fixtures, using JUnit.
- *i*SQL*Plus Extension, an extension that enables you to load or execute SQL*Plus scripts from within JDeveloper.
- Oracle Business Intelligence Beans, a set of standards-based JavaBeans™ that enables developers to build business intelligence applications.
- Other extensions to JDeveloper contributed by the JDeveloper community.

To install extensions after you have downloaded them from OTN:

- For extensions created for the current release, see the *Oracle Fusion Middleware Developer's Guide for Oracle JDeveloper Extensions*.
- For extensions created for earlier releases, see: "Extension Packaging and Deployment For Previous Versions of JDeveloper" in the Extension SDK. Extensions were packaged differently and placed in a different location in earlier releases.

3.9.4 How to Install Extensions Using the JDeveloper dropins Directory

JDeveloper supports the concept of a "watched directory". A watched directory is a location where a user or script can drop files and have them discovered by JDeveloper automatically the next time it starts.

To install an extension using the dropins directory:

- Drop your extension jar in the JDeveloper dropins directory, which is located in the `jdeveloper/dropins` folder.
- Additional dropins directories can be specified via the `ide.bundle.search.path property`, either at the command line or by adding an entry in the jdev.conf file.

3.10 Using the Online Help

You can access the JDeveloper online help through the Help Center. This section describes how you can effectively use the features of the Help Center.

The JDeveloper Help Center comprises two windows: the help navigator and the help topic viewer.

The following types of content are available:

- Conceptual and procedural information, which is available in this guide.
- Context sensitive online help topics, which open when you press F1 or click Help in a dialog or wizard, or click the help icon in a wizard.
- Developer guides, which provide end-to-end information for developing applications with specific technologies.
- Tutorials, which provide introductions to many JDeveloper application scenarios.

From the Help Center, you can also access additional documentation on Oracle Technology Network (OTN).

The Help Center search feature lets you search the installed documentation, the documentation available from OTN, and the Fusion Middleware Documentation Library.

You can also customize the way you view content.

3.10.1 Using the Help Center

The Help Center enables you to browse the table of contents, locate relevant topics in the dynamic help links lists, and do a full text search of installed and online content. It also provides a Favorites navigator for saving links to frequently referenced topics. The Help Center comprises two windows: the help navigator and the help topic viewer. You can customize some aspects of these windows.

The following table describes the features available in the Help Center toolbar.

Table 3–27 Help Center Toolbar Icons

Icon	Name	Description
	**Keep Help Center on Top** (Alt+K)	Keeps the Help Center on top of all other open windows.
	**Navigators**	Opens Help Center navigators you have previously closed.

Table 3–27 (Cont.) Help Center Toolbar Icons

Icon	Name	Description
	JDeveloper **Forum**	Launches an external browser window and visit the JDeveloper Forum on Oracle Technology Network (OTN).
	**Search**	Searches all the documentation installed as online help, Oracle Technology Network (OTN) and the Fusion Middleware and Database Libraries.

The Help Center includes tabs for navigating content on the left:

- **Contents** - Displays the table of contents for all installed content in the help system, including traditional online help, tutorials, developer guides, and the user guide.
- **Favorites** - Displays folders of user defined help topics and external links you have saved for quick retrieval.

The Help Center includes the following tabs for viewing content and search results on the right:

- **Help content viewers** - Display the selected online help and developer guide contents. Multiple tabbed pages open for selected content.
- **Tutorial viewer** - Displays a selected tutorial. Only one tutorial viewer opens.
- **Search results** - Displays the results of the full text search.

3.10.2 How to Open the Online Help

The JDeveloper Help Center comprises two windows: the help navigator and the help topic viewer.

To open the online help, use any of these methods:

- Press F1, click **Help**, or click the **Help** icon at any time to display context-sensitive help.
- From the main menu, choose **Help > Search**.
- From the main menu, choose **Help > Table of Contents**.
- From the main menu, choose **Help > Help Favorites**.
- From the Start page, choose any link with a tutorial, book or help topic icon.

To see a help page that is already open:

- Select a tab at the top of the help topic window.
- Click the scroll buttons at the top of the help topic window to scroll through all available tabs and select a tab.
- Click the **Tab List** button at the top of the help topic window to display the list of all available pages and select a page.

3.10.3 How to Search the Documentation

You can search all the documentation installed as online help by doing a full-text search, and you can also search Oracle Technology Network (OTN) and the Fusion

Middleware and Database Online Documentation Libraries. You can search an individual help topic that is open by using the **Find** icon in the topic viewer toolbar.

To do a full-text search from the Help Center:

1. If the Help Center is not open, from the main menu, choose **Help > Search**.
2. In the **Search** field, enter the word or phrase you are searching for.
3. Optionally, open the **Search Options** menu and select the locations you want to search. By default, **Local Documentation** and the **Fusion Middleware** library are selected.
4. Set the other search options as needed; these apply only to the online help search.
5. Click the **Go** icon or press Enter.

   The Search Results page opens in the help viewer area, with the titles and sources of each matching document, as well as the beginning text.
6. To select a topic, double-click its title.

   Each help topic opens in a separate tabbed page. The Search Results page remains available. Each OTN and Documentation Library page opens in your default browser.

Using the Boolean Expressions option:

BooleanExpression is a recursive tree structure for expressing search criteria involving boolean expressions. The BooleanExpression is based on the following grammar:

```
BooleanExpression ::
     BooleanExpression AND BooleanExpression
     BooleanExpression OR  BooleanExpression
     BooleanExpression NOT BooleanExpression
     BooleanExpression + BooleanExpression
     BooleanExpression -  BooleanExpression
     + BooleanExpression
     - BooleanExpression
     NOT BooleanExpression
     StringExpression (base case)
```

To begin a documentation search from the main toolbar Search field:

1. In the Search field, enter the word or phrase you are searching for.
2. Open the **Search Options** menu and select only the documentation: **Help: Local**, **Help: OTN**, **Help: iLibrary**. Deselect other locations.

   By default, all locations are selected.
3. Click the **Go** icon or press Enter.

   The Help Center opens with the Search Results page on the right, showing the titles and sources of each matching document, as well as the beginning text.

3.10.4 How to Add Bookmarks to the Favorites Page

You can save links to frequently referenced help topics, stored in folders you create and name, on the Favorites page in the Help Center. The help topic must be open in the help topic viewer, in order to bookmark it. You can also add links to external sites.

To add links to help topics to the Favorites page:

1. Click the **Add to Favorites** icon in the help topic viewer toolbar.

The Add to Favorites dialog is displayed.

2. Select the folder to which you want to add the link and click **OK**.

To add links to external sites to the Favorites page:

1. Click the **Add External Favorites** icon in the Favorites page toolbar, or right-click a node on the Favorites page and choose **Add External Favorites** from the context menu.

   The Add External Favorites dialog is displayed.

2. Enter a title for the page or document in the **Name** field.
3. Enter the fully qualified path in the **URL** field.
4. Select the folder to which you want to add the link and click **OK**.

To create a new Favorites folder:

1. Click the **New Folder** icon in the Favorites page toolbar, or right-click a node on the Favorites page and choose **New Folder** from the context menu.
2. Enter the new folder name and click **OK**.

You can also create a new folder when the Add to Favorites dialog is open, by clicking **New Folder**.

To rename a Favorites folder:

1. Right-click a folder on the Favorites page and choose **Rename** from the context menu.
2. Enter the new folder name and click **OK**.

You can also rename a folder when the Add to Favorites dialog is open, by clicking **Rename**.

To delete a Favorites folder or link:

- Click the **Delete** icon in the Favorites page toolbar, or right-click a node on the Favorites page and choose **Delete** from the context menu.

You can also delete a folder when the Add to Favorites dialog is open, by selecting the node and clicking **Delete**.

3.10.5 How to Customize the Online Help Display

You can customize some features of the Help Center window, as well as the navigators and topic viewers through the toolbars and context menu.

Use the **Keep on Top** icon to keep the Help Center in front of all open windows, including JDeveloper.

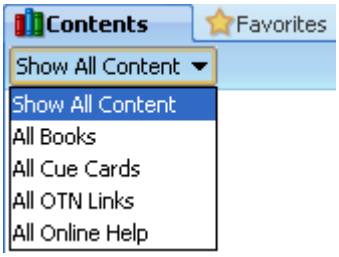
You can select the following types of help that you want to display from the **Navigators** drop down in the Help Center toolbar:

- **Contents** - Displays the table of contents for all installed online help topics and books.
- **Favorites** - Displays folders of user defined links for quick access to installed and external documentation.

Alternatively, you can right-click in the Help Center and choose a navigator from the **Configure Tabs** option on the context menu, to open navigators you previously closed.

You can also choose to view all help topics in the **Contents** navigator, or reduce what is displayed by selecting a single content type from the **Contents** drop down, as shown in the following figure.

Figure 3–1 Contents Dropdown List



Use the **Change Font Size** options in help topic viewer toolbar to increase or decrease the font size incrementally.

3.10.6 How to Open and Close Multiple Help Topics

When you navigate through topics in the help system, the topics open in new tabbed pages.

To see a help page that is already open, use one of the following ways:

- Select a tab at the top of the help topic window.
- Click the scroll buttons above the help topic viewer to scroll through all available tabs and select a tab.
- Click the **Tab List** button above the help topic viewer to display the list of all available pages and select a page.

When you open topics by clicking links within topics, the topics open within the same viewer. To cycle through those topics, click the **Forward** or **Back** icons in the help topic viewer toolbar. Note that you cannot navigate forward or back between different types of help viewer tabs; for example, the search results and help topic tabs. Use the scroll buttons instead.

To close one or more pages open in the help topic viewer:

- Right-click in the help topic viewer tab and choose from options on the context menu.

  You can close the page in front, all the pages, or all the pages except the page in front.

3.10.7 How to Print Help Topics

You can print help topics individually or by section.

To print an individual help topic:

1. Open a help topic in the help topic viewer.
2. In the help topic viewer toolbar, click the **Print** icon.

To print a topic grouping:

1. Click the **Contents** tab in the Help Center.
2. In the table of contents tree, select a topic folder.
3. Right-click and choose **Print Topic Subtree**.

The container topic and its children are printed. Topics listed as links are not printed.

3.11 Common Development Tools

This section provides an introduction to fundamental JDeveloper IDE functionality and concepts.

3.11.1 Application Overview

Use the Application Overview pages to guide you as you build a Fusion Web application, and to create files and objects and view the status of them.

3.11.1.1 Checklist

The Application Overview Checklist steps you through the building of a Fusion Web application, according to Oracle recommended best practices. The Checklist is displayed by default when a Fusion Web application is created, as part of the Application Overview pages.

The checklist optionally walks you through the entire process of configuring and building your application, with links to specific dialogs and wizards. Each step is also designed to teach you about the architecture, tools and resulting files using a combination of links to step-by-step instructions, relevant sections of the Developer's Guides, and descriptions of what happens in the IDE as a consequence of doing the work in a step.

Unlike a wizard, the Checklist itself is intended to provide a linear, but ultimately flexible and lightweight guide. You can follow the prescribed path in exact sequence, or explore tasks in a different preferred order. When using the Checklist, it suggests a best way to accomplish your goals, but you are not restricted by it. You can also close the Application Overview and work directly in the IDE, or work in both the IDE and Checklist interchangeably.

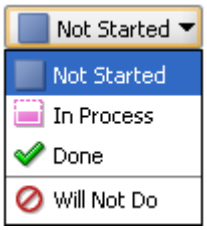
To use the Checklist:

- Expand a step and read the prerequisites and assumptions.

- Optionally click any of the documentation links.

- Click the button that takes you to the relevant area of the IDE.

- Use the status indicator dropdown to change the status as you work through tasks.



3.11.1.2 File Summary Pages

All files and artifacts that you create within JDeveloper appear in the Application Overview file summary pages, organized by object type. You can create new files and artifacts, and view them filtered by status and project. The following table describes the types of file summary pages.

Table 3–28 File Summary Pages

Page	Function
Status	Displays information about the object types available, using these status icons: ▪ **Error** ▪ **Warning** ▪ **Incomplete** ▪ **Advisory** ▪ **Ok** ▪ **Unchecked**
File	Displays the names of the objects. You can sort the objects in ascending or descending order by clicking the **Sort** icon in any of the column headings.
Project	Displays the project in which the file or object is located.

File Summary Pages Toolbar

The following table describes the icons in the File Summary Pages toolbar and their functions.

Table 3–29 Icons in the File Summary Pages Toolbar

Icon	Name	Function
	**New**	Creates new objects of the types listed, in the selected project. The context menu lists the files and objects associated with the technology that can be created in each project.
	**Edit**	Opens the selected file or object in its default edito.r
	**Delete**	Removes the selected file or object.

Table 3–29 (Cont.) Icons in the File Summary Pages Toolbar

Icon	Name	Function
	**Filter Status or Project**	Displays the list of all files of a particular status by selecting the status, as described above. By default, **Show All** is selected. If there is more than one project within the current application, use this list to select which project or projects you wish to be included in the file summary pages. You can choose: ■ all projects ■ a specific project from those available in the application

3.11.2 File List

Use the File List to search for and work on objects that you have created within an application. The rules, code assists, and metrics that are used to analyze Java code are specified by the Code Assist profile.

3.11.2.1 File List Tab Header

The following table describes the options available in the file list tab header.

Table 3–30 File List Tab Header Options

Option	Function
Look in	If you have more than one project within the current application, use this list to select which project or projects will be searched for objects. The list includes all projects in the current application, plus options to show all projects and a selection of projects (multiple projects). You can choose: ■ a specific project from those available in the application ■ All Projects ■ Multiple Projects, which opens the Select Projects dialog where you choose the projects from those available in the application.
Saved Searches	Initially contains **<New Search>**. After you have saved at least one search, also lists all saved searches. Selecting a saved search will display the search criteria for that search. The search results will show the results of the most recent search, even as you change between saved searches. To obtain new search results, click the **Search** button. Saving a search is one of the actions available from the **More Actions** button.
Show History	Opens the Recent Searches dialog, through which you can return to a recent search. The search criteria of the selected search is shown, while the search results remain as they were for the most recent search. To obtain new search results, click the **Search** button.

3.11.2.2 Search Criteria Area

The following table describes the features available in the search criteria area.

Table 3–31 Features in the Search Criteria Area

Option	Function
Search criteria input line(s)	Initially contains a single input line for search criteria. You can add further lines by clicking the **Add** icon at the end of the line. You can remove lines by clicking the **Delete** icon at the end of the line that you want to remove. By default, the first field in the line contains File Name: you can change this to File Extension, Date Modified, Status, or Category. The second field contains the options available for extending the entry in the first field. The third field contains a list of all object types that can be searched for.

Table 3–31 (Cont.) Features in the Search Criteria Area

Option	Function
Match options	Choose between Match All and Match Any to determine the scope of the search.
Search	Click to begin a search based on the search criteria currently shown.
More Actions	Click to reveal the following menu of options for use with named searches: ■ **Save** - Saves the current search criteria with the name currently in the Saved Searches box (even if the name is <New Search>). ■ **Save As** - Opens the Save As dialog, through which you can save the current search criteria as a new named search. ■ **Restore** - Restores a deleted named search if used immediately after the **Clear** option on this menu has been used. ■ **Clear** - Clears the search criteria for this named search. You can restore the criteria to this named search by immediately selecting the **Restore** option on this menu. ■ **Delete** - After confirmation, deletes the current named search.

3.11.2.3 Search Results Table

The following table describes the options available in the Search Results table.

Table 3–32 Options Available in the Search Results Table

Option	Function
Results summary	Shows the number of files that match the search criteria, and the date and time that the search was completed.
Refresh	Reruns the search with the current search criteria.
Customize table	Opens a menu from which you can choose the columns that will be displayed in the results table. Also contains a Select Columns option, which opens the Customize Table dialog, through which you can choose which columns to display and the order in which they are displayed in the results table. The columns that are shown by default are Status, File, Project, and Date Modified, in that order. Other columns that you can choose to show are Application and Category.
Table headings	You can change the order of the columns by grabbing a table heading and moving it horizontally. You can change whether objects are shown in ascending or descending order within the columns by clicking a heading to give it focus, then clicking again to change the sort order. The sort icon (or) in the table heading will change as appropriate.
Objects list	Lists all the objects returned by the search. You can initiate actions for an object by selecting the name, right-clicking, and selecting from the context menu.

3.11.3 Compare Window

The Compare Window allows you to view the differences between two files or two directories.

You might want to do this when deciding whether to check in a particular file to a source control system, especially if doing so will overwrite a file whose contents you are unfamiliar with. The Compare Window is integrated with the Application Overview and the Application Navigator, and with the Subversion source control system.

3.11.3.1 Toolbar

The following table describes the icons in the Compare Window toolbar and their functions.

Table 3–33 Compare Window Toolbar Icons

Icon	Name	Function
	**Go to First Difference**	Click to move the cursor to the first difference.
	**Go to Previous Difference**	Click to move the cursor to the previous difference.
	**Go to Next Difference**	Click to move the cursor to the next difference.
	**Go to Last Difference**	Click to move the cursor to the last difference.
	**Generate Patch**	Click to open the Generate Patch dialog, where you can generate a patch containing changes that have been made to the files.

3.11.3.2 Source and Target Areas

The title bar of each area identifies the file that contains the differences. The versions are aligned line by line. Lines with differences are highlighted using shaded boxes, joined as appropriate.

3.11.4 Application Navigator

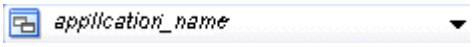
The Application Navigator allows you to manage the contents and associated resources of an application.

3.11.4.1 Application Navigator Toolbar

This section describes the features available from the Application Navigator toolbar.

Main dropdown list

Use the main dropdown list, displayed in the figure below, to create a new application, open an existing application, or choose from the list of open applications. Use the context menu to choose from the list of application level actions available.



Application menu

Use the application menu, displayed in the figure below, to choose from a list of actions available.

The following table describes the options available from the **Application Menu**.

Table 3–34 Application Menu Options

Menu Option	Function
New Project	Opens the New Gallery ready for you to select the type of project to create.

Table 3–34 (Cont.) Application Menu Options

Menu Option	Function
New (Ctrl+N)	Opens the New Gallery. Only those items available to be created from an application are available
Open Project	Opens the Open Project dialog, where you navigate to a project that you want to open in this application.
Close Application	Closes the current application.
Delete Application	Deletes the application control file (.jws) from disk.
Rename Application	Opens the Rename dialog where you can change the name of the current application.
Find Application Files	Opens the File List, where you can search for specific files.
Show Overview	Opens the Application Overview which is the home for all files you can create in this application.
Filter Application	Opens the Manage Working Sets dialog where you can specify the files to include or exclude from being listed in the Application Navigator.
Secure	Secures your application resources.
Deploy	Allows you to choose from the deployment profiles defined for the application.
Application Properties	Opens the Application Properties dialog where you can set various properties for the application.

3.11.4.2 Application Operations

You can several application operations from the Application Navigator. These include:

- In the initial view, before any application content is shown, select the New Application link to create a new application or select the Open Application link to open an existing application.
- Open any currently closed navigator, or bring a currently open navigator to the foreground, using **View** > *navigator-name*.
- Move, size, float, minimize, maximize, restore or close the Application Navigator using the context menu available by right-clicking its tab or by pressing Alt+Minus.
- Change the application shown in the navigator by choosing one from the main dropdown list or, if the one you want is not shown, by choosing **Open Application**.
- Create a new application by choosing New Application from the dropdown list.
- Open the context menu for the application by right-clicking the application, or by clicking the **Application Menu** icon (to the right of the application name).

3.11.4.3 Projects Panel Operations

You can perform the following operations from the projects panel of the Application Navigator:

- View the project properties by clicking the **Project Properties** icon.
- Refresh the project contents by clicking the **Refresh** icon.

- Filter the project content that you work with by selecting options from the **Working Sets** dropdown menu.
- Change what is shown in the navigator by selecting options from the **Navigator Display Options** dropdown menu.
- Obtain a context-sensitive menu of commands for any node by right-clicking it.
- Display the structure of an object in the Structure window by clicking the object's name.
- Open an object in its default editor, or bring the default editor into focus, by double-clicking the object's name.
- Rename a file using **File > Rename**.
- Relocate a file using **File > Save As**.
- Search for items visible in the panel by putting the focus anywhere inside it and typing a search string for the object you are looking for. (Precede with an asterisk to search for instances of names containing the search string.)
- Close or open the panel by clicking its bar.
- Remove the panel from view by opening its dropdown menu (panel bar, far right) and choosing **Minimize**. Restore it by clicking the three dots at the very bottom of the navigator and then clicking Projects.

3.11.4.4 Application Resources Panel Operations

You can perform the following operations in the Application Resources panel:

- Close or open the panel by clicking its bar.
- Change the area used by the panel by grabbing its bar and moving it up or down.
- Remove the panel from view by opening its dropdown menu (panel bar, far right) and choosing **Minimize**. Restore it by clicking the three dots at the very bottom of the Application Navigator and then clicking Application Resources.
- Obtain a context-sensitive menu of commands for any node by right-clicking it.
- Display the structure of an object in the Structure window by clicking its name.
- Open an object in its default editor, or bring the default editor into focus, by double-clicking the object's name.
- Search for items visible in the panel by putting the focus anywhere inside it and typing a search string for the object you are looking for. (Precede with an asterisk to search for instances of names containing the search string.)

3.11.4.5 Data Controls Panel Operations

You can perform the following operations in the Data Controls panel:

- Close or open the panel by clicking its bar.
- Change the area used by the panel by grabbing its bar and moving it up or down.
- Remove the panel from view by opening its dropdown menu (panel bar, far right) and choosing **Minimize**. Restore it by clicking the three dots at the very bottom of the Application Navigator and then clicking **Data Controls**.
- Obtain a context-sensitive menu of commands for any node by right-clicking it.
- Edit the definition of a data control by opening its context menu and choosing **Edit Definition**.

- Search for items visible in the panel by putting the focus anywhere inside it and typing a search string for the object you are looking for. (Precede with an asterisk to search for instances of names containing the search string.)

3.11.4.6 Recently Opened Files Panel Operations

You can perform the following operations in the Recently Opened Files panel:

- Close or open the panel by clicking its bar.
- Change the area used by the panel by grabbing its bar and moving it up or down.
- Remove the panel from view by opening its dropdown menu (panel bar, far right) and choosing Minimize. Restore it by clicking the three dots at the very bottom of the Application Navigator and then clicking Recently Opened Files.
- Open an object in its default editor, or bring the default editor into focus, by double-clicking the object's name.
- Search for items visible in the panel by putting the focus anywhere inside it and typing a search string for the object you are looking for. (Precede with an asterisk to search for instances of names containing the search string.)

3.11.5 Application Server Navigator

The Application Server Navigator allows you to manage connections to application servers. It is integrated with the Resource Palette.

When you create an application server connection in the Application Server Navigator it is available in the Resource Palette. Similarly, when you create an application server connection in the Resource Palette, it is available in the Application Server Navigator.

From the context menu of the Application Server Navigator, you can:

- Create a new connection to an application server by choosing New Application Server from the context menu of the Application Servers node.
- Import connections by clicking Import from the context menu of the Application Servers node.
- Export connections by clicking Export from the context menu of the Application Servers node.
- Edit the properties of an existing application server connection by choosing Properties from the context menu of the connection.

From the context menu of IntegratedWebLogicServer, you can:

- Start the Integrated WebLogic Server.
- Start the Integrated WebLogic Server in debug mode.
- Create the Default Domain. When you first start the Application Server Navigator, the only node is IntegratedWebLogicServer (domain unconfigured). Before you can work with Integrated WebLogic Server, you must create a default domain. If you are creating the default domain for the first time, you must enter an administrator password for the new domain.
- Update the Default Domain.
- Configure a log to help diagnose problems.
- Launch the Admin Console for:
  - Integrated WebLogic Server.

– Oracle WebLogic Server.

The following table describes the icons in the Application Server Navigator toolbar:

Table 3–35 Application Server Navigator Toolbar Icons

Icon	Name	Function
	**Refresh**	Click to refresh the contents of the selected application server connection.
	**Delete**	Click to delete the selected application server connection.

3.11.6 Structure Window

The Structure window offers a structural view of the data in the document currently selected in the active window of those windows that participate in providing structure: the diagrams, the navigators, the editors and viewers, and the Property Inspector.

Depending on the document currently open, the Structure Window enables you to view data in two modes:

- **Source** - displays the code structure of the file currently open in the editor. Applicable to technologies that allow code editing. For example, this tab will not be available when a diagram is open for editing.
- **Design** - displays the UI structure of the file currently open in the editor.

In the Structure window, you can view the document or diagram data in a variety of ways. The structures available for display are based upon document or diagram type. For a Java file, you can view code structure, UI structure, or UI model data. For an XML file, you can view XML structure, design structure, or UI model data.

The Structure window is dynamic, tracking always the current selection of the active window (unless you freeze the window's contents on a particular view), as is pertinent to the currently active editor. When the current selection is a node in the navigator, the default editor is assumed. To change the view on the structure for the current selection, select a different structure tab.

The windows that participate in providing structure also follow selections made in the Structure window. Double-clicking the node for a method in the Structure window, for instance, makes the source editor the active view and takes you directly to the definition for that method.

You can open multiple instances of the Structure window, freezing the contents of any number of them, in order to compare the structures of different files. You can also switch structure views without changing editors.

Diagram objects (such as UML elements) listed in the Structure window can be dragged from the window and dropped directly onto diagrams.

3.11.6.1 Structure Window Toolbar

The following table describes the icons in the Structure Window toolbar and their functions:

Table 3–36 Structure Window Toolbar Icons

Icon	Name	Function
	**Freeze**	Click to freeze the Structure window on the current view. A window that has been frozen does not track the active selection in the active window.
	**New View**	Click to open a new instance of the Structure window. The new view appears as a tabbed page in the same window.

3.11.6.2 Structure Window Views

The Structure window view depends upon the document type of the current selection in the active window. Each view offers different options for viewing and sorting the structure of your files based on file type.

The following table describes the Structure Window views.

Table 3–37 Structure Window Views

View	Description
ADF Business Components View	When you select any ADF business component in one of the navigators, the Structure window offers a structured view of the component's files, attributes, and other properties.
Cascading Style Sheet View	This view allows you to select and group CSS elements for easy editing. When a CSS file is open for editing, CSS selectors in the file are displayed in the Structure window as one of three types: Element, Class, and ID.
Java View	This view displays the code as well as design structure of the Java file currently being edited. Additionally, you can specify several display preferences to view structural data.
JSP/HTML View	This view displays the code structure and UI bindings for the JSP/HTML file that is currently selected.
Struts View	The Struts view shows the hierarchy of elements and attributes for the Struts configuration file currently selected in the active navigator or editor.
TopLink View	The TopLink view displays detailed information about the TopLink element selected in Application Navigator or TopLink editor, including descriptors, sessions, and mappings.
UML View	The UML view displays the behavior, interaction, and code structure in UML-based diagrams such as Activity Diagrams, Class Diagrams, and Use Case Diagrams.
Diagram View	When a diagram is open for editing, the Diagram view displays the components that have been added to the diagram. You can select an element in the Structure Window's diagram view and locate it in the diagram

3.11.7 Application Navigator - Data Controls Panel

Use to view the data controls created to represent an application's business services and to create databound UI components by dragging and dropping the control panel objects onto an open web page or ADF Swing panel.

Note: The Data Controls panel may appear empty if no data controls have been created for or imported into the application.

The panel displays objects to which your UI components can be bound, including data collections, attributes, and methods that the business services developer exposed through the Oracle ADF data control, as well as specific, built-in operations that are generic to all data collections.

When you drag an object from the Data Controls panel onto a page, the context menu displays the UI components you can create for that specific object. Creating components this way means that they will automatically be databound to the dropped object.

After inserting a databound UI component into the displayed web page or Java panel, you can view the Oracle ADF data binding:

- In the code view of a web page, where data binding objects appear in expressions that get evaluated at runtime using the expression language features of the JSTL tag library.
- In the code view of an ADF Swing panel or form, where the `setModel()` method call on the UI component initializes the data binding object and accesses the Oracle ADF binding context (specified by the `setBindingContext()` method call on the panel).
- In the associated page definition file. The page definition file defines the bindings created for the page, panel, or form.

Data Controls panel toolbar

The following table describes the icons in the Data Controls panel toolbar and their functions:

Table 3–38 Data Controls Panel Toolbar Icons

Icon	Name	Function
	**Refresh Panel**	Click to reload the panel if the underlying business components have changed.
	**Filter Panel**	Click to enter search criteria to find a specific item in the panel.

3.11.8 Log Window

The Log window displays tabbed windows for specific feedback from various components of the IDE.

The Log window displays information on:

- Compiler. The compiler reports error messages that you can double-click to navigate directly to the correct line in the source file referenced.
- Apache Ant. When you build your project using Apache Ant, the Log Window displays relevant build information.
- Debugger
- Audit
- Profiler

To bring up the context menu for the contents of the Log window, right-click within the window. To bring up the context menu for the Log window as window, right-click on the tab.

From the context menu for the general Log window, you can:

- Copy the contents of the window
- Select all data within the window
- Wrap the text in the window
- Clear the contents of the window
- Save the contents of the window to another format
- Close the window

Other actions may be available within the tabbed sections generated by specific processes.

From the context menu for the window itself, you can:

- Close the window
- Close all other tabs but for the currently selected tab
- Close all tabs within the window

3.11.9 Status Window

The Status Window is one of the JDeveloper features that helps you to audit your code. It displays audit violations in the document selected in the File List and provides information to help you resolve the issues.

The Code Assist audit profile determines the audit violations that are reported.

Status Window Toolbar

You can choose the items you want to view using the icons in the Status window toolbar.

The following table describes the icons in the toolbar and their functions:

Table 3–39 Status Window Toolbar Icons

Icon	Name	Function
	**Show Error Issues**	Toggle to show just the number of errors in the selected file, or to list the errors in the file.
	**Show Warning Issues**	Toggle to show just the number of warnings in the selected file, or to list the warnings in the file.
	**Show Incomplete Issues**	Toggle to show just the number of incomplete issues in the selected file, or to list the incomplete issues in the file.
	**Show Advisory Issues**	Toggle to show just the number of advisory issues in the selected file, or to list the advisory issues in the file.
	**Fixes**	Select one of the issues in the list, and click Fixes. A suggested fix is displayed, for example: **Add missing Javadoc tags**.

3.11.10 Tasks Window

Use this dockable window to record tasks associated with applications, projects and files.

If you are working in a Java Class source file, a task will automatically be created whenever you type `// TODO` (in other words, when you create a comment and use the source tag recognized by JDeveloper).

While you are using the Tasks window, these features are available:

- Sort the information by clicking the column headings.
- Show or hide columns by opening the context menu for any heading and choosing from the list. Alternatively, you can choose **Show/Hide Columns** from the context menu of any task.
- Add a task by choosing **Add Task** from the context menu of any task.
- Edit an existing task by choosing **Edit Task** from the context menu of the task.
- Delete a task by choosing **Remove Task** from the context menu of the task.
- Delete completed tasks by choosing **Remove Completed Tasks** from the context menu of any task.
- Open the file that the task refers to by choosing **Go to Source** from the context menu of the task.

Tasks Window Toolbar

The toolbar enables you to manage the tasks displayed in the Tasks window. The following table describes the icons in the Tasks Window toolbar and their functions.

Table 3–40 Tasks Window Toolbar Icons

Icon	Name	Function
	**Current Application**	Choose to define the current application as the scope of the tasks displayed.
	**Current Project**	Choose to define the current project as the scope of the tasks displayed.
	**Current File**	Choose to define the current file as the scope of the tasks displayed.
	**Add task**	Click to create a new task (independent of source file comments).
	**Edit task**	Click to edit the highlighted task.
	**Delete task**	Click to remove highlighted task.
	**Filter**	Click to open the Filter Tasks dialog, where you can set up filters to determine which tasks are and are not shown.

3.12 Adding External Tools to JDeveloper

External tools are custom JDeveloper menu items and toolbar buttons that launch applications installed on your system, applications that are not packaged as part of JDeveloper.

To find all external programs that JDeveloper is preconfigured to support:

1. From the main menu, choose **Tools > External Tools**.
2. In the External Tools dialog, click **Find Tools**.

To add access to an external program from JDeveloper:

1. From the main menu, choose **Tools > External Tools**.
2. In the External Tools dialog, click **New**. Follow the instructions in the wizard.

To change how an external program appears, or remove access to an external program from JDeveloper:

1. From the main menu, choose **Tools > External Tools**.
2. In the External Tools dialog, click **Edit** or **Delete**. If you are editing the options, display, integration or availability of an external tool from JDeveloper, select the corresponding tab and change the values. Click **Help** for help choosing valid values.
3. Click **OK**. Your changes are reflected immediately.

Part II

Developing Applications with Oracle JDeveloper

This part describes how to develop applications with Oracle JDeveloper. You can find information on how to effectively build, test, run, and deploy applications.

This part contains the following chapters:

- Chapter 4, "Getting Started with Developing Applications with Oracle JDeveloper"

  This chapter provides an overview of the features for developing applications available in JDeveloper.

- Chapter 5, "Working with Applications and Projects"

  This chapter describes how you can effectively work with applications and projects in the Oracle JDeveloper IDE.

- Chapter 6, "Versioning Applications with Source Control"

  This chapter describes the version control features available in JDeveloper.

- Chapter 7, "Building, Running and Debugging Applications"

  This chapter provides an overview of the building, running, and debugging features in JDeveloper.

- Chapter 8, "Auditing and Profiling Applications"

  This chapter provides an overview of the auditing, profiling, and testing features in JDeveloper.

- Chapter 9, "Deploying Applications"

  This chapter describes how to deploy applications in JDeveloper.

4

Getting Started with Developing Applications with Oracle JDeveloper

This chapter provides an overview of the features for developing applications available in JDeveloper.

This chapter includes the following sections:

4.1 About Developing Applications with Oracle JDeveloper

JDeveloper provides several tools and features for developing applications. You can use these features to effectively build, test, run, and deploy your application. These features include:

- Navigators, windows, and palettes for managing and working with different object types and resources associated with applications, projects, and files.
- Several visual and code editing tools to facilitate the task of creating different types of source documents. The editors are integrated with other tools in the IDE such as navigators and palettes, thus drag and drop operations and simultaneous, automatic updates among the various integrated tools are supported.
- Tools to simplify the task of testing and analyzing source code, processes, and application modules or packages.

5

Working with Applications and Projects

This chapter describes how you can effectively work with applications and projects in the Oracle JDeveloper IDE.

This chapter includes the following sections:

5.1 About Working with Applications and Projects

The application is the highest level in the control structure. It is a view of all the objects you need while you are working. An application keeps track of all your projects while you develop programs.

A project is a logical container for a set of files that define a JDeveloper program or portion of a program. A project might contain files representing different tiers of a multi-tier application, for instance, or different subsystems of a complex application. These files can reside in any directory and still be contained within a single project.

You can remove application and project control files from the IDE without deleting them from the disk. (This is not true for other types of file, which will be deleted from the disk at the time that they are removed from the IDE.)

JDeveloper can recognize many different file types, displaying each in its appropriate viewer or editor when you double-click the file.

When adding a project to an application, you can choose to:

- Create a new project, with specific objects and attributes you define.
- Create a new empty project, which inherits default project properties.
- Open an existing set of files from outside JDeveloper into a new project.

As soon as you create a new project or open an existing one, it is added to the application selected.

Projects control their files lists directly through the directory. Applications and packages also define where and how the files within a project are stored.

Note: The same object (physical file) can appear in more than one project. This means that any actions carried out on the object in one project will show up in the other project (although some effects will become apparent only after the project is compiled). For packages, two or more projects should not share a package unless, first, they also share a source path used to generate the package and, secondly, the package is already compiled and will never be changed.

5.2 Creating Applications and Projects

New applications and projects are managed from the Application Navigator.

5.2.1 How to Create an Application

This section describes how to create a custom JDeveloper application and a project within it.

To create a new application:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **General**, select **Applications**.
3. In the **Items** list, double-click the application type you want to create.
4. In the Create Application dialog, enter application details like the name and directory. For help with the wizard, press F1.
5. Click **Next** to open the Project Name page, where you can optionally provide details for your project.
6. When you are done, click **Finish**.

5.2.2 How to Create a Custom Application

This section describes how to create a custom application that includes a single project that can be customized to include any features.

To create a custom application:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the **Categories** tree, under **General**, select **Applications**.
3. In the **Items** list, double-click **Custom Application**. The Create Custom Application wizard opens.
4. In the Create Custom Application dialog, enter application details like the name and directory. For help with the wizard, press F1.
5. Click **Next** to open the Project Name page, where you can optionally provide details for your project.
6. When you are done, click **Finish**.

5.2.3 How to Create a New Project

In JDeveloper, you use the Application Navigator to keep track of the projects (collections of related files or components) you use while developing your application. You can also create a new project based on existing source.

5.2.3.1 How to Create a New Project

You can create a new project from existing source or populated with new objects, or you can create a new empty project.

All projects inherit the settings specified in the Default Project Properties dialog. As soon as you create the project, it is added to the active application.

To create a new project from existing source or with new objects:

1. In the Application Navigator, select the application within which the project will appear.
2. Click the **Application Menu** icon, and select **New Project** to open the Projects page of the New Gallery.
3. In the Items list, double-click the project type you want to create.
4. Complete the Create Project wizard, and click **Finish**. For help on the wizard, press F1.

The new project appears in the navigator. It inherits whatever default properties you've already set. To alter project properties for this project, either double-click the filename or right-click and choose **Project Properties**.

5.2.3.2 How to Create a New Custom Project

A custom project can be customized to include any feature. All projects inherit the settings specified in the Default Project Properties dialog. As soon as you create the project, it is added to the active application.

To create a new custom project and add it to the active application:

1. In the Application Navigator, open the application that will contain the new project.
2. Click the **Application Menu** icon, and select **New Project** to open the Projects page of the New Gallery.
3. Under **Items**, select **Custom Project**.
4. Click **OK**.

5.3 Managing Applications and Projects

JDeveloper provides several features to effectively manage your applications and projects.

5.3.1 How to Open an Existing Application or Project

You can create new applications and projects from scratch or open existing ones. As soon as you create or import the application, it is added to the Applications node in the Application Navigator. As soon as you create or import a project, it is added to the selected application.

To open an existing application and add it to the Application Navigator:

1. In the Application Navigator, select **Open Application** from the dropdown list.
2. Navigate to the application file and select it.

   Be sure that the file type field either specifies `.jws` files or allows all types to be displayed.

3. Click **Open**.

   The application is added to the list of applications in the navigator.

To open an existing project and add it to an application:

1. In the Application Navigator, select the application to which the project will be added.
2. From the main menu, choose **File > Open**.
3. Navigate to the project file and select it.

   Be sure that the file type field either specifies `.jpr` files or allows all types to be displayed.
4. Click **Open**.

   The project is added to the active application.

5.3.2 How to Import Existing Source Files into JDeveloper

You can create new files of various types from scratch or open existing ones. When opening existing files, you can import them, along with their file structure, into an existing project or build a completely new project around them.

Alternatively, you can add source files to projects you already have.

5.3.2.1 Importing Existing Files into a New JDeveloper Project

You can import existing files of any type into JDeveloper, creating a new project as you do so.

To open existing files and import them into a new JDeveloper project:

1. In the Application Navigator, select or create the application to which the new project will be added.
2. With the application selected choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **General** and select **Projects**.
4. In the **Items** list, double-click **Project from Existing Source**.
5. On the **Location** page of the Project from Existing Source wizard, enter a name for the new `.jpr` file or accept the default.

   For more information on this or subsequent wizard pages, press F1 or click **Help** from within the wizard.

   Alternatively, you can select **File > Import**, and choose either **Java Source** or **Source into New Project**.
6. Accept the default directory path, enter a new path, or click **Browse** to navigate to one.
7. Click **Next**.
8. On the Specify Source page, in the **Java Content** area, click **Add** to open the Choose Directory dialog.
9. In the dialog, navigate to the directory containing the files you wish to add. Click **Select** to close the dialog and display the directory in the wizard.
10. When you have finished adding directories, you can apply file or directory filters. To apply filters, click **Add** next to the Included tab.

11. When the import list is complete, optionally select **Copy Files to Project** directory to clone the selected files in your project rather than simply pointing to the original source.
12. Define a default output directory and default package.
13. Click **Finish**.

    The new project appears under the selected application node, populated with the imported files.

You can fine tune your project directories structure, for example to point to resource directories, in the Project Properties dialog.

5.3.2.2 How to Import a WAR File into a New JDeveloper Project

You can import a WAR file into JDeveloper, creating at the same time a new project to contain its extracted contents.

To open a WAR file and import it into a new JDeveloper project:

1. In the Application Navigator, select or create the application to which the new project will be added.
2. With the application selected choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **General** and select **Projects**.
4. In the **Items** list, double-click **Project from WAR File**.
5. Complete the Create Project from WAR File wizard.

   For information when using this wizard, press F1.

The wizard analyzes the WAR file and extracts its contents.

The new project appears under the selected application node, populated with the imported files.

5.3.2.3 Importing an EAR File into a New JDeveloper Application

When you import an EAR file, JDeveloper will always create a new application and populate it with projects based on the EAR modules extracted. You cannot add the contents of an EAR file to an existing application or project.

You should not use this procedure to import an EAR file that you simply wish to deploy using JDeveloper. To do this, create a new application and project, then copy your EAR file into the project directory (or add its location to the project's content). The EAR file will then appear in the Application Navigator under the project's Application Sources node. From here, you can deploy the file by right-clicking it and choosing **Deploy to**.

To open an EAR file and import it into a new JDeveloper application:

1. From the main menu, choose **File > Import** and double-click **EAR File**.

   The Import EAR File wizard is not sensitive to context, so you need not select anything specific in the navigator first.
2. Complete the Import EAR File wizard.

   On the Finish page, the contents of the final application are displayed.
3. Click **Finish** to accept the listing and create the application.

The new application appears in the navigator, populated with projects based on the imported modules.

5.3.3 How to Import Files into a Project

You can create new files of various types from scratch or open existing ones. When opening existing files, you can import them, along with their file structure, into an existing project or build a completely new project around them.

You can also create new projects from existing source.

5.3.3.1 How to Import Files into a Project

You can import existing files of various types into a project you've already created, while maintaining their original file structure.

Note: You can use the Import Existing Sources wizard to add `.zip` or `.jar` files to projects. You cannot use it to add `.war` or `.ear` files. A .war file requires the Import WAR File wizard to property extract its contents into the project. An EAR file requires the Import EAR File wizard, which extracts its contents into a new application.

To open an existing file and add it to a project using the Import Existing Sources wizard:

1. In the Application Navigator, select the project to which the file will be added.
2. From the main menu, choose **File > Import**.
3. In the Import dialog, double-click **Existing Sources**.
4. On the **Add Source Files and Directories** page of the Import Existing Sources wizard, click **Add** to open the Select Files or Directories dialog.

   For more information on this or subsequent wizard pages, press F1 or click **Help** from within the wizard.
5. In the dialog, navigate to the directory containing the files you wish to add, or to the individual files themselves, and click **Open** to close the dialog and display the files in the wizard.

   You can return to this dialog as many times as you want, adding as many individual files or directories as you would like, by clicking **Add** again once you have returned to the wizard.
6. When you have finished adding files or directories, and have returned to the wizard, you can refine your list by selecting and deselecting individual files or by applying filters. To apply filters, click **File Filter** or **Directory Filter**.
7. When your import list is complete, optionally select **Copy Files to Project** directory to clone the selected files in your project rather than simply pointing to the original source. If you select this option, accept the default src directory, enter a new directory, or click **Browse** to navigate to one.
8. Click **Next**.
9. On the Finish page, review the listing of new project files. To accept this list, click **Finish**.

   The files are now added to the selected project.

5.3.4 How to Manage Folders and Java Packages in a Project

JDeveloper enables you to create custom folders or Java packages within your project to better organize your project files.

To create a folder or Java package:

1. In the Application Navigator, select the project or folder within which you want to create the custom folder.
2. On the **File** menu, select **New**.
3. In the New Gallery, under **Categories**, select **General**.
4. Under **Items**, select **Folder** to create a new folder. To create a Java Package, under **Items**, select **Java Package**.
5. In the Create Folder or Create Java Package dialog, specify the name of the folder or Java package, and the directory you want to create it in.

To delete a folder or Java package:

1. Select the folder or Java package that you want to delete.
2. On the **File** menu, select **Delete**.
3. On the Confirm Delete Folder dialog, confirm the deletion of the folder or Java package. Click **Show Folder Files** to see the files contained in the folder or Java package.

5.3.5 How to Manage Working Sets

Working sets allow you to configure the navigator to show you a subset of files from your project. This is particularly useful when working with large projects. Before you define your own working sets the only one available is Default, and it is a working set which includes all the files in the current application.

You can run and debug a working set in just the same way as you run and debug a project. This allows you to work on just a subset of a large application, for example a Java EE application, without affecting the entire application or incurring a performance hit.

You can define a working set by selecting from files or containers in the Application Navigator, or by providing include and exclude filter patterns through the Manage Working Sets dialog.

To group objects in the Application Navigator into a working set:

1. In the Application Navigator, select the objects that you want to include in a new working set.
2. In the Application Navigator, click the **Working Sets** icon and select **New from Selection**.

   This opens a Save As dialog. For more information at any time, press F1 or click **Help** from within the Save As dialog.
3. Enter a name for the working set, then click **OK**.

To create a working set by defining file and directory filters:

1. In the Application Navigator, click the **Working Sets** icon and select **Manage Working Sets**.

This opens the Working Sets dialog. Use the tree on the left to select the projects to include. In the right panel, select which files in the current project to include. For more information at any time, press F1 or click **Help** from within the Working Sets dialog.

2. Click **Save As** to save the working set.

To create a working set from the results of a search in the Log window:

1. In the Log window, right-click and choose **Save as Working Set** from the context menu.
2. In the Create Working Set dialog, enter a name for the working set.

To see which working set you are currently using:

- In the Application Navigator, hover the mouse over the **Working Sets** icon. The name of the current working set is displayed as a tooltip. Alternatively, click the **Working Sets** icon to bring up a menu in which the active working set is checked.

To change the active working set:

- In the Application Navigator, click the **Working Sets** icon and select the working set you want to open.

  Files not belonging to the working set are removed from view.

To edit files and projects in a working set:

1. In the Application Navigator, click the **Working Sets** icon and select **Manage Working Sets**.

   This opens the Working Sets dialog. For more information at any time, press F1 or click Help from within the Working Sets dialog.
2. Select the working set that you want to change from the Working Set drop-down list.
3. Make the changes as required.

To restore the view in the Application Navigator to show all files:

- In the Application Navigator, click the **Working Sets** icon and select **(All Files)**.

To run and debug a working set:

1. Ensure that you are using the working set you want to run or debug. This should include the projects that represent the Java EE modules (Web applications, EJB modules) that you are working on and any dependencies.

   Be aware that any projects that are explicit dependencies (in the Dependencies page of the Project Properties dialog) will be included even if they are excluded from the working set.
2. Choose **Run > Use Current Working Set (Java EE Only)**. Now, when you open the context menu of the source editor or a file or project in the Application Navigator, the run and debug options have "Working Set" as part of the name. For example, **Run (Working Set)** or **Debug (Working Set)**.

5.3.6 How to Browse Files in JDeveloper Without Adding Them to a Project

Sometimes, you may not want to add files directly to a project, but yet have them handy for browsing. You can bring files into the JDeveloper IDE, without adding them to a project.

To open files in JDeveloper without adding them to a project:

1. From the main menu, choose **File > Open**.

   As you are only going to view the files, it doesn't matter which node in the Application Navigator is currently selected.

2. Navigate to the file or files to be opened. Be sure that the file type field either specifies the appropriate file type or allows all types to be displayed

3. Select the file or files. You can select as many files, or directories, from the list as you would like.

   Archive files appear twice: once as a virtual directory and then again as a file. If you will be opening an archive file, select its appearance in the list as a directory.

4. With your selection made, click **Open**.

5.3.7 How to View an Archive

You can easily inspect the contents of any archive, after first opening the archived file in JDeveloper. You can add the contents of an archive to an existing or new JDeveloper project.

To open an archive in JDeveloper and view its contents:

1. From the main menu, choose **File > Open**.

   As you are only going to view the contents of the archive, it doesn't matter which node in the Application Navigator is currently selected.

2. Navigate to the directory containing the archive. Archive files appear twice: once as a virtual directory and then again as a file.

   If you do not see the archive files, double-check that all file types are being displayed.

3. Select the second appearance of the archive, the archive as a file, and click **Open**.

5.3.8 How to View an Image File in JDeveloper

You can easily view any `.gif`, `.jpg`, `.jpeg`, or `.png` file from within JDeveloper.

To open and view an image in JDeveloper:

1. From the main menu, choose **File > Open**.

   As you are only going to view the image, it doesn't matter which node in the Application Navigator is currently selected.

2. Navigate to the image or images to be opened. Be sure that the file type field either specifies all file types or the image types.

3. Select the image.

4. With your selection made, click **Open**.

   The image is displayed in the main working area of JDeveloper.

To view an image already imported into JDeveloper:

1. In the Application Navigator, select the image file.
2. Double-click the file, or right-click and choose **Open**.

5.3.9 How to Set Default Project Properties

You can set the project properties for all subsequently created projects or fine-tune the properties for any individual project.

When you set project properties for an individual project, you override the default values for that project alone.

To view or change the default settings for a project:

1. From the main menu, choose **Application > Default Project Properties**.
2. In the Default Project Properties dialog, select the appropriate category.
3. View or set the various properties as desired.
4. When finished, click **OK**.

The procedures you follow for setting default project properties are identical to those for setting properties for individual projects — with the exception that, as you are in default properties, you do not need to first select an individual project. Note that some project properties cannot be set from the Default Project Properties dialog.

5.3.10 How to Set Properties for Individual Projects

You can set the project properties for all subsequent projects, or fine-tune the properties for any individual project. When you set project properties for an individual project, you override the default values for that project alone.

Additional project properties are also available, based upon specific tasks such as compiling, or debugging.

To view or change the current output path for an individual project:

1. In the Application Navigator, select the appropriate project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.

   The Project Properties dialog opens with the input paths displayed on the last page that you viewed.
3. On the Project Source Paths page, change the output directory as desired by typing in the new values or by clicking **Browse**.
4. When finished, click **OK**.

5.3.10.1 How to Include Libraries in a Project

When you include libraries in a project, the source paths defined for those libraries automatically become part of the project's classpath.

To view the current libraries for an individual project:

1. In the Application Navigator, select the appropriate project.
2. From the context menu, choose **Project Properties**.

3. Select the **Libraries and Classpath** node. The libraries currently included in the project are shown in the **Classpath Entries** list

To add an existing library to a project:

1. With the project selected in the Application Navigator, open the Project Properties dialog.
2. Select the **Libraries and Classpath** node
3. On the Libraries and Classpath page, click **Add Library**.
4. Locate the required library in the selection tree and click **OK**.

To create a new library and add it to a project:

1. With the project selected in the Application Navigator, open the Project Properties dialog.
2. Select the **Libraries and Classpath** node.
3. On the Libraries and Classpath page, click **Add Library**.
4. On the Add Library dialog, click **New**.
5. In the Create Library dialog, enter a name for the new library and select its location.
6. For each path type, click **Add Entry** or **Add URL** as appropriate. To remove a path, or correct an addition, click **Remove**. To rearrange the order of entries, use the reordering buttons to the right of the display area.
7. Once you have clicked either **Add Entry** or **Add URL**, in the resulting selection dialog enter the filename or browse through the list to select one. When your entry is complete, click **Select**.
8. In the Create Library dialog, click **OK**.
9. On the Libraries and Classpath page, if finished click **OK**.

To edit an existing library in a project:

1. With the project selected in the Application Navigator, open the Project Properties dialog.
2. Select the **Libraries and Classpath** node.
3. On the Libraries and Classpath page, select the library to be altered from the **Classpath Entries** list.
4. Click **Edit**. (This button remains the **View** button if the library is not editable.)
5. In the Edit Library Definition dialog, the appropriate library's name should appear in the first field. Make any desired changes to the library name by typing directly into the field.
6. For each Edit Path dialog, click **Add Entry** or **Add URL** as appropriate. To remove a path, or correct an addition, click Remove. To rearrange the order of entries, use the reordering buttons to the right of the display area.
7. Once you have clicked either **Add Entry** or **Add URL**, in the resulting selection dialog enter the directory name or browse through the list to select one. When your entry is complete, click **Select**.
8. In the Edit Library dialog, click **OK**.
9. On the Libraries and Classpath page, if finished click **OK**.

5.3.10.2 How to Remove Libraries from a Project

When you remove libraries from a project, the source paths defined for those libraries no longer form part of the project's classpath.

To remove a library from a project:

1. In the Application Navigator, select the appropriate project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.
3. Select the **Libraries and Classpath** node.
4. On the Libraries page, select the desired library or libraries from the **Libraries** list and click **Remove**.
5. If finished, click **OK**.

5.3.10.3 How to Set the Target Java SE for a Project

Setting the target Java SE specifies which Java SE JDeveloper will use when compiling and running your project.

To view or change the current Java SE for an individual project:

1. In the Application Navigator, select the appropriate project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.

   The Project Properties dialog opens with the common input paths displayed or on the last page that you viewed.
3. On the **Libraries and Classpath** page the Java SE Version used for the project is displayed. Click **Change** to define a new Java SE.
4. When finished, click **OK**.

5.3.10.4 How to Manage Project Dependencies

Complex applications generally comprise multiple projects, which may be related though dependencies. That is, project A must depend on project B when project A uses classes or resources from project B. When this dependency is set, compiling project A will automatically compile project B.

Deployment profile dependencies are created and edited in the Project Properties dialog available from the Tools menu.

To manage the project dependencies for an individual project:

1. In the Application Navigator, select the appropriate project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.
3. Select the **Dependencies** node.
4. On the Dependencies page, view the current dependency hierarchy for the project.
5. Select or deselect projects as desired.
6. To change the current dependency ordering, click **Ordering**.
7. When finished, click **OK**.

5.3.10.5 How to Associate Features with a Project

When features are associated with a project, JDeveloper's design time filters the choices you see based upon what you are most likely to need for a project of this type.

To associate features with a project via its project template:

1. From the main menu, choose **Application > Manage Templates**.
2. In the Manage Application Templates dialog, click the project template for which the features are to be associated.

   Application templates are listed as first-level nodes under **Application Templates**. Project templates appear below their application template.
3. In the panel to the right, select the appropriate features from the **Available Project Templates** list and use the shuttle buttons to transfer them to the **Selected Project Templates** list.
4. When finished, click **OK**.

To associate features with an individual project:

1. In the Application Navigator, select the appropriate project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.
3. Select the **Features** node.
4. On the Features page, click the **Add Features** button.
5. In the Add Features dialog, select the features to be associated with the project in the **Project Features** list.
6. Click the shuttle button to transfer your selection to the **Selected** list.
7. Click **OK**.

5.3.10.6 How to Set Javadoc Properties for a Project

Every project you create carries the JDeveloper project defaults or those you have supplied yourself for all projects. You can also replace these defaults on a project-by-project basis. Setting these properties is the same in either case: only the location, and application, of the information differs.

To set Javadoc properties for an individual project:

1. In the Application Navigator, select the project.
2. From the main menu, choose **Application > Project Properties**, or right-click and choose **Project Properties**.
3. Under **Profiles**, select the active profile node.
4. Under the active profile node, select the **Javadoc** node.
5. When finished, click **OK** to close the Project Properties dialog.

5.3.11 How to Manage Application and Project Templates

Application and Project templates can assist you in organizing your projects and standardizing on that organization. When you combine templates with features, you can streamline the way you design and produce applications.

5.3.11.1 How to Define a New Application Template

An application template organizes one or more project templates which specify the project types expected in the application. Using such templates enables you to standardize the way you develop an application.

To define a new application template:

1. Begin the process of creating a new application.
2. In the Create Application dialog, click **Manage Templates**. Alternately, if you are not in this dialog, choose **Application > Manage Templates**.

   For more information at any time, press F1 or click **Help** from within the appropriate dialog.
3. In the Manage Templates dialog, select the Application Templates node and click to open the Create Application Template dialog
4. Enter a name for the new template and click **OK**.

   The new template appears in the template list of the Manage Templates dialog. All application templates are listed as first-level nodes under **Application Templates**.
5. Complete defining the Application Template. For more information at any time, press F1 or click **Help** from within the Manage Templates dialog.

The application template appears in the New Gallery in the Applications category of the Business Tier.

5.3.11.2 How to Define a New Project Template

Project templates specify the various types of projects expected in a given application. Project templates are contained within application templates.

To define a new project template:

1. Define a new application template.

   Alternately, if the template has already been defined, choose **Application > Manage Templates**.
2. In the Manage Templates dialog, select the **Project Templates** node and click the **Add** icon to open the Create Project Template dialog.
3. Enter a name for the new template and click **OK**.

   The new template appears in the template list of the Manage Templates dialog. All project templates are listed as first-level nodes under **Project Templates**.
4. Complete defining the Project Template. For more information at any time, press F1 or click **Help** from within the Manage Templates dialog.

The project template appears in the New Gallery in the Projects category.

5.3.11.3 How to Share Application and Project Templates

You can create an application or project template in a shared location. Other users can read templates from the shared location and use the same templates for their application and projects.

To create a shared template:

1. Choose **Application > Manage Templates**.

2. In the Manage Templates dialog, select either the **Application Templates** or **Project Templates** node and click the **Add a shared location** icon.
3. In the Add Templates Directory dialog, enter or browse to the location where you want the shared template to be stored.

The shared templates folder is listed under both the Application Templates and Project Templates node.

5.3.11.4 How to Edit an Existing Application or Project Template

You can editing existing user-defined application or project templates.

To edit an existing application or project template:

1. From the main menu, choose **Application > Manage Templates**.
2. In the Manage Templates dialog, select the template you want to edit.

   For more information at any time, press F1 or click **Help** from within the Manage Templates dialog.
3. In the panel to the right, edit the attributes of the templates as desired.
4. When finished, click **OK**.

5.3.11.5 How to Delete an Existing Application or Project Template

You can delete existing user-defined application or project templates.

To delete an existing application or project template:

1. From the main menu, choose **Application > Manage Templates**.
2. In the Manage Templates dialog, select the name of the template to be deleted.

   Application templates are listed as first-level nodes under **Application Templates**. Project templates are listed as first-level nodes under **Project Templates**.

   For more information at any time, press F1 or click **Help** from within the Manage Templates dialog.
3. Click **Delete.**
4. Click **OK**.

5.4 Managing Application, Project, or Individual Files

This section describes how to save, rename, or close an application, project, or individual component.

5.4.1 How to Save an Application or Project

You can save an application or project in several ways.

To save all the components across applications, including all projects:

- From the main menu, choose **File > Save All** or click the **Save All** icon.

Alternately, you can save components individually by using **File > Save**.

It is important to note that saving the application or project container (`.jws, .jpr`) file alone does not save the individual files governed by that application or project. Nor does saving individual contained files save the container node.

Each node is an independent entity and must be saved as such. Using **Save All** takes care of changes to these container files, as well as all content files.

Using **Save** or **Save As** on a selected application or project node saves or duplicates the `.jws` or `.jpr` file only: it does not save or duplicate the files contained within the node.

Note too that if you do a Save As on a application or a project container file, that container is replaced, but the files contained are not altered. If you do a **Save As** on an individual file, that file is duplicated. However, if you want to rename a file, you should use **File > Rename**.

5.4.2 How to Save an Individual Component or File

You can save an individual component in several ways.

To save an individual component or file:

1. In the Application Navigator, select the component or file to be saved.
2. From the main menu, choose **File > Save** or click the **Save** icon in the toolbar.

   The file is immediately saved, its italicized name changing to Roman font.

It is important to note that saving the application or project container (`.jws`, `.jpr`) file alone does not save the individual files governed by that application or project. Nor does saving individual contained files save the container node.

Each node is an independent entity and must be saved as such. Using **Save All** takes care of changes to these container files, as well as all content files.

Using **Save** or **Save As** on a selected application or project node saves or duplicates the `.jws` or `.jpr` file only: it does not save or duplicate the files contained within the node.

You can rename an individual file or component using **File > Rename**.

Note that if you do a **Save As** on a application or a project container file, that container is replaced, but the files contained are not altered. If you do a **Save As** on an individual file, that file is duplicated.

5.4.3 How to Rename an Application, Project, or Individual Component

You can rename application control files, project control files, and individual files. The correct way of renaming Java classes is to use refactoring.

To rename an application or project container, or an individual source file:

1. In the Application Navigator, select the node to be saved.
2. From the main menu, choose **File > Rename**.

   For simple files, the Rename dialog opens. For Java files, the Rename File dialog opens.
3. If the Rename File dialog has opened, choose between renaming only the selected file, or renaming the file, the class defined by it, and all references to the class

   If you choose to rename the class and update references, the Rename *Object_Name* dialog opens.
4. If the Rename *Object_Name* dialog opens, change the name and choose options as required, then click **OK**.

5. If the Rename dialog opens, change the name as required and click **Save.**

   The node now appears in the navigator with the new name.

Alternately, you can use **File > Save As**. Note that **Rename** always replaces the target file. **Save As** replaces application or project container (`.jws`, `.jpr`) files, but duplicates source files.

When you are saving files, remember that saving a container file alone does not save the contents of the entire application or project. For that, you need to use **Save All**.

5.4.4 How to Relocate an Application, Project, or Project Contents

The Application Navigator presents a visual representation of the logical structure of applications and projects. It is not a file directory. It does not necessarily represent the physical location of those files.

To change the physical location of individual files, you can work in JDeveloper. To change the physical location of a group of files, it is easier to work through your operating system's file manager.

To change the association of files with projects or projects with applications, you would work in the Application Navigator, adding or removing as appropriate.

> **Note:** The best practice for relocating Java classes is to use the options available on the Refactor menu.

To change the physical location of an individual file, whether within the project or a container (.jws or .jpr) file:

1. In the Application Navigator, select the file to be moved.
2. From the main menu, choose **File > Rename**. If you have chosen a Java file, the Rename File dialog will open. You will be able to relocate the file only if you choose the option **Rename the file only, do not update references** in this dialog.
3. In the Rename dialog, navigate to the new location for the file and change the file's name if you wish.
4. Click **Save**.

   The file is now physically stored in the new directory. Its logical representation does not change in the navigator unless you explicitly alter it.

To change the physical location of an entire application or directory:

1. In your operating system's file manager, navigate to the directory in which the files currently reside. Files stored in the JDeveloper default directory reside in the `mywork` folder.
2. Select the entire directory (application, project, or files within a project) to be moved and move it to the new location.

   The files have now been moved, but JDeveloper no longer knows where they are.
3. When you return to JDeveloper, in the Application Navigator, and choose **Open Application** from the drop-down list.
4. Navigate to the new physical location of the application or project and click **Open**.

To change the physical location of a group of files from one project to another:

1. In your operating system's file manager, navigate to the directory in which the files currently reside.
2. Select the files to be moved and move them to the new location.
3. When you return to JDeveloper, select the project in the Application Navigator, and choose **Project Properties** from the context menu.
4. In the Project Source Paths page of the Project Properties dialog, use the **Add** button and navigate to the location of the files you want to add.

   The files are now physically located where you placed them in step 2, and logically associated in the navigator wherever you targeted them in step 4.

5.4.5 How to Close an Application, Project, or Other File

When you close an application, project, or file in the Application Navigator, that application or project is unloaded from memory. When an application or project is closed, it appears in its unexpanded form in the navigator.

In addition, you can remove applications, projects, or files from the navigator, which removes them only from the list, or you can delete them permanently, wherever they reside, from within JDeveloper.

To close an application or project:

1. In the Application Navigator, select the application or project to be closed.
2. From the main menu, choose **File > Close**.

   If any files within that application or project were changed and not saved, you are prompted to save them.

   The application or project now collapses and appears in the navigator with the plus sign indicating that is ready for expansion.

You can close a file opened in a viewer or an editor by clicking on the close box of the corresponding document tab above the editor window.

5.4.6 How to Remove a File from a Project

You can remove files from a project, which removes them only from the navigator list, or you can delete them permanently, wherever they reside, from within JDeveloper.

To remove a file from a project:

1. In the Application Navigator, select the file or files you wish removed.
2. Select **File > Delete**.
3. The Confirm Delete Dialog is displayed. If you are certain that you want to delete the file, click **Yes**.

5.4.7 How to Remove a Project from an Application

You can remove projects from the application by deleting the project control file (.jpr) from within JDeveloper.

To remove a project from an application:

1. In the Application Navigator, select the project you wish to remove.

2. Select **File > Delete Project**.
3. The Confirm Delete Project Dialog is displayed. To confirm the deletion, click **Yes**.

5.4.8 How to Remove an Application

You can remove an application from within JDeveloper:

To remove an application from the IDE:

1. In the Application Navigator, click the **Application Menu**.
2. Select **Close Application**.
3. The Confirm Close Application Dialog is displayed. Select an option based on your preference.

5.5 Managing Libraries and Java SEs Outside the Project Scope

JDeveloper enables you to manage libraries and Java SEs outside the project scope.

5.5.1 How to Import Libraries or Java SEs Outside the Project Scope

You can work with libraries completely outside the JDeveloper project scope, setting them up to be either available to you for use in any of your projects or available to a group of users across an installation.

To work with libraries or Java SEs outside of the scope of a project:

1. From the main menu, choose **Tools > Manage Libraries**.
2. In the Manage Libraries dialog, select either the **Libraries** or the **Java SE Definitions** tab.
3. Select the **User** node to import libraries for your own use. Select the **Extension** node to import libraries for use across a group.
4. Click **Load Dir**.
5. In the Load Directories dialog, navigate to the library that you wish to import and click **Select**.
6. When finished, click **OK**.

5.5.2 How to Create Libraries or Java SEs Outside the Project Scope

You can work with libraries completely outside the JDeveloper project scope, setting them up to be either available to you for use in any of your projects or available to a group of users across an installation.

To create libraries or Java SEs outside the scope of a project:

1. From the main menu, choose **Tools > Manage Libraries**.
2. In the Manage Libraries dialog, select either the **Libraries** or the **Java SE Definitions** tab.
3. Select the **User** node to create libraries for your own use. Select the **Extension** node to create libraries for use across a group.
4. Click **New**.

5. In the Create Library dialog or the Create Java SE dialog, complete the details for the new library or Java SE.
6. When finished, click **OK**.

5.5.3 How to Edit Libraries or Java SEs Outside the Project Scope

You can work with libraries completely outside the JDeveloper project structure, setting them up to be either available to you for use in any of your projects or available to a group of users across an installation.

To edit libraries or Java SEs outside the scope of a project:

1. From the main menu, choose **Tools > Manage Libraries**.
2. In the Manage Libraries dialog, select either the **Libraries** or the **Java SE Definitions** tab.
3. In the tab list, select the library to be edited. Its attributes are displayed in the fields to the right.
4. To change the Java SE executable, click **Browse**.
5. To change the class, source, or doc paths, select the path that you want to change then click one of the buttons beneath the paths panel: **Add Entry**, **Add URL**, or **Remove**.

   You can also reorder the entries, by clicking the up and down buttons in the right margin.
6. When finished, click **OK**.

5.5.4 How to Delete Libraries or Java SEs Outside the Project Scope

You can work with libraries completely outside the JDeveloper project scope, setting them up to be either available to you for use in any of your projects or available to a group of users across an installation.

To delete libraries or Java SEs outside the scope of a project:

1. From the main menu, choose **Tools > Manage Libraries**.
2. In the Manage Libraries dialog, select either the **Libraries** or the **Java SE Definitions** tab.
3. In the tab list, select the library to be deleted. You can delete only those libraries you have created.
4. Click **Remove** and respond to the confirmation dialog.

   The library is deleted immediately.
5. To close the Manage Libraries dialog, click **OK**.

6

Versioning Applications with Source Control

This chapter describes how to use source control systems to manage the versions of applications developed in a team environment. It discusses the available version control systems, how to download the various version-control extensions available to Oracle JDeveloper, and then includes instructions for each of the source control systems that can be used with JDeveloper.

This chapter includes the following sections:

6.1 About Versioning Applications with Source Control

Developing in teams often requires coordination among multiple developers who may be called upon to make changes to the same files, to track these changes against project management or bug reporting systems, and eventually to check in or commit their edited files to a commonly used repository of content that will be built into a functioning product.

To assist in team development of software products, JDeveloper integrates the popular version control system, Subversion, into its available feature set. You can access a number of commands for Subversion directly from the JDeveloper interface, through the Version menu or through the Versioning Navigator.

For users familiar with other versioning systems, JDeveloper offers support for CVS, Perforce and Serena Dimensions as downloadable extensions. You can choose from these and other extensions by selecting **Help > Check for Updates**.

Finally, this section includes a compilation of several best-practices recommendations on writing XML in a way that makes version control much more effective and useful for this important data type.

6.2 Downloading Source Control Extensions in Oracle JDeveloper

JDeveloper offers a number of tools for developing in teams. These include an integrated solution, Subversion, as well as a selection of extensions that interface JDeveloper with popular version control systems. In addition, an application lifecycle management system, Oracle Team Productivity Center, is also available as a downloadable extension.

While most members of a team will interact with their selected versioning system by checking files in and out and managing their changes to the project they are working on, at least one team member is typically required to administer and maintain the versioning system as it relates to JDeveloper. If you are the administrator for your versioning system, you will most likely have additional tasks beyond checking files in and out.

If your team uses one of the versioning systems that require you to download a JDeveloper Extension to integrate your versioning system with JDeveloper, you can browse for the versioning system from the Update Center by selecting **Help > Check for Updates**. Be sure to select all update centers when you search for your versioning system.

6.3 Using Subversion with Oracle JDeveloper

JDeveloper is integrated with the popular team development solution Subversion (SVN). If you are part of a team that uses Subversion, JDeveloper's Versioning menu contains commands for using Subversion to manage the content you are working on while maintaining a connection to your team's repository and tracking changes, merges, and more. Setting up Subversion involves creating a repository for your source-controlled files, making sure that JDeveloper can connect to that repository, importing files to the repository, and more.

In general, you begin by importing your working files into the Subversion repository to bring them under version control.

Once in the repository, your files are then available to be checked out from the Subversion repository to a local folder known as the "Subversion working copy". The working copy is typically in your local file system for ease and speed of access, but it can also be in a network location if you prefer.

When you create a new file in JDeveloper (or move it into JDeveloper), you store it in the Subversion working copy. When you are ready to make your work available to the team, you add these new files to Subversion control.

When it comes time to make your changed and new files available to other users, you can do so by committing them to the Subversion repository.

To take advantage of the work others on your team have done, you can copy changed files from the Subversion repository to your working copy by updating your files.

After completing setup, your work with Subversion will revolve around checking files out, editing them in JDeveloper, and checking them in with your changes. You may also need to resolve conflicts between changes you made and those made by others in your team. Files may also be moved in and out of Subversion control, and finally, you might use special properties of the files associated with specific versions for tracking bugs, customer requests, and other characteristics.

6.3.1 How To Set Up Subversion and JDeveloper

Setting up Subversion involves creating a repository for your source-controlled files, making sure that JDeveloper can connect to that repository, importing files to the repository, and more.

You do not have to install any Subversion software in addition to the JDeveloper Subversion extension, except in the following circumstances:

- You wish to create a local Subversion repository using the JDeveloper Subversion VCS extension.
- You wish to use a Java binding (helper library) other than SVNKit, which is the one supplied with the extension.
- You wish to connect to a Subversion repository through a proxy server

In all of the above cases, you will need to install separate Subversion client software. If you wish to use an alternate Java binding, you will additionally have to install the binding software.

An alternate Java binding that you could use is JavaHL. This has the advantage of being developed and maintained by the makers of Subversion and thus allows repository access via a wide range of protocols (`http`, `https`, `file`, `svn`, `svn+ssh`). To use the JavaHL binding, you must install Subversion client software independently of JDeveloper. Once installed, JDeveloper will allow you to choose between JavaHL and SVNKit client options.

To Install Subversion Client Software:

1. Download the Subversion installer (`svn-1.3.2-setup.exe`) from `http://subversion.apache.org/` (to, for example, `c:\downloads`).
2. Run the installer and place the Subversion client in a convenient location, for example `c:\subversion`. Reboot your computer.

This procedure assumes that the operating system is Windows. For non-Windows environments, consult the documentation for the operating system package management system to ensure the vendor-supplied Subversion client contains JavaHL.

To check the installation so far, open a command prompt and type **`svn help`**. You should see a list of subcommands. If not, check that the system path contains the bin directory of the location where the client software was installed (in this example, `c:\subversion\bin`).

To Install JavaHL Binding Software:

1. Download the JavaHL binary installer (`svn-win32-1.3.2_javahl.zip`) from `http://subversion.apache.org/` (to, for example, `c:\downloads`).
2. Using WinZip or a similar tool, extract the file `libsvnjavahl-1.dll` from `svn-win32-1.3.2_javahl.zip` into the `bin` directory of your Subversion client installation (in this example, `c:\subversion\bin`).
3. Run the installer and place the Subversion client in a convenient location, for example `c:\subversion`. Reboot your computer.
4. Start or restart JDeveloper.

6.3.1.1 How to Connect to a Subversion Repository Through a Proxy Server

If you wish to connect to a Subversion repository through a proxy server, you must first install separate Subversion client software.

Once you have installed the Subversion client software, you will have a Subversion subdirectory in your Windows Application Data directory. To find the Application Data directory, at the `c:/` prompt type **`cd %APPDATA%`**. Then open the Subversion subdirectory. (On Linux the equivalent subdirectory will be in `~/.subversion`, where ~ is the home directory.)

In the Subversion subdirectory will be a file named `servers`. Open this file with a text editor and find the [global] section. Remove the comment marker (#) from the line `http-proxy-host` and overtype the placeholder proxy information with the details of the proxy server that you use. Remove the comment marker (#) from the line `http-proxy-port` and overtype the placeholder port information with the port number for the proxy server. If you wish to exclude certain URLs from using the proxy server, remove the comment marker (#) from the line `http-proxy-exceptions` and overtype the placeholder URLs with URLs that you wish to exclude.

Add additional `http-proxy-host` and `http-proxy-port` lines with details of any other proxy servers that you use.

It is important that the proxy server supports all the `http` methods used by Subversion. Some proxy servers do not support the following methods by default: PROPFIND, REPORT, MERGE, MKACTIVITY, CHECKOUT. If you experience problems with using a proxy server to access a Subversion repository, ask the server's system administrator to change the configuration to support these `http` methods.

6.3.1.2 How to Check the Installation

In JDeveloper, select Subversion as the versioning system (**Versioning > Configure**, and then select **Subversion**).

Open the main Subversion preferences page (**Tools > Preferences > Versioning**), and then check that the required client installation is available. If more than one is listed, select the one that you wish to use.

Important: If you subsequently accept an update of the JDeveloper Subversion extension from the Update Center (Official Oracle Extensions and Updates), the client preference will be reset to SVNKit, even if you had previously chosen an alternate client.

6.3.1.3 How to Create a Subversion Repository

In most cases, you will connect to your team's Subversion repository. As you develop your projects and applications, you will check files out from the Subversion repository, modify them, and check them back in. This is the typical, and recommended, practice for using Subversion.

Depending on your installation, however, you may find it necessary to create a Subversion repository on your local file system through JDeveloper. A connection to the repository will be created at the same time.

JDeveloper will try to use the `file:///` protocol to access the newly created repository. SVNKit, the Subversion client installed with JDeveloper, supports the `file:///` protocol. If you are using a Subversion client that does not support the `file:///` protocol, you will need to use a different access method (`http://`, `https://`, `svn://` or `svn+ssh://`). Consult the Subversion documentation for how to do this.

To Create a Subversion Repository:

1. Choose **Versioning > Create Local Repository**.

If your installation does not support local repository creation, you will see an error message. Otherwise, the Create Subversion Repository dialog will open.

2. Complete the Create Subversion Repository dialog.

   To obtain help while using the dialog, press **F1** or click **Help**.

To Browse a Subversion Repository:

1. Expand the connection to your Subversion repository in the Versioning Navigator.
2. Double-click on a folder to view its contents.
3. Right-click on an element to view available operations.

6.3.1.4 How to Create or Edit a Subversion Connection

Before you can work with a Subversion repository through JDeveloper, you must create a connection to it. You can subsequently edit the connection details if they change for any reason.

Typically, you will obtain the details of your Subversion connection (server name, user ID, password, etc.) from your team or version control administrator. You will need to know those details before you create a connection to your Subversion repository.

To Create a Subversion Connection:

1. In the Versioning Navigator (**View > Team > Versioning Navigator**), right-click the Subversion node and choose **New Repository Connection**.

   The Create Subversion Connection dialog is opened. For help when using this dialog, press F1 or click **Help**.
2. Enter the URL of the location of the Subversion repository.
3. Optionally, enter a name for the connection.
4. If the Subversion repository has been set up with password protection, enter the username and password.
5. If you want to test the connection to the Subversion repository, click the **Test Connection** button. The results will be displayed in the Status area.
6. To complete the connection, click **OK**.

To Edit a Subversion Connection:

1. In the Subversion Navigator (**View > Team > Versioning Navigator**), right-click the Subversion connection name and choose **Properties**.

   The Edit Subversion Connection dialog is opened. For help when using this dialog, press F1 or click **Help**.
2. Make changes as required and click **OK**.

6.3.1.5 How to View Subversion Repository Content

You can view the current content of the Subversion repository through the Versioning Navigator. The nodes under your selected Subversion connection unfold to reveal the structure and file content of the Subversion repository.

You can open a read-only version of a Subversion repository file by choosing Open from its context menu. This will let you see what changes have been made to the files in the Subversion repository since you checked out or updated your local versions.

Folders in the Subversion repository, visible from the Versioning Navigator, offer the following operations:

New
Opens the new gallery, from which you can create applications, connections, projects, and other entities.

New Remote Directory
Opens the Create Directory dialog, which lets you create a new directory to associate with the URL of the element on which you right-clicked.

Delete
Removes the selected element immediately from the JDeveloper view, without a confirmation dialog. Use with caution.

Check Out
By default, opens the **Check Out from Subversion** dialog.

If you have configured JDeveloper for a different version control system, Check Out will open the checkout dialog for your selected version control software.

6.3.1.6 How to Check Out Files from the Subversion Repository

When you begin working on a project in Subversion, you check out the files you will be working with. It is recommended that you check out the entire application from the Subversion repository, so that you will have access to all files in that application in your local work area. Subversion uses the term modules to refer to the application it is recommended you check out.

> **Note:** With Subversion, there is no "check in" procedure, so you do not check in files that you have updated and check them out again when you next want to work on them. Instead, when you have finished working on your local copies of files, you commit them to the Subversion repository to bring the files held there up to date.

To incorporate into your local copies changes that other developers have committed to the Subversion repository since you checked out your files, you update them.

When you check out Subversion files, they are copied from the Subversion repository to a new location (specified by you) on your local machine. This location and the files within it constitute the Subversion "working copy".

To check out modules from the Subversion repository:

1. In the Versioning Navigator, select the repository node or folder containing the files that you want to check out.
2. Choose **Versioning > Check Out**.

   If there is no current connection to a Subversion repository, the Create Subversion Connection dialog is opened for you to create one.

   The Check Out from Subversion dialog is displayed. To obtain more information when working with the dialog, press F1 or click **Help**.
3. Make sure that the Subversion connection that the dialog displays is the correct connection (if you have more than one Subversion connection or repository).

4. Browse to the path in the Subversion connection containing the application files you wish to check out.
5. Enter the destination in your work area to which you wish the checked-out files to be copied, or click Browse to navigate to your local work area.
6. You have the option of checking specific tags, if your team uses them.
7. If you wish to check out files within folders contained by this Subversion module, make sure to select Depth.

   When you have made all your selections, click **OK**.

6.3.1.7 How to Update Files from the Subversion Repository

Use these procedures to bring the files you have been working on up-to-date with a revision in the Subversion repository.

It is recommended that you perform the update operation on a working copy.

When you use Update Working Copy, all the files in your checked out working copy will be updated, regardless of which node you have active in your application in the JDeveloper Application Navigator. The alternative is to select Update. This will only update the folder or file (and any child folders and files) that you have selected in the Application Navigator.

To update a working copy (recommended):

1. In the Application Navigator, select a navigator node or file that is part of the working copy.
2. Select **Versioning > Update Working Copy**.

   The Update Working Copy dialog is displayed with the working copy folder listed.
3. Ensure that the folder shown is the correct one for the working copy that you wish to update. If it is not, cancel the dialog and begin again from step 1.
4. Set the options on the Update Working Copy dialog as required.

   To obtain more information about the options, press F1 or click **Help**.
5. To update the working copy from the Subversion repository, click **OK**.

You can also update individual files. However, this runs the risk of not updating all files that may have been modified by your team members since the last time you checked them out.

To update individual files:

1. In the Application Navigator, select the file(s) that you wish to update and choose **Versioning > Update**.

   The Update Resources dialog is displayed with the file(s) listed.
2. Set the options on the Update Resources dialog as required.

   To obtain more information about the options, press F1 or click **Help**.
3. To update the listed file(s) from the Subversion repository, click **OK**.

6.3.1.8 How to Import JDeveloper Files Into Subversion

Files that you created within (or brought into) JDeveloper before using Subversion control must be imported into the Subversion repository, and then checked out from it.

To import an existing JDeveloper project or application into Subversion:

1. In the Application Navigator, select the application or project that you want to import into Subversion.
2. Select **Versioning > Import Files**.

   The Import to Subversion wizard opens.
3. Complete the wizard. For help while using the wizard, press F1 or click **Help**.

   If you allowed the wizard to check out the imported files, the files are shown in the Application Navigator with a version number next to them. You may have to refresh the view before the files are shown.

   If you did not allow the wizard to check out the imported files, you must now check them out before you can work on them.

6.3.1.8.1 How to Import an Application to Subversion

You can also import an entire application into Subversion using the JDeveloper Version Application feature.

To import files using Version Application:

1. Select the application you wish to add to version control.
2. Select **Versioning > Version Application**.
3. From the Version Application dialog, select the Subversion repository. This will open the Import to Subversion wizard.

After you import files into Subversion using the Version Application feature, you will notice that Subversion creates two directories, one as your work area and one as a backup directory.

For example: after creating a new application called Catalog, select V**ersioning > Version Application > Subversion**. Be sure to select the **Perform Checkout** from the Options page, then finish the wizard.

When the wizard completes, browse to the local directory that you have specified as the Source Directory for this application in Subversion. You will see two directories listed there: `Catalog.svn-import-backup` and `Catalog.svn-import-workarea`.

JDeveloper (and Subversion) will use the `Catalog.svn-import-workarea` directory for file access, checkout/checkin, and other activities. You should avoid editing, moving, or manipulating files in those directories outside of JDeveloper and Subversion.

6.3.2 How to Work with Files in Subversion

After completing setup, your work with Subversion will revolve around checking files out, editing them in JDeveloper, and checking them in with your changes. You may also need to resolve conflicts between changes you made and those made by others in your team. Files may also be moved in and out of Subversion control, and finally, you might use special properties of the files associated with specific versions for tracking bugs, customer requests, and other characteristics.

6.3.2.1 How to Add a File to Subversion Control

When you create a new file in JDeveloper that is part of a local working copy (that is, an application that has been versioned and checked out of your SVN repository), you need to add and then commit the file to Subversion control before you can use the

JDeveloper Subversion facilities with it. The preferred method is to set up JDeveloper to do this automatically, through the Preferences menu.

To add new files on commit:

1. Select **Tools > Preferences > Versioning > General**.
2. Select **Automatically Add New Files On Committing Working Copy.**
3. Click **OK**.

You can also place individual files under Subversion control.

To put individual files under Subversion control:

1. Select the files in the Application Navigator and choose **Versioning > Add**.

   The files can be your work files, or they can be the application and project files used by JDeveloper.

   The Add to Source Control dialog is displayed with the files listed.

2. To add the files to Subversion control, click **OK**.

   The files are now shown in the Application Navigator with a black cross, meaning that they are stored in your JDeveloper workarea but are not yet committed to the Subversion repository. Files must be committed to the Subversion repository before they can be versioned and accessed by other users.

3. To commit files to the Subversion repository, select the files in the Application Navigator and choose **Versioning > Commit**.

   The Commit Resources dialog is displayed with the files listed.

4. Add your versioning comments in the Comments box.

   You will later be able to see these comments when viewing the list of versions of a particular file.

5. To commit the files to the Subversion repository, click **OK**.

   The files are now shown in the Application Navigator with an orange dot, indicating that they are known to the Subversion repository and are up to date.

6.3.2.2 How to Use Change Sets

Change sets, or change lists, are essentially labels which can be applied to working copy files to enable group operation on each change list. The idea behind adding files to a change set is similar to sorting files into directories, but change lists can be created dynamically and the labels applied to each file, regardless of their level in the file system hierarchy. You can then address all the files in the change set as a single group. For example, if you make a single bug fix that requires editing three different files, you can add all three files to the change set and track them as a single logical unit in the JDeveloper Pending Changes window.

Subversion lets you associate files with a named change set, either manually or automatically. You make additions to the change set through the menu system; automatic additions are also possible through default association, when JDeveloper detects outgoing changes.

You can browse changes for a named change set in a view of the Pending Changes window. From there, you can manipulate the change sets, and commit associated changes to the repository.

To add a selected file to a new change set:

- Select a file from the Pending Changes window, then select **Versioning > Add To > New Change Set**.

To add file to a change set:

1. In the Pending Changes window, select a file to add to an existing change set and click the right mouse button.
2. Select **Add To**, then choose an existing change set.
3. Select one of the existing change sets displayed in the dialog, or select New Change Set to create a new change set containing this file.
4. Click **OK**.

6.3.2.2.1 Editing Change Sets JDeveloper creates a default, unnamed change set for each installed version control system, and uses this change set until you specify another change set as the default. You can make changes to the content and the status of individual change sets, including this default change set, by right-clicking any change set and selecting from the following:

Edit
Change the content of the selected change set.

Remove
Deletes the selected change set from Pending Changes. Does not delete the files associated with the change set.

Make Default
Makes the selected change set the default for future operations. All newly created and edited files will be made part of this change set until you either change the default or manually add the file to a different change set.

6.3.2.3 How to View the History of a File

Use this procedure to open the History Viewer and view the history of Subversion files.

To view the history of a file:

- With the file selected in the Application Navigator, choose **Versioning > Version History** from the context menu.

  For more information while using the History Viewer, press F1 or click **Help**.

6.3.2.4 How to Commit Files to the Subversion Repository

Use these procedures to bring the Subversion repository up to date with the latest version of the files you have been working on, at the same time adding any new files to or removing any unwanted files from the Subversion repository.

You can perform the commit operation on individual files, or in the context of a working copy.

If an individual object that you want to commit has uncommitted parent objects, you must first commit the parent objects. An alternative is to commit the working copy that the objects are part of, in which case all the uncommitted objects will be committed.

You can also use change sets to manage groups of files, which can help ensure that you commit all files pertaining to a particular sub-project or task within the overall application.

To commit individual files shown in the Application Navigator or the Pending Changes window:

- Select the file(s) and choose **Versioning > Commit.**

  The Commit Resources dialog is displayed with any files listed. Set the options on the Commit Resources dialog as required.

  To obtain more information about the options, press F1 or click **Help**. To commit the listed file or files to the Subversion repository, click **OK**.

To commit a working copy from the Application Navigator:

1. Select a navigator node or file that is part of the working copy.
2. Select **Versioning > Commit Working Copy**.

To obtain more information about the options, press F1 or click **Help**. To update the Subversion repository with the content of the working copy, click **OK**.

When you use Commit Working Copy, all the files in your checked out working copy will be updated, regardless of which node you have active in your application in the JDeveloper Application Navigator. The alternative is to select **Commit**. This will only commit the folder or file (and any child folders and files) that you have selected in the Application Navigator.

Additionally, you can commit a working copy from the Pending Changes window.

To commit a working copy from the Pending Changes window:

1. Put the Pending Changes window into Outgoing Changes mode.

   To obtain more information about the Pending Changes window, press F1 or click **Help**.
2. Select a file from the working copy that you wish to commit.
3. Select **Versioning > Commit Working Copy**.

6.3.2.4.1 Saving Work Item ID with the Oracle Team Productivity Center Extension If you are using Oracle Team Productivity Center, the work item ID will automatically be saved as a tag in the comment dialog when you commit.

6.3.2.5 How to Use Templates in Subversion

Many team environments require the developer to enter comments when a file is checked in. These comments might include bug report numbers, dependencies on other files, or some other explanatory information to be stored in connection with the file being committed to the repository.

JDeveloper lets you create and select templates for use with such comments. The templates are available from the Commit menu.

To create a new template:

1. Select **Tools > Preferences > Versioning > Subversion > Comment Templates**.
2. Click **Add**.

To select a template:

1. Click **Choose a template or previous comment**.
2. Select the template from the list.
3. Click **OK**.

To make a comment to a file being committed:

1. Click in the Comments box to select it.
2. Type the comment you wish to make with the file being committed.
3. Click **OK**.

6.3.2.6 How to Revert Files to their Previous State

Use the Revert command to:

- Undo changes that you have made locally to the contents of a file.
- Change the status of a file that has been added, but not yet committed, back to unadded.
- Stop a file that is scheduled for removal (in the Pending Changes window) from being removed from the Subversion repository.

To revert a file:

1. Select the file in the Application Navigator or Pending Changes window and choose **Versioning > Revert**.

   The Revert Local Changes dialog is displayed with the file or files listed.

   For help while using the dialog, press F1 or click **Help**.
2. To revert the listed file or files, click **OK**.

6.3.2.7 How to Replace a File with the Subversion Base Revision

Use this procedure to replace a file with the base revision. The base revision is the revision from which the one you are currently working on originated.

To replace a file with the Subversion base revision:

1. In the Application Navigator, select the file to be replaced.
2. Choose **File > Replace With Base Revision**. The Replace With Base Revision dialog opens. Check that the file that you want to replace is shown in the dialog.
3. To replace the file, click **OK**.

6.3.2.8 How to Compare Files in Subversion

Use these procedures to compare files that are under Subversion control with other revisions of the same files, or with other files**.**

To compare revisions of a file:

1. From the context menu for the file, choose **Compare With**.
2. Select either **Previous Revision, Latest Revision**, or **Other Revision**.

   If there are no differences, a message is displayed. Otherwise the revision or revisions are shown in the Compare panel of the History tool.

To compare a file with another file:

1. From the context menu for the file, choose **Compare With > Other File**.

   The Select File to Compare With dialog is opened.

2. Select the file to be compared.

   The files are shown in the Compare panel of the History tool.

To compare two files:

1. Select the two files in the Application Navigator.
2. From the context menu for one of the files, choose **Compare With > Each Other**.

   The files are shown in the Compare panel of the History tool.

You can hide (and later expose) the Compare panel of the History tool to view other panels in JDeveloper.

6.3.2.9 How to Resolve Conflicts in File Versions

If there is a conflict between your copy of the file and the one in the Subversion repository, the icon next to the affected file will include an exclamation point. You will not be able to submit such a file to the Subversion repository. To overcome this problem, you should do one of the following:

- Revert to a non-conflicting version of the file.
- Resolve the conflict using the JDeveloper merge tool.
- Indicate to the Subversion control system that the conflict has been resolved (**Versioning > Mark Resolved**), even if no changes have been made (usually necessary only for binary files).

Another reason you might need to do this is if you have resolved the conflict yourself in the file, rather than using the merge tool. This might be the case if you have chosen to merge files at the server rather than the more usual solution of merging files locally.

To revert to a non-conflicting file version:

- Select the file in the Application Navigator and choose **Versioning > Revert.**

To resolve the conflicts using the merge tool:

1. Select the file in the Application Navigator and choose **Versioning > Resolve Conflicts**.

   The file is opened with the Merge tab displayed, showing the merge tool.

2. Use the merge tool to resolve the conflicts.

   For help while using the merge tool, press F1 or click **Help**.

To indicate that the conflict has been resolved, even if no changes have been made:

- Select the file (usually a file with binary content) in the Application Navigator and choose **Versioning > Mark Resolved**.

6.3.2.10 How to Resolve Conflicts in Subversion

Use this procedure to merge two revisions of a file, where the revisions contain conflicting content. Conflicts are notified in the Pending Changes window: the

outgoing status is "conflicts" or "conflicts on merge", and the Resolve Conflicts button is active.

To merge two revisions with conflicting content:

1. On the Outgoing tab of the Pending Changes window, select the revision that has conflicts and click the **Resolve Conflicts** button. (You can also select the revision in the Application Navigator.)
2. The merge tool is opened (as the Merge tab of the file editor).

   For help while using the merge tool, press F1 or click **Help**.

   The merge tool has three panels. The left panel contains the content of the version in the repository. The right panel contains the content of the most recent local version. The center panel contains the results of the merge. In the margins between the panels are symbols representing suggested actions to resolve each conflict.
3. View the suggested actions for resolving the conflicts by reading the tooltip of the margin symbols.

   More suggested actions may be available from the context menus of the margin symbols.
4. Resolve the conflicts by implementing a suggested action in each case.

   Accepting an initial suggested action may cause the appearance of additional suggested actions.

   You can also make changes to the content of the center panel by typing into it.
5. To complete the merge, save the changes that have been made by clicking on the **Save Changes** button on the merge tool (not the JDeveloper Save option).

6.3.2.11 How to Resolve Property Conflicts in Subversion

Subversion allows you to create and save properties associated with folders or files. These properties have a name and a value string.

You can resolve any such conflicts using Subversion's Resolve Tree Conflicts feature.

To resolve Subversion property conflicts:

1. In the Application Navigator, select the element under Subversion control that has a property conflict.
2. Click the right mouse button, and then select **Versioning > Resolve Tree Conflict**.

This displays the versions with the conflicting properties in two adjacent panes, as with the Version Compare.

To resolve the conflict, you can make changes in the Subversion Properties window.

6.3.2.12 How to Use the Merge Wizard

The Merge Wizard is instrumental to the way that JDeveloper supports Subversion merge tracking. Merge tracking in Subversion means in essence that Subversion remembers your merges so you don't have to. The Merge Wizard provides you with an easy way of selecting which components you wish to merge, such as specific revisions, branches, or change sets.

The Merge Wizard gives you a number of options:

Merge Selected Revision Range
Select this when merging a range of revisions to another branch, for example, when you are back-porting a group of bug fixes to the release branch.

Reintegrate a branch
Normally used when merging the changes on a branch back to the trunk, for example, if you completed the work on a feature branch and want to reintegrate the changes back to trunk.

Merge two different trees
Select this to merge the differences between two branches into the working copy.

Block specific revisions from being merged
Select this if you know that specific revisions are not yet ready, or not appropriate, to be merged into the trunk.

6.3.2.13 How to Work with Branches and Tags

When you wish to work on files independently of the main line of development (the "trunk") you can create a branch. Using the same feature, you can also create a tag, a collection of files that captures the state of development at a particular point.

When you wish to put the work you have being doing on a branch back into the main line of development, you can start the process by using the merge revision facility. This will compare the content of two revisions and apply the differences to the current working copy. You can also use this facility whenever you wish to copy changes made in one revision to another revision.

You may want to change your working copy so that it is based on a different branch. You can do this using the switch feature, either as part of branch creation or independently.

To create a branch or tag:

1. Ensure that you have committed your files to Subversion before continuing.
2. In the Application Navigator, select a project or file that is in the line of development that you wish to branch or tag.
3. Select **Versioning >Branch/Tag**.
4. Complete the Branch/Tag dialog.

   For help when completing the dialog, press F1 or click **Help**.

To use the merge facility (that is, to compare two revisions and apply the results to the working copy):

1. In the Application Navigator, select a project or file that is in the start revision (that is, the resource that is to be compared against).
2. Select **Versioning > Merge**.
3. Complete the Merge dialog.

   For help when completing the dialog, press F1 or click **Help**.

To switch the working copy to be based on another location in the repository:

1. In the Application Navigator, select a project or file that is in the current working copy.
2. Select **Versioning > Switch**.

3. Complete the Switch dialog.

   For help when completing the dialog, press F1 or click **Help**.

6.3.2.14 How to Add and View Subversion Properties

Subversion lets you define and add properties to various levels of the elements in the Application Navigator: files, folders, and other resources. You can define these properties and use them as a way of tracking files or folders that have something in common. For example, you can associate a specific Subversion property with a newly added feature. Viewing all files or folders with this Subversion property lets you see all the files associated with this feature: an HTML file, a JavaScript file, a class definition file, or any other elements that are involved in adding this new feature to your application.

If your team has been using Subversion properties for some time, you can use the View Subversion Properties menu to see a list of all elements that use a selected Subversion property. You can also compare the Subversion properties between different versions.

To view a list of Subversion properties:

1. Select an element under Subversion control from the Application Navigator.
2. Select **Versioning > Subversion > View Subversion Properties**.

If your project needs a new property for tracking and managing a particular aspect (such as a new feature or a bug fix), you can also add new properties.

To add a new Subversion property:

1. Select an element under Subversion control from the Application Navigator.
2. Select **Versioning > Subversion > Add Subversion Property**.
3. Enter the values for the property, then click **OK**. Refer to the following section for examples of Subversion properties and how to use them.

6.3.2.14.1 Example of Subversion Properties

When you add or edit Subversion properties, the dialog lets you select or specify the following elements:

Resource file

The file (or folder or other resource) to which this property is to be applied. To change this value, select a different file or resource. Note that if you wish to add this property at the application or project folder level, edit the resource file entry so that it refers to the folder, not the file.

Property name

Select a property name from the available list, or enter a new name to create a new Subversion property. Preface the new property name with svn: to be tracked as a Subversion property.

Value string

Enter the string to be displayed with this Subversion property when you view properties. For example, you can associate a specific Subversion property with a particular bug identification number or a specific upcoming release.

Note that the Value String might differ depending on the property. For instance, consider a property named `svn:externals` meant to record the connection between a local file and its external URL in the SVN repository. This property's value string

would be a pair of text strings, respectively showing the local directory where the external file is to be checked out and the URL to the external file in the SVN repository

Assume for this example that the resource file is `D:\temp` and the property name is `svn:externals`. The value string (a value pair) might be:

```
external_libs
https://ukp16449.uk.oracle.com/repos/trunk/FOD/StoreFront.jar
```

This indicates that the file `StoreFront.jar` held in the Subversion repository at that URL is to be checked out to `D:\temp\external_libs`. If the Value String entries were held in a specific file pointed to from this property, use the Value File entry.

Value file
If you know you will be adding the same Subversion property to a number of resources in your application, you can save the value string in a text file. Click **Browse** to select the text file that contains the value string you wish to use.

Set property recursively
Select this if you wish Subversion to apply this property to all files and elements below the current level in the application or project hierarchy.

6.3.2.14.2 Specifying a Revision Number with a Subversion External Property When you set an external property with a revision number, make sure you follow the correct format for the value string. You can use either of the following as the value string for a property of type **`svn:external`** to set the ExternalWebINF revision to 16, using the JDeveloper integrated Subversion:

```
ExternalWebINF -r 16
https://myserver.myteam.com/svn/repos/public-html/WEB-INF
```

```
https://myserver.myteam.com/svn/repos/public-html/WEB-INF@16
ExternalWebInf
```

However, note that with Subversion 1.4 clients, only the first format is accepted.

6.3.2.15 How to View the Status of a Subversion File

Use this procedure to check the content status and any associated property status of a file that is under Subversion source control. You can also refresh the status of a file.

To view the status of a file:

1. With the file selected in the Application Navigator, open the context menu and select **Versioning > Properties**.
2. Select the Versioning tab.

The status labels shown are those used by Subversion to describe the source control status of content and any associated property.

The main statuses for content are:

- **added** - The content has been added to source control but has not yet been committed to the Subversion repository.
- **modified** - The property has been locally modified since it was copied from the repository.
- **unmodified** (normal) - The property is unmodified since it was last updated from the Subversion repository.

- **conflicted** - There were conflicts when the property was updated from the Subversion repository.
- **deleted** - The file (content and any associated property) will be removed from the Subversion repository with the next commit action.

The main statuses for associated properties are:

- **modified** - The property has been locally modified since it was copied from the repository.
- **unmodified** (normal) - The property is unmodified since it was last updated from the Subversion repository.
- **conflicted** - There were conflicts when the property was updated from the Subversion repository.

6.3.2.16 How to Refresh the Status of Files Under Subversion Control

The source control status of a file is indicated in the JDeveloper navigators (Application Navigator and Team Navigator) by icon overlays, as below.

If the status of a file is changed outside JDeveloper, for example by using a Subversion client application, the new status might not immediately be shown in JDeveloper. To ensure that the status indicated in the navigator matches the true status of the file in the source control system, you can perform a manual refresh.

To refresh the status of files in JDeveloper:

- Select **View > Refresh**.

6.3.2.17 How to Remove Files from Subversion Control

If you wish to remove a file from Subversion control, use the JDeveloper Delete feature. This performs a "safe delete," which searches for usages of the file you are deleting and provides you with a dialog with options for proceeding.

To remove a file from Subversion control:

1. In the Application Navigator, select the file to be removed from Subversion.
2. Select **Edit > Delete** (or right-click the file and select **Delete**).
3. Make sure that **Delete Safely** is selected.
4. Click **OK**.

   If JDeveloper finds usages of the file you are deleting, a dialog will offer you options for proceeding. Choose the appropriate option, then click **OK**.

6.3.3 How to Use Export Features

Subversion provides features for creating and applying patches—methods for determining changes between two revisions of a file, and then applying those changes to a third file. In addition, Subversion contains features for exporting the details about repository connections, as well as files in the repository.

6.3.3.1 How to Create and Apply Patches

You may wish to record the changes between two revisions of a file, then apply those changes to a third file. You do this by creating a patch and then applying it.

To create a patch:

This generates a patch comprising the differences between a controlled revision of a file and a revision of the file held locally.

1. In JDeveloper, open the file for which you want to create a patch.
2. Click the **History** tab.

   The History view lists all the revisions of the file. In the lower portion of the History view, the left pane shows the contents of a local revision, and the right pane shows the contents of the controlled revision.
3. Select the revision combination for which you want to create a patch.
4. From the context menu, choose **Generate Patch**.

   The Select Patch Context dialog may open. For help while using this dialog, press F1 or click **Help**.

   The Generate Patch dialog opens. Complete the dialog as required. For help while using the dialog, press F1 or click **Help**.

To apply a patch:

1. In the navigator, select the resource to which you want to apply a patch.

   The resource can be an application, a project, or a source file.
2. Select **Versioning > Apply Patch**.

   If you chose to apply a patch to a project, the Select Patch Context dialog opens, through which you should specify whether you are applying a project file (.jpr) patch, or whether you are updating the contents of a project.

   The Apply Patch dialog is opened.
3. In the grid at the top of the Apply Patch dialog, check that the target resources are correctly identified.
4. Choose the source of the patch. For more information about this and the other options on the dialog, press F1 or click **Help**.
5. Click Preview. This opens the Apply Patch Preview window, in which you can accept or reject particular changes. For more information about the options in the Apply Patch Preview window, press F1 or click **Help**.
6. To apply the patch, click **OK**.

6.3.3.2 How to Export Subversion Controlled Files from JDeveloper

You can export copies of JDeveloper files that are under Subversion control. You can do this from the Application Navigator, in which case the files will be exported from the Subversion "working copy", or from the Subversion Navigator, in which case the files will be exported from the Subversion repository. If you export using the Subversion Navigator, you can specify which revision of the files to export. Exporting the files means copying them to a local file system directory that you specify.

To export files from the Subversion "working copy":

1. In the Application Navigator, select the project containing the files that you wish to export.
2. Select **Versioning > Export Files**.

   An Export Files dialog opens.

3. In the Destination Path box, enter or browse to the location where you want the files to be copied to.
4. To export the files, click **OK**.

To export files from the Subversion repository:

1. In the Subversion Navigator, select the repository node or directory containing the files that you wish to export.
2. Select **Versioning > Export Files**.

   An Export Files dialog opens.
3. In the **Destination Path** box, enter or browse to the location where you want the files to be copied to.
4. If you want to export a particular revision of the files, select **Use Revision** and enter the revision number in the adjacent text box.
5. To export the files, click **OK**.

6.3.3.3 How to Export and Import Subversion Repository Connection Details

You can export the details of your Subversion repository connections to a file. You can subsequently import the connection details from the file to recreate the Subversion repository connections.

To export Subversion connection details to a file:

1. In the Subversion Navigator, select the Subversion node and, from the context menu, choose **Export Connections**.

   The Export Subversion Connections dialog opens.
2. Enter a location and name for the file that will contain the connection details, then click OK.

To import Subversion connection details from a file:

1. In the Subversion Navigator, select the Subversion node and, from the context menu, choose **Import Connections**.

   The Import Subversion Connections dialog opens.
2. Browse to the file that contains the connection details that you wish to import, then click **OK**.

6.4 Using Concurrent Version System (CVS) with Oracle JDeveloper

JDeveloper allows you to use the source control features of Concurrent Versions Support (CVS). Once you have installed the CVS extension, JDeveloper works in a seamless manner with CVS so that you can connect to the CVS repository, check out files, work on them and commit changes, all from within the JDeveloper navigators and menus.

Note: For extensive information about how to use and administer CVS, see the CVS online manual at `http://www.cvshome.org`.

6.4.1 How to Set Up CVS with Oracle JDeveloper

In general, CVS uses a common repository of files, accessible to JDeveloper, that you and your team share while developing a software project. To modify files in that repository, you first check them out so that CVS tracks the who, when, and what of file access. In the event that two team members edit the same file at the same time, CVS contains tools that help you determine whether those changes conflict, and to resolve problems that may arise and merge these simultaneous changes into a single, comprehensive file. Finally, CVS lets you check these changed files back into the repository so that your build tools will have access to the latest files, with new and/or merged content.

Before you can use CVS to manage your shared content, you need to connect JDeveloper to CVS. This means configuring JDeveloper, making a connection to your team's CVS repository, creating a local repository, and more. The topics in this section cover all the steps you'll need to make sure CVS is available from JDeveloper after downloading the CVS extension from Check For Updates. If your team is already using CVS, you should check with them for specifics on how CVS is implemented in your organization.

The process of setting up CVS with JDeveloper involves configuring JDeveloper, creating a CVS connection, importing files for the project into your CVS repository, and then checking out the CVS modules to be edited.

6.4.1.1 How to Configure JDeveloper for Use with CVS

Before you can use CVS, you need to configure JDeveloper by setting preferences.

To configure JDeveloper for use with CVS:

1. Choose **Tools > Preferences**, then select **Extensions** from the left panel of the Preferences dialog.
2. In the right panel, ensure that Versioning Support *n.n* is checked.
3. In the left panel of the Preferences dialog, open the Versioning node and then the CVS node. The main CVS preferences panel is shown. Other CVS preferences panels are shown when you click on the items beneath the CVS node.
4. Make changes to the preferences as required. For more information about the specific preferences, press F1 or click **Help**.
5. If you wish to use binary file types with CVS, select the File Types node in the right panel, and use the File Types page to create the binary file types that you want to use with CVS.

   For help while using this page, press F1 or click **Help**.
6. Click **OK** to close the Preferences dialog.

To select CVS as the versioning system:

- Choose **Versioning > Version System: [...] > CVS**.

After you have configured JDeveloper and selected CVS as the versioning system of course, you are ready to make the connection to CVS so that you can view and access files in the CVS repository

6.4.1.2 How to Create a CVS Connection

Before you can work with a CVS repository through JDeveloper, you must create a connection to it. When you create a local CVS repository, you can choose to create a connection to the repository automatically

CVS connections are shown in the CVS Navigator. You can subsequently edit the connection details.

To create a CVS connection:

1. In the CVS Navigator (**View > CVS Navigator**) right-click the CVS node and choose New CVS Connection.

   The Create CVS Connection wizard is opened.

2. Complete the Create CVS Connection wizard.

   For help when using the wizard, press F1 or click **Help**.

To edit a CVS connection:

1. In the CVS Navigator (**View > CVS Navigator**) right-click the connection name and choose **Properties**.

   The Edit CVS Connection wizard is opened.

2. Use the wizard to make changes as required.

   For help when using this wizard, press F1 or click **Help**.

6.4.1.3 How To Import JDeveloper Project Files Into CVS

Before you can start using your JDeveloper project with CVS, you have to import the project files into the CVS repository. This copies all your folders and files to the CVS repository and places them under source control.

You import your project files into the CVS repository using the Import to CVS wizard.

To use the Import to CVS wizard:

1. Choose **Versioning > Import Module**. The Import to CVS wizard is displayed.
2. Complete the import as prompted by the wizard. For help when using this wizard, press F1 or click **Help**.

Before you can change any files, you have to copy them back to your machine, where you can work on them locally.

6.4.1.4 How to Check Out CVS Modules

This is a configuration task you perform when you first start to use JDeveloper with files and folders that are under CVS source control. You perform this task once, after (if necessary) importing your JDeveloper project into the CVS repository.

To check out modules from the CVS repository:

1. In the CVS Navigator, select the CVS module that you want to check out.

   Either:

   Choose **Versioning > CVS > Check Out Module**.

   or:

   from the context menu, select **Check Out Module**.

The Check Out from CVS dialog is displayed.

2. Complete the dialog. For help when using this dialog, press F1 or click **Help**.

6.4.2 How to Configure CVS For Use with JDeveloper

In addition to setting up JDeveloper to be able to use CVS, there are certain tasks you need to perform to make CVS usable with JDeveloper. Some of these tasks may be performed by your administrator. You should always check to make sure which of these tasks have been performed in your installation.

In general, you need a local CVS repository for storing files as you are working on them. You may also need to configure a secure shell (SSH) for communicating with CVS, and you may need to choose a character set. Finally, you will need to log in to CVS.

6.4.2.1 How to Create a Local CVS Repository

From within JDeveloper, you can create a new CVS repository on your local file system. This feature is available only if you are using external CVS client software, rather than the internal CVS client installed as part of the CVS extension to JDeveloper.

To create a local CVS repository:

1. Select **Versioning > Create Local Repository.**
2. In the Repository Folder box, enter the path of a directory where you want the new local repository to be created.

   You can specify or select an existing directory if it is empty, or you can specify a new directory. If the directory you have specified exists and is not empty, you will see a warning dialog telling you to specify an empty or new directory for the repository.
3. If you want to create a connection to the local repository that you are creating, make sure that the Create Repository Connection box is checked.

   The connection will be given a name in the form :local:{path}. If you later want to change this name, you can do so through the CVS Navigator: from the context menu of the connection name, open the properties dialog and, on the Name tab, overtype the existing name with a new one.
4. Click **OK**. You will see a confirmation dialog when the new local repository has been created.

6.4.2.2 How to Configure SSH (Secure Shell), CVS and JDeveloper

JDeveloper supports SSH Levels 1 and 2 as access methods for CVS repositories.

6.4.2.2.1 Configuring for SSH Level 1 (SSH) JDeveloper does not provide a direct way of using SSH Level 1 as an access method for the CVS repository. It is however possible to configure SSH Level 1 so that it can be used for remote shell access.

To configure SSH Level 1 to enable remote shell access:

1. Generate public and private keys using the command: `ssh-keygen`
2. Concatenate the `~/.ssh/identity.pub` public key file with `~/.ssh/authorized_keys` on the machine with the CVS repository.

Before running JDeveloper and attempting to use CVS with SSH Level 1, users should be explicitly authorized and the environment correctly configured. Follow the steps below to configure the environment correctly.

To configure the environment for SSH Level 1:

1. Set the `CVS_RSH` environment variable to the location of the SSH client.
2. At the UNIX command line, enter `ssh-agent {shell}`, and then press **Enter**.
3. At the UNIX command line, enter `ssh-add`, and then press **Enter**.
4. Start JDeveloper.
5. Select External as the CVS access method when using the CVS Connection Wizard.

6.4.2.2.2 Configuring for SSH Level 2 (SSH2) JDeveloper provides a direct way of using SSH2 as an access method for the CVS repository.

To use SSH2 for remote shell access:

1. On the JDeveloper CVS preferences page, set the CVS Client preference to **Internal to** JDeveloper **[...]**.
2. Start the CVS Connection Wizard.
3. While using the CVS Connection Wizard, on the Connection page, choose **Secure Shell via SSH2** as the Access Method. For more help at this stage, press F1 or click **Help**.
4. On the Connection page, click Generate SSH2 Key Pair. This opens the Generate SSH2 Key Pair dialog. For help using this dialog, press F1 or click **Help**.
5. After generating the SSH2 key files, an information dialog will appear that explains where to install the files.
6. Install the SSH2 key files as instructed in the dialog.
7. Complete the CVS Connection Wizard to create the CVS connection.

If you are using an internal CVS client, you can generate SSH2 key files at any time by choosing **Versioning > Administration > Generate SSH2 Key Pair**. If you are using an external CVS client, this menu option is unavailable.

6.4.2.3 How to Choose a Character Set (Local Client Only)

If your installation uses a local CVS client, you need to choose a character set.

For each CVS repository connection, you can choose the character set to be used for the encoding of files. The default is to use the character set specified by the platform/operating system.

You can change to the IDE default or to a specific character set through the Set Encoding dialog.

To choose a character set:

1. Select a connection in the CVS Navigator.
2. Clicking the right mouse button and choose **Set Encoding.**
3. Select the desired character set.

6.4.2.4 How to Log In to CVS

Some types of connection to a CVS repository require you to log in independently of making the connection. If you cannot access any CVS features even though a CVS connection exists, you need to log in.

To log in to a CVS repository:

1. In the CVS Navigator, select **Versioning > Log In**.

   If the Log In menu option is unavailable but the **Log Out** option is available, you are already logged in.

2. In the Log In To CVS dialog, enter your password. If you want your password to be remembered and supplied automatically when you connect to the CVS repository in future, check the Connect Automatically on Startup box.

3. Complete login by clicking **OK**.

6.4.2.5 How to Access Local Files with CVS

If JDeveloper finds a path to a CVS client on your machine, the JDeveloper CVS preferences will by default be set to use that CVS client (rather than the internal CVS client installed with JDeveloper). If no path to a CVS client is found, the preferences will be set to use the internal CVS client.

The internal CVS client cannot be used to access a local CVS repository (that is, one on your own machine). If you wish to access a local CVS repository, you must install a full client/server version of CVS onto your machine and set the JDeveloper CVS preferences accordingly.

If you wish to use an external CVS client, we recommend the following:

- CVSNT 2.0.58a for Windows platforms
- cvshome's CVS 1.11.9 for other platforms

> **Note:** You may already have a CVS installation that is client-only. This will not be able to access a local CVS repository, and you should install a full client/server version instead. If you are unable to expand the connections node in the CVS Navigator or open the list of modules from the Get Module List button in the CVS wizards, you probably have client-only CVS software that is attempting to access a local CVS repository. You can check which type of CVS installation you have by typing `cvs -v` at the CVS command prompt. A client-only installation will display (client) at the end of the version information line, whereas a client/server installation will display (client/server).

To access CVS through a firewall:

If you are accessing a CVS server through a firewall, you can connect to it if:

- the firewall allows TCP/IP communication on the CVS port, or
- you use a CVS client that supports HTTP Tunneling (for example, CVSNT).

If there is an authentication failure when you log in, try using the CVS command line to connect. If this fails, the connection may be being blocked by the firewall, and you should contact your network administrator.

If necessary, you can alter the value of the CVS root variable to support connection through a firewall.

6.4.2.5.1 Handling CVS File Types The CVS administrator has to configure the CVS repository for the automatic handling of binary files produced by JDeveloper, such as image file formats.

Where other file types are updated, CVS attempts to merge them. If you do not want is to occur, you must change the configuration of the CVS repository.

For more information about CVS, refer to the CVS documentation, or see the CVS website, `http://www.cvshome.org`. This is also where you can download CVS software.

6.4.3 How to Use CVS After Configuration

Once JDeveloper is configured and your project files are available in the CVS repository, you will most likely use a workflow that follows the basic sequence of update, checkout, modify, and commit. In addition, you may occasionally need to resolve edit conflicts, and merge the resulting file(s) into the repository.

6.4.3.1 How to Update a Project, Folder, or File in CVS

The CVS update operation updates your local files with data in the CVS repository. Alternately, you can choose to completely replace your local files with those held in the CVS repository.

You can update individual files (including project files), or you can update the entire contents of a project folder.

You can view the contents of the CVS repository through the CVS Navigator. The nodes under CVS Server unfold to reveal the structure and file content of the CVS repository. You can open a read-only version of a CVS repository file by choosing Open from its context menu. This will let you see what changes have been made to the files in the CVS repository since you checked out or last committed your local versions.

To update an individual file (including a project file):

1. Select the file(s) in the Application Navigator, and then choose **Versioning > Update**.
2. Set the options as required. For information about these options, press F1 or click **Help**.
3. To update all the files listed, click **OK**.

To update the contents of a project folder:

1. Select the project folder(s) in the Application Navigator and then, from the context menu, choose **Update Project Folders**.
2. Set the options as required. For information about these options, press F1 or click **Help**.
3. To update all the files listed, click **OK**.

To update files shown in the Pending Changes window:

1. With the Pending Changes window in Incoming Changes mode, select the files that you want to update.

   To obtain more information about the Pending Changes window, press F1 or click **Help**.
2. Click the **Update** button.

6.4.3.2 How to Edit and Watch Files in CVS

Editing and watching are available only when an external CVS client executable is used.

These procedures allow you to obtain and release an editor on a file, to know who else in your team is editing files, and to know who is watching for files to be edited. Two or more developers retain the ability to edit the same file at the same time.

To set up JDeveloper to use editing and watching:

1. Open the preferences page obtainable from **Tools > Preferences | Versioning | CVS**.
2. Ensure that External Executable is selected and that valid details are entered.
3. Select **Run CVS in Edit/Watch Mode**.
4. Open the preferences page obtainable from **Tools > Preferences | Versioning | CVS | General**.
5. Deselect **Automatically Make Files Editable**.

To obtain an editor on a file:

1. With the file selected in the Application Navigator, select **Versioning > Edit.**
2. Check that you want the operation to apply to all of the files highlighted in the file selection box.
3. To set up a watch for this file, select the Set Watch Actions checkbox and select a watch action from the drop-down list.
4. Click **OK**.

To release an editor on a file (to unedit a file):

This action reverses changes made in the current edit. Any local file modifications will be lost when the editor is released.

1. With the file selected in the Application Navigator, select **Versioning > Unedit**.
2. Check that you want the operation to apply to all of the files highlighted in the file selection box.
3. Click **OK**.

To turn on or turn off the file watching facility:

1. In the Application Navigator, select a project containing files about which you want to be notified.
2. Select **Versioning > Watch**.
3. In the Watch CVS Files dialog, choose **Turn On Watching** or **Turn Off Watching** from the Command Type drop-down list.
4. Click **OK**.

To add yourself to the list of people who receive notification of work done on files:

1. In the Application Navigator, select the project containing the files about which you want to be notified.
2. Select **Versioning > Watch**.

3. Check that you want the operation to apply to all of the files in the file selection box.
4. On the Watch Settings tab, choose **Add File Watch** as the Command Type from the drop-down list.
5. Optionally, check the Set Watch Actions checkbox and choose the particular actions that you want to be notified about.
6. Click **OK**.

To remove yourself from the list of people that receive notification of work done on files:

- Follow the procedure for adding yourself to the list (above), but choose **Remove File Watch** from the Command Type dropdown list.

To see who is watching for changes being made to files:

- Select **Versioning > Edit Notifications**.

The Edit Notifications window is opened. The Watchers tab shows the files that are being watched and the user(s) who are currently watching for changes.

To see who is currently editing files:

- Select **Versioning > Edit Notifications**.

The Edit Notifications window is opened. The Editors tab shows the files that currently have editors on them and the user(s) who have obtained those editors.

6.4.3.3 How to Commit Changes to CVS

Use these procedures to update the CVS repository with the latest version of the files you have been working on, and to add any new files to or remove any unwanted files from the CVS repository.

You can perform this on a single file, or in the context of a project. When in the context of a project, JDeveloper determines which files have changed since they were last committed and displays them as a list.

If you select a project to be committed that includes files that are not yet part of CVS version control, the Add Files to CVS message dialog will open. To obtain information about using this dialog, press F1.

You can view the current contents of the CVS repository through the CVS Navigator. The nodes under CVS unfold to reveal the structure and file content of the CVS repository. You can open a read-only version of a CVS repository file by choosing Open from its context menu. This will let you see what changes have been made to the files in the CVS repository since you checked out or updated your local versions.

To commit individual files shown in the Application Navigator:

1. Select the file(s) in the Application Navigator, and then choose **Versioning > Commit**.

   The Commit to CVS dialog is displayed with the file(s) listed.
2. Set the options on the Commit to CVS dialog as required.

   To obtain more information about the options, press F1 or click **Help**.
3. To update the listed file(s) in the CVS repository, click **OK**.

To commit the contents of project folders shown in the Application Navigator:

1. Select the project folder(s) in the Application Navigator and, from the context menu, choose **Versioning > Commit Project Folders**.

   If there are files in the project that are not under CVS control, you will be asked whether you want to add them.

   The Commit to CVS dialog is displayed with the folder(s) listed.

2. Set the options on the Commit to CVS dialog as required.

   To obtain more information about the options, press F1 or click **Help**.

3. To update the listed file(s) in the CVS repository, click **OK**.

To commit files shown in the Pending Changes window:

1. With the Pending Changes window in Outgoing Changes mode, select the files that you want to commit.

   To obtain more information about the Pending Changes window, press F1 or click **Help**.

2. Click the **Commit** button.

6.4.3.4 How to Merge Files in CVS

Use this procedure to merge two revisions of a file, where the revisions contain conflicting content. Conflicts are notified in the Pending Changes window: the outgoing status is "conflicts" or "conflicts on merge", and the Resolve Conflicts button is active.

To merge two revisions with conflicting content:

1. On the Outgoing tab of the Pending Changes window, select the revision that has conflicts and click the Resolve Conflicts button.

2. The merge tool is opened (as the Merge tab of the file editor).

   For help while using the merge tool, press F1.

   The merge tool has three panels. The left panel contains the content of the version in the repository. The right panel contains the content of the most recent local version. The center panel contains the results of the merge. In the margins between the panels are symbols representing suggested actions to resolve each conflict.

3. View the suggested actions for resolving the conflicts by reading the tooltip of the margin symbols.

   More suggested actions may be available from the context menus of the margin symbols.

4. Resolve the conflicts by implementing a suggested action in each case.

   Accepting an initial suggested action may cause the appearance of additional suggested actions.

   You can also make changes to the content of the center panel by typing into it.

5. To complete the merge, save the changes that have been made, using the **Save** button.

6.4.4 How to Work with Branches in CVS

CVS lets you define branches, used when development needs to be carried out separately from the main (or trunk) branch of a project. JDeveloper gives you access to CVS branches in your repository through the Tag, Branch and Merge menu.

In CVS, you can create a separate branch when you want to carry out specific work (such as bug fixes or specialized feature development) without any impact to the main set of files, also called the trunk.

Once you have created a branch, you interact with it as normally with CVS -- check out files, commit changes, etc. You can switch back and forth between branches, and you can merge the changes you have made to your branch back into the trunk.

CVS also lets you apply tags to specific branches, or to specific files in a branch (as well as generating a new tag for the branch you create, when you create it).

6.4.4.1 How to Create a New Branch

You create a new branch when you are beginning a project based on an earlier version of your code repository, such as for fixing bugs after a major release, or working on specific features for a subset of your customers.

To create a new branch:

1. In the CVS repository, select the file or folder on which you wish to base your new branch, then click the right mouse button.
2. Choose **Branch > Branch in CVS.**
3. Type in the branch name. JDeveloper converts the branch name to the default tag for the branch, by appending **_BASE** to the branch name as you type it.
4. Choose whether the branch source is the trunk or the working copy. If you select the trunk, the HEAD revision of every file is branched.
5. Click **Save**.

The base tag is applied before the branch is created, allowing you to specify these versions as a merge point in future.

You can also specify that you wish to create your new branch from an existing branch, by choosing the branch to use as the base.

To create a new branch from an existing branch:

1. Click **Details**.
2. Select the desired branch from the list of existing branches.

6.4.4.2 How to Use Branches in CVS

Branch selection is integrated into a number of CVS functions. You can switch branches or versions for files you are editing or have checked out; you can choose tags, branches, or version dates while updating the contents of your work area, as well as while you are checking out a CVS module.

6.4.4.2.1 How to Switch the Branch or Version You can switch the branch or version of a file you are editing, either from the JDeveloper Versioning menu or from the file or project's context menu.

To select a branch, version or date from the Versioning menu:

1. From the Versioning menu, choose **Tag, Branch or Merge > Switch Branch** or Version.
2. Click the chooser to display a list of branches or versions.
3. Select the branch or version you wish to use.
4. Optionally, click **Add Date** to specify a date to use.
5. Click **OK**.

To select a branch, version or date from the project's context menu:

1. Choose **Versioning > Switch Branch or Version**.
2. Click the chooser to display a list of branches or versions.
3. Select the branch or version you wish to use.
4. Optionally, click **Add Date** to specify a date to use.
5. Click **OK**.

6.4.4.2.2 How to Choose a Branch while Updating

When you are updating your content to capture the latest revisions to the repository, you have the option of branch (via its associated tag) at the same time.

To select a tag and branch while updating:

1. From the project's context menu, choose **Update Project Folders**.
2. In the Update from CVS dialog, check the box marked **Use Revision, Tag or Date**, then click the chooser icon.
3. Select a tag to use.
4. Optionally, click **Add Date** to specify a date to use.
5. Check any other boxes (**Overwrite Local Changes, Prune Empty Folders**, etc.) that you wish to apply to the current update, then click **OK**.

6.4.4.2.3 How to Choose a Branch While Checking Out

As with other CVS operations, tags and branches are integrated into the process of checking out a CVS module.

To choose a branch while checking out:

1. Click the right mouse button on the content in the Versioning Navigator to bring up the context menu, then choose **Check Out Module**.
2. Check the box labeled **Use Revision, Tag or Date**, then click the chooser to select a tag.
3. Select a tag. Optionally, you can click the **Add Date** button to specify a date. When you have made your selection, click **OK** to close the Tags dialog.
4. Choose any other options (**Force Match, Ignore Child Folders**, etc.), then click **OK** to close the Check Out from CVS dialog.

6.4.4.3 How to use Tags in CVS

Tags are a way of identifying branches, branch-specific content, or other content that you wish to identify and manipulate as a single logical group. You can tag files, folders, or projects. You can then later use these tags to identify branches, update files from a branch with a specific tag, and other operations.

You can select and browse tags from context menus as well as the Versioning > CVS > Tags menu. The availability of tags differs depending on the context of the operations you are performing on your content.

6.4.4.3.1 How to Add a Tag to a Project You can identify a project by adding a tag to it. You can then operate on this project by selecting the tag from any of the CVS menus that contain the tag chooser.

To add a tag to a project:

1. Select the project you want to tag.
2. Choose **Versioning > CVS > Tag, Branch and Merge > Tag**.
3. Type the tag you want to use, or click the chooser icon to browse the existing tags.
4. Optional: Choose **Use Revision, Tag or Date**, then type the tag or click the icon to browse the list.

6.4.4.3.2 How to Apply Tags While Updating a Project or File You can choose and apply a tag while using the Update from CVS dialog.

To select an existing branch, version or date from the Projects view:

1. From the project's context menu, select **Versioning > Tag**.
2. Choose **Use Revision, Tag or Date**.
3. Click the tag chooser icon.
4. Choose a tag from the list that appears.

6.4.4.3.3 How to Delete a Tag You can also delete a tag. Deleting a tag removes it from any resources to which you have applied it. Deleting the tag does not delete the content to which the tag was applied; it merely removes it from the list of available tags.

To delete a tag:

1. Select **Versioning > Tag, Branch and Merge > Delete Tag**.
2. Click the chooser icon. Choose the tag you wish to delete, then click **OK**.

In this context, only existing tag versions (regular non-branch tags) can be selected.

6.4.5 How to Work with Files in CVS

As a very general rule, working with files in CVS means checking out the latest version of a file, making your edits, and checking the file in with your changes. Occasionally, if you and a colleague have made edits to the same file, you may need to merge your changes to make sure your work is not lost. Other functions of CVS are also available, such as adding a new file or removing unused/obsolete files from the repository, but your general workflow will follow the checkout-edit-checkin pattern.

The file operations in CVS include refreshing the display of CS objects, adding and removing files, using templates, comparing files, replacing a file in CVS, viewing the history and status of a file, locking and unlocking files, and working with revisions and tags.

6.4.5.1 How to Refresh the Display of CVS Objects

The source control status of an object is indicated in the Application Navigator by an icon overlay, as listed in Table 6–1.

Table 6–1 CVS Object Status

Icon	Description
	The object has been copied from the CVS repository and added to your working files directory.
	The object is not under CVS source control, but may be added to the CVS repository.
	There were conflicts when the object (a file) was updated from the CVS repository. In this case, you have to manually edit the file to resolve all the points of conflict.
	The object has been scheduled for removal from the CVS repository with the next commit action.
	The object is out of synch with the CVS repository due to local or remote changes.
	The object is unmodified since it was last copied from the CVS repository.
	The object is unmodified since it was last copied from the CVS repository but is read-only.
	The package or node is a CVS sandbox folder.
	The apparent object may comprise several underlying objects, the statuses of which may not all be identical.

If the status of an object is changed outside JDeveloper, for example by checking in an object using external source control software, the new status might not immediately be shown in JDeveloper. To ensure that the status indicated in the Application Navigator matches the true status of the object in the source control system, you can perform a manual refresh.

To refresh the status of objects in JDeveloper:

- In the Application Navigator or CVS Navigator, click the refresh button.

6.4.5.2 How to Add and Remove Files

You can add a file to CVS only if it is part of a project that is already under CVS version control.

When you create a new file, for example a new class, it has to be added to source control before you can use the other CVS operations on it. The file is added to source control locally, and the CVS repository is not updated. The file is identified in the Application Navigator by the icon **+**.

To add a file to CVS through the Application Navigator:

1. Select the file in the Application Navigator and choose **Versioning > Add** (or, if the file is binary, **Versioning > Add as Binary**). JDeveloper usually recognizes binary files and adds (Binary) after the file name in the navigator. The Add to CVS dialog (or Add to CVS as Binary dialog) is displayed, with the file listed.
2. Click **OK**.

   The file will be added to the CVS repository when the next commit is performed.

To add files shown in the Pending Changes window:

1. With the Pending Changes window in Candidate Files mode, select the files that you want to add to source control.

   To obtain more information about the Pending Changes window, press F1 or click **Help**.
2. Click the **Add** button.

To remove a file from CVS:

When you remove a file from CVS it is removed from your local disk.

1. In the Application Navigator, select one or more files to be removed, then choose **Versioning > Remove**.
2. The Remove from CVS dialog is displayed with the files listed.
3. Click **OK**.

The file or files will be removed from the CVS repository when the next commit is performed.

6.4.5.3 How to Use CVS Templates

Many team environments require the developer to enter comments when a file is checked in. These comments might include bug report numbers, dependencies on other files, or some other explanatory information to be stored in connection with the file being committed to the repository.

JDeveloper lets you create and select templates for use with such comments. The templates are available from the Commit menu.

To create a new template:

1. Select **Tools > Preferences > Versioning > CVS > Comment Template**s.
2. Click **Add**.

To select a template:

1. Click on Choose a template or previous comment.
2. Select the template from the list.
3. Click **OK**.

To make a comment to a file being committed:

1. Click in the Comments box to select it.
2. Type the comment you wish to make with the file being committed.
3. Click **OK**.

6.4.5.4 How to Compare Files in CVS

Use these procedures to compare revisions of files that are under CVS source control. You can compare a file with its immediate predecessor, or you can compare with any of the file's previous revisions.

To compare a file shown in the Application Navigator:

1. From the context menu for the file, choose **Compare With**.
2. Select either **Previous Revision, Head Revision** or **Other Revision**.
3. If you are comparing with previous revisions, these are listed in the **Compare CVS File** dialog: Select the file that you want to compare with.

   If there are no differences, a message is displayed. Otherwise the Compare tool is displayed.

To compare a file shown in the Pending Changes window:

You can compare a file in the Pending Changes window either with a previous revision or with the HEAD revision, depending on which mode the window is in. To obtain more information when using the Pending Changes window, press F1.

- With the window in Outgoing Changes mode, select the file to be compared, then select the **Compare with Previous Revision** button.
- With the window in Incoming Changes mode, select the file to be compared, then select the **Compare with Head Revision** button.

If there are no differences, a message is displayed. Otherwise the Compare tool is displayed.

6.4.5.5 How to Replace a File with a CVS Revision

Use this procedure to replace a file with the base or head revision, or with a file with a specific revision number or tag. The head revision is the latest one. The base revision is the revision from which the one you are currently working on originated.

To replace a file with a CVS revision:

1. In the navigator, select the file to be replaced.
2. Do one of the following:
   - To replace with the base revision, choose **File > Replace With > Base Revision**. The Replace With Base Revision dialog opens.
   - To replace with a specific revision number or tag, choose **File > Replace With > Tagged Revision**. The Replace With Tagged Revision dialog opens.
   - To replace with the head revision, choose **File > Replace With > Head Revision**. The Replace With Head Revision dialog opens.
3. Check that the file that you want to replace is shown in the dialog.
4. When replacing with a specific revision number or tag, enter the revision number or tag into the text box on the dialog.
5. To replace the file, click **OK**.

6.4.5.6 How to View the History and Status of a File

The history and status of a file will tell you what has been done to it, and what has been done to it last. This can help you make the determination of what you need to do to bring the file up to date, or to begin making your own modifications.

Use this procedure to open the History Viewer and view the history of CVS files.

To view the history of a project or file:

- With the project or file selected in the Application Navigator, choose **Versioning > Version History** from the context menu.

  For more information while using the History Viewer, press F1.

Use this procedure to check the status of a file that is under CVS source control. You can also refresh the status of files under CVS control.

To view the status of a file:

1. With the file selected in the Application Navigator, open the context menu and select **Versioning > Properties**.
2. Select the Versioning tab. The status of the file is the first item on the tab.

Possible statuses are:

- Changed locally - the file has been locally modified since it was copied from the repository.
- Changed in repository - the file has been modified by another user since it was copied from the repository.
- Locally removed - the file will be removed during the next commit.
- Locally added - the file will be added during the next commit.
- Up-to-date - the file is up-to-date with the latest CVS repository revision.
- File has conflicts - these may have resulted from a file update or commit action. If necessary, consult your CVS administrator for assistance.
- Needs merge or needs patch - the file has been updated externally, for example, by another user.
- Modified - the file previously had merge conflicts, but the timestamp has changed since.

6.4.5.7 How to Lock and Unlock Files

> **Note:** The locking of files is not supported in newer releases of CVS client software and this facility may be removed in future releases of JDeveloper.

You can choose to prevent other users working on a file while you are working on it yourself. This is not normally considered necessary, because CVS can usually reconcile differing versions of files as they are being committed to the CVS repository. The JDeveloper compare and merge facilities will reconcile differing versions of files automatically, or present you with a tool for doing so manually if there are serious conflicts.

You may want to ensure that a file is worked on only by you, until you have finished working on it. This might be because a file is in binary format and therefore inherently

difficult to merge. In this case, you can lock the file that you want to work on. The file is locked in the CVS repository, and other users are prevented from accessing it. When you want to let others work on the file, you unlock it.

To lock files in CVS:

1. With the file or files that you want to lock selected in the Application Navigator, choose **Versioning > CVS> Administration > Lock**.
2. Check that you want the operation to apply to all of the files highlighted in the file selection box.
3. Click **OK**.

To unlock files in CVS:

1. With the file or files that you want to lock selected in the Application Navigator, choose **Versioning > CVS> Administration > Unlock**.
2. Check that you want the operation to apply to all of the files highlighted in the file selection box.
3. Click **OK**.

6.4.5.8 How to Work with Revisions and Tags

These procedures allow you to work with revisions and tags.

To open a CVS file revision:

This procedure will obtain a revision of a file from the CVS repository so that you can view it or save it locally.

1. With the file selected in the Application Navigator, choose **Versioning > Open Revision**.
2. Set the options on the dialog as required. To obtain descriptions of the options, press F1 or click **Help**.
3. Click **OK**.

To assign CVS tags:

This procedure will assign symbolic tags to the local CVS revisions of selected files.

1. In the Application Navigator, select a single file, a project or a workspace. If you select a project or a workspace, all the files within the project or workspace will be selected for tagging.
2. Choose **Versioning > Tag > Tag**.
3. Check that you want the operation to apply to all of the files highlighted in the file selection box.
4. Enter a name for the tag in the Tag Name box.
5. Set the other options as required. To obtain descriptions of the options, press F1.
6. Click **OK**.

To delete CVS tags:

This procedure will delete symbolic tags from the local CVS revisions of selected files.

1. In the Application Navigator, select a single file, a project or a workspace. If you select a project or a workspace, the tag will be deleted from all the files within the project or workspace.
2. Choose **Versioning > Tag > Delete Tag**.
3. Check that you want the operation to apply to all of the files highlighted in the file selection box.
4. Enter the name of the tag in the Tag Name box.
5. Click **OK**.

To view CVS tags:

This procedure will display a dialog containing information about any existing tags that have been applied to the file revision.

1. From the context menu of the file, choose **Versioning > Properties**.
2. Select the Versioning tab. The sticky tag, date and options (if any) are shown, as is a list of existing tags for the file revision.

To reset CVS tags:

This procedure will remove any sticky tags or dates that have been applied to the selected files and reset them to the HEAD revision.

1. In the Application Navigator, select the file or files whose tags you wish to reset.
2. Choose **Versioning > Tag > Reset Tag**s.

6.4.6 How to Use External Tools and Export Features

At times, you may wish to use CVS with external tools (for comparing versions of files to be merged, for example) or to create and apply patches for projects you are developing with files under CVS control. Additionally, you may find it necessary to export a CVS module, or to copy the CVSROOT value to the clipboard. These procedures explain how.

6.4.6.1 How to Use an External Diff Tool with CVS

JDeveloper has an integrated compare viewer that works well for most circumstances. However, you may prefer to use another compare tool or the simple output from CVS DIFF. JDeveloper lets you integrate third party tools and applications. This procedure describes how to use the External Tools support in JDeveloper to integrate external compare viewers.

To integrate CVS DIFF:

1. In JDeveloper, select **Tools > External Tools**.
2. Click **Add**. This opens the Create External Tool wizard.
3. On the External Program Options page, enter the following information:

Program Executable
The location of your CVS installation (for example `c:\cvsnt\cvs.exe`) or just `cvs`

Arguments

```
-d ${cvs.root} diff ${file.name}
```

Alternate arguments

```
-d ${cvs.root} diff -r ${cvs.revision} -r ${cvs.second.revision}
${file.name}
```

Run Directory
`${file.dir}`

Enter the alternate arguments if you want to integrate a tool that compares two specific CVS revisions when the history tool is visible.

4. On the Display page, enter a caption for the diff tool (for example CVS Diff with Repository) in the Caption for Menu Items box.
5. On the Integration page, choose how you want the diff tool to be integrated into JDeveloper. For example, select the Tools Menu, Navigator Context Menu, and Code Editor Context Menu items.
6. On the Availability page, select **When a File is Selected or Open in the Editor**.
7. Click **Finish**.

To integrate a third party diff utility:

You can use external tools macros to view differences between two revisions in the history tool using a third party utility such as Araxis Merge. The following steps will install a menu item to invoke Araxis Merge. For other utilities, consult the documentation of the utility to determine which command line arguments need to be passed in.

1. In JDeveloper, select **Tools > External Tools**.
2. Click **Add**. This opens the Create External Tool wizard.
3. On the External Program Options page, enter the following information:

Program Executable The path to the third party tool (for example `c:\araxismerge\compare.exe`)

Arguments

```
/wait /title1:"${file.name} revision ${cvs.revision}"
/title2:"${file.name} revision ${cvs.second.revision}" /2
${cvs.revision.file} ${cvs.second.revision.file}
```

4. On the Display page, enter a caption for the third party tool (for example Araxis Diff) in the Caption for Menu Items box.
5. Complete the remainder of the wizard as required. For help when using the wizard, press F1 or click **Help**.
6. Click **Finish**.

To integrate other CVS commands:

You can take advantage of the supplied external tool macros to easily integrate other CVS commands into JDeveloper. An example is the CVS annotate command (sometimes referred to as "blame"), which shows a summary of who changed each line of code and when the change was made. To integrate a tool for CVS annotate, set the following options in a new tool:

1. In JDeveloper, select **Tools > External Tools**.
2. Click **Add**. This opens the Create External Tool wizard.
3. On the External Program Options page, enter the following information:

Program Executable
The path to the CVS executable (for example, `C:\cvs\cvs.exe`)

Arguments
`-d ${cvs.root} annotate ${file.name}`

Run Directory
`${file.dir}`

4. Complete the remainder of the wizard as required. For help when using the wizard, press F1 or click **Help**.
5. Click **Finish**.

6.4.6.2 How to Export a CVS Module

You use the CVS Export wizard to export the revisions of files for a module, creating a deployment-ready file structure.

To use the CVS Export wizard:

1. Choose **Versioning > Export Module**. The CVS Export wizard is displayed.
2. Complete the export as prompted by the wizard. To obtain more information when working with the wizard, F1 or click **Help**.

The files are exported to the location you have specified.

6.4.6.3 How to Copy the CVSROOT Path to the Clipboard

You can copy the path of the CVSROOT from a node in the CVS Navigator to the Clipboard, for use in other applications.

To copy the CVSROOT path to the Clipboard:

1. In the Connection Navigator, right click the connection name.
2. From the context menu, choose **Copy CVSROOT**.

The full path of the CVSROOT is copied to the Clipboard, from where you can paste it into another application.

6.4.7 How to Create and Apply Patches

You may wish to record the changes between two revisions of a file, then apply those changes to a third file. You do this by creating a patch and then applying it.

To create a patch:

This generates a patch comprising the differences between a controlled revision of a file and a revision of the file held locally.

1. In JDeveloper, open the file for which you want to create a patch.
2. Click the History tab.

   The History view lists all the revisions of the file. In the lower portion of the History view, the left pane shows the contents of a local revision, and the right pane shows the contents of the controlled revision.
3. Select the revision combination for which you want to create a patch.
4. From the context menu, choose **Generate Patch**.

The Select Patch Context dialog may open. For help while using this dialog, press F1.

The Generate Patch dialog opens. Complete the dialog as required. For help while using the dialog, press F1.

To apply a patch:

1. In the navigator, select the resource to which you want to apply a patch.

   The resource can be an application, a project, or a source file.

2. Select **Versioning > Apply Patch**.

   If you chose to apply a patch to a project, the Select Patch Context dialog opens, through which you should specify whether you are applying a project file (.jpr) patch, or whether you are updating the contents of a project.

   The Apply Patch dialog is opened.

3. In the grid at the top of the Apply Patch dialog, check that the target resources are correctly identified.

4. Choose the source of the patch. For more information about this and the other options on the dialog, press F1 or click **Help**.

5. Click **Preview**. This opens the Apply Patch Preview window, in which you can accept or reject particular changes. For more information about the options in the Apply Patch Preview window, press F1.

6. To apply the patch, click **OK**.

6.5 Using Perforce with Oracle JDeveloper

Perforce uses a local directory structure to receive files that are going to be placed under formal source control. This location is called the "Perforce client workspace". Files created in (or moved into) JDeveloper must be stored in this location. Once files are in your Perforce client workspace, you bring them fully under source control by submitting them to a central location called the "Perforce depot". Files must be submitted to the Perforce depot before they can be versioned and accessed by other users.

6.5.1 How to Set Up Perforce with JDeveloper

Before using Perforce with JDeveloper, in addition to downloading the Perforce extension, you need to install a number of Perforce features so that they are available to JDeveloper. Once installed, you configure JDeveloper and connect to the Perforce client workspace. Finally, you need to bring your working files under Perforce control so that they are available from within JDeveloper while using Perforce.

6.5.1.1 How to Install Perforce Components for Use with JDeveloper

There must be at least one Perforce server installed, on a machine that is accessible to the intended JDeveloper users. If a Perforce server installation does not already exist, obtain the necessary software (for example, from www.perforce.com) and install it in accordance with Perforce's instructions. Record the identity of the machine on which the Perforce server software has been installed: you will need this when you connect to it through JDeveloper.

Perforce Client Installation

You must install the Perforce client application on the machines that contain (or that will contain) JDeveloper. The Perforce client application can be installed from the same software as the server software, obtainable from www.perforce.com. The installation must include the "Command Line Client (P4)" and, for Windows installations, "Visual Merge for Windows (P4WinMrg)".

When you first run the Perforce client application, you will be required to create a Perforce client workspace. The Perforce client workspace is where the working copies of files under Perforce control will be stored. You can use the JDeveloper default directory as the Perforce client workspace, whether or not it already contains JDeveloper files. The JDeveloper default directory is `<installation_directory>\jdev\mywork`. Alternatively, you can accept the default Perforce client workspace, or specify one of your own. In these cases, you should note the location you have used, because you will need to specify it when creating applications and projects in JDeveloper.

f you set up passwords in the Perforce client application, you will also need to use them when connecting to Perforce through JDeveloper.

JDeveloper Installation

JDeveloper must be installed in the normal way. Each installation of JDeveloper can act as a client application for Perforce. You can install JDeveloper on every machine that you wish to be a Perforce client, or you can use a mixture of JDeveloper installations and Perforce's own client applications. The JDeveloper and Perforce client applications will work together in a seamless manner. In addition to the JDeveloper embedded support for Perforce, you will also be able to access a Perforce client application through the JDeveloper interface.

6.5.1.2 How to Configure JDeveloper for Use with Perforce

Before you can configure JDeveloper to use Perforce, you must have installed the Perforce server and client software.

To configure JDeveloper for use with Perforce:

1. Choose **Tools > Preferences**, then select Extensions in the left pane of the Preferences dialog.
2. In the right pane, make sure that Versioning Support n is checked, then click Configure.
3. Ensure that Versioning Support for Perforce *n* is checked.
4. In the left pane of the Preferences dialog, open the Versioning node and then the Perforce node. The main Perforce preferences panel is shown. Other Perforce preferences panels are shown when you click on the items beneath the Perforce node.
5. Make changes to the preferences as required. For more information about the specific preferences, press F1.
6. Click **OK** to close the Preferences dialog.

To select Perforce as the version system:

- Choose **Versioning > Version System: [...] > Perforce**.

6.5.1.3 How to Connect to Perforce

Before Perforce operations become available within JDeveloper, you must connect to Perforce.

To connect to Perforce manually:

1. Choose **Versioning > Connect to Perforce**.

   The Connection dialog is opened. The username, port and client information should have been derived automatically and should now appear in the Connection dialog.

2. If not already present, enter the correct username, port and client information.
3. If the Perforce server has been set up with password protection, enter the password. (If you want the password to be remembered for the next time you make a connection, check the **Remember Password** box.)
4. If you want to test the connection to the Perforce server, click the Test Connection button. The results will be displayed in the rectangular text area.
5. To complete the connection, click **OK**.

To connect to Perforce automatically when you start JDeveloper:

1. Choose **Tools > Preferences**, then select the Versioning node and then the Perforce node.
2. Check the **Connect Automatically on Startup** box.
3. To close the Preferences dialog, click **OK**.

6.5.1.4 How to Make Multiple Connections to Perforce

In some development environments, you may need to make more than one connection to Perforce. For example:

- Your organization uses one Perforce server for development and another Perforce server for test.
- You wish to connect using two different Perforce clients.
- You wish to use different Perforce user IDs.

The Perforce extension to JDeveloper permits all these operations. You begin by giving each Perforce connection a name as you create it.

To create a named Perforce connection:

1. Choose **Versioning > Connect to Perforce**.

   The Connection dialog is opened. The username, port and client information should have been derived automatically and should now appear in the Connection dialog.

2. If not already present, enter the correct username, port and client information.
3. Enter a name to use for this Perforce connection. Make sure it is different from any other Perforce connection that you currently have open.
4. If the Perforce server has been set up with password protection, enter the password. (If you want the password to be remembered for the next time you make a connection, check the **Remember Password** box.)

5. If you want to test the connection to the Perforce server, click the Test Connection button. The results will be displayed in the rectangular text area.
6. To complete the connection, click **OK**.

Note that your Perforce changelist will display the connection that applies to each file in the changelist.

6.5.1.5 How to Bring Files Under Perforce Control

Files that you create within JDeveloper, or files that you bring into JDeveloper from outside, must be brought under Perforce control before you can use the JDeveloper Perforce versioning facilities with them.

If you have an existing JDeveloper project that you wish to bring under Perforce control, use the Import to Perforce wizard.

To put individual JDeveloper files under Perforce control:

1. Select the files in the navigator and choose **Versioning > Open for Add**.

   The files can be your work files, or they can be the application and project files used by JDeveloper.

   The Add Files to Perforce dialog is displayed with the files listed.
2. If you wish to lock the files, making them unavailable to others for editing, check the Lock Files box.
3. To add the files to Perforce control, click **OK**.

   The files are now shown in the navigator with a red cross, meaning that they are stored in your Perforce client workspace but not yet in the Perforce depot. Files must be added to the Perforce depot before they can be versioned and accessed by other users.
4. To add files to the Perforce depot, select the files in the navigator and choose **Versioning > Submit.**

   The Submit Files dialog is displayed with the files listed.
5. Add your versioning comments in the Comments box.

   You will later be able to see these comments when viewing the list of versions of a particular file.
6. To submit the files to the Perforce depot, click **OK**.

   The files are now shown in the navigator with a green dot, indicating that they are known to the Perforce depot and are up to date.

To bring files created outside JDeveloper under Perforce control:

1. Copy or move the files into an existing `\src` directory under the JDeveloper file storage directory (which should be the same as the Perforce client workspace).
2. Close and reopen JDeveloper.

   The files should now appear in the navigator, within the project whose `\src` directory you used. The files are marked with a white-on-blue diagonal cross, showing that they are known to JDeveloper but not under source control.
3. Bring the files under Perforce control as described in the previous procedure.

6.5.1.6 How to Import JDeveloper Files Into Perforce

Before you can start using existing JDeveloper project and source files with Perforce, you have to import them into your Perforce client workspace. Once they are in your Perforce client workspace, you bring them fully under source control by submitting them to the Perforce depot.

You import JDeveloper project and source files into your Perforce client workspace using the Import to Perforce wizard.

To use the Import to Perforce wizard:

1. If you have not already done so, connect to Perforce by choosing **Versioning > Connect to Perforce**.
2. In the Application Navigator, select the JDeveloper project that you want to bring under Perforce control.
3. Choose **Versioning > Import Project**. The Import to Perforce wizard is displayed.
4. Complete the import as prompted by the wizard. To obtain more information when working with the wizard, press F1.

   The project and files will be shown in the navigator. If you have chosen to display overlay icons, these will indicate the current source control status of the files.
5. To bring the files fully under Perforce source control, submit them to the Perforce depot.

6.5.2 How to Work with Files in Perforce

Perforce provides features for creating and applying patches—methods for determining changes between two revisions of a file, and then applying those changes to a third file. In addition, Perforce contains features for exporting the details about repository connections, as well as files in the repository.

6.5.2.1 How to Synchronize Local Files With the Controlled Versions

Another person may edit a file through their Perforce client and submit their changes to the Perforce depot. This will cause your copy of the file to become out of date compared with the controlled version.

To test that your view is showing the latest file statuses:

- Choose **View > Refresh**.

  A file that is out of date with the controlled version is shown with an exclamation point icon.

To bring your files up to date compared with the controlled version:

1. From the Connection drop-down list, select the preferred Perforce connection (if you have more than one) for this changelist.
2. Select the files in the navigator and choose **Versioning > Sync**.

   The Sync Files dialog is displayed with the files listed.
3. Complete the dialog.

   For more information about the dialog options, press F1.
4. To synchronize the files, click **OK**.

Your local files are replaced with copies of the controlled versions. The icon shown next to the files changes to a green dot.

6.5.2.2 How to Synchronize Files With the Perforce Navigator

The Perforce Navigator lets you browse the Perforce depot and update your working directory from content at the depot. Using the navigator, you can select folders or files to sync to your client workspace, downloading content from the Perforce Server to your computer. If new files are detected in the depot, you have several options for handling them.

If you opens a connection node and no connection has been made, Perforce displays the connection dialog.

To synchronize your files using the Perforce Navigator:

1. Expand the content under Perforce in the Versioning Navigator, selecting the folders and/or files you wish to synchronize. When you expand to the level of the project you're working on, right-click the file or folder, and then select **Sync From Depot**. This displays the Sync From Depot dialog.
2. The project you selected displays in the Name pane of the Sync From Depot dialog. Below that are fields you can select or specify:

   Head Revision

   Synchronize to the Head revision of your project. If you select this, the Sync From Depot dialog displays the **Force sync** checkbox. Select Force Sync if you wish to download the depot content to your working directory regardless of the contents of each (for example, if you know you want to start with a clean download of the depot's contents).

   Revision Number

   Select this to synchronize to a specific revision number. The Sync From Depot opens the Target field; use the Target field to type the revision number to which you wish to synchronize your local working copy.

   Changelist

   Select this to synchronize to a specific changelist. The Sync From Depot opens the Target field; use the Target field to type the name of the change list from which you wish to synchronize your local working copy.

   Label Name

   Select this to open files with a specific label (typically, used to identify a specific branch). The Sync From Depot opens the Target field; use the Target field to type the name of the label from which you wish to synchronize your local working copy.

   Date

   Select this to specify a date (and, optionally, time) from which you wish to synchronize your local files. The Sync From Depot opens the Target field; use the Target field to type the date (in either yyyy/mm/dd or yyyy/mm/dd:hh:mm:ss format) of the files from which you wish to synchronize your local working copy.

   Choose the field that applies to your current project, then click **OK**.
3. If the depot contains files that do not exist in your source, Perforce tells you that new files were detected, and lists the following options:

   Open files in active project

Copy the files, and open them in the project you have selected.

Create new project from files

Creates a new project, using the files Perforce has detected.

Open editors onto files

Open the files in editor windows, so that you can review them and determine the best resolution (keep, rename, discard, or modify).

Do not open files

Leaves the files unopened, without copying them from the depot to your working directory.

6.5.2.3 How to Filter Files By Perforce Workspace

If you have a very large number of files in your Perforce depot, it can be much easier to navigate to the files you're working on by filtering files in the Perforce workspace. You can do this by setting things up in the Perforce client, and then displaying the filtered view in JDeveloper.

Filtering files in Perforce (specifically, p4v) requires making sure that you are viewing the Depot Tree, then select the **Filter icon > Tree Restricted to Workspace View**.

To filter files in JDeveloper:

- **Version Navigator > Perforce > Connection name > Context menu - Filter by Client Workspace.**

You will only see a difference in the JDeveloper Version Navigator if the Perforce client has a rule that restricts the Perforce workspace. (In p4v, the rules are shown and set in the View field of the Workspace dialog for the selected workspace.) You could restrict the workspace view in your p4v client with a rule like the following:

```
//depot/JDeveloper_1013/... //<client name>//JDeveloper_1013
```

In JDeveloper, if you select **Filter by Client Workspace**, the navigator would be filtered so only `//depot/JDeveloper` is shown.

6.5.2.4 How to Edit Files

By default, you can start editing a file under Perforce control just by opening it from the navigator. While the Perforce server is being contacted, you may experience a delay before you can type into the opened file. If you would prefer files to remain closed until you manually open them for editing, set the Automatically Open Files for Edit preference to off. The following procedure works whichever way the preference is set.

To edit a file under Perforce control:

1. Select the file in the navigator and choose **Versioning > Open for Edit**.

   The Open Files for Edit dialog is displayed with the file listed.

2. If the file is out of date with the controlled version and you wish to edit the controlled version, check the **Sync files to** box.

   If you do not obtain the controlled version before editing the file, you may create a conflict between your file and the version in the Perforce depot. You will then have to resolve the conflict before your changes can be accepted into the controlled version.

3. If you wish to lock the file, check the **Lock Files** box.

   Locking a file means that others can edit the file but cannot submit the file until the person who applied the lock has released it.

4. To make the file editable under Perforce control, click **OK**.

   The file will be indicated to Perforce as editable. A red check mark is added to the file's icon in the navigator.

5. To edit the file, choose **Open** from the file's context menu.

6. Make your changes and save the file.

   You can also close the file if you wish.

The changes that you made to the file are now saved in your Perforce client workspace. To add your changes to the controlled version of the file and make them available to other users, you must now submit them.

6.5.2.5 How to Submit Changed Files to the Perforce Depot

Any changes that you make to a file are initially saved in your Perforce client workspace. To add these changes to the controlled version of the file and make them available to other users, you must submit them. In the following procedure, if the Submit menu option is unavailable, it is because there are unresolved conflicts between your copy of the file and the one in the Perforce depot. Before proceeding, you will have to resolve the conflicts or revert the file to a non-conflicting version.

To submit changes to the Perforce depot:

1. With the file selected in the navigator, choose **Versioning > Submit**.

   The Submit Files dialog is displayed with the file listed.

2. Add your versioning comments in the Comments box.

3. To submit the files to the Perforce depot, click **OK**.

   The file is now shown in the navigator with a green dot, indicating that it is up to date compared with the version in the Perforce depot.

6.5.2.6 How to Resolve Conflicts in File Versions

If there is a conflict between your copy of the file and the one in the Perforce depot, the icon next to the affected file will include an exclamation point. You will not be able to submit such a file to the Perforce depot. To overcome this problem, you should either revert to a non-conflicting version of the file, or resolve the conflict. You can resolve the conflict using the locally installed Perforce client application, or another merge tool of your choice, as specified on the Preferences page (**Tools > Preferences > Versioning > Perforce > Version Tools**).

To revert to a non-conflicting file version:

- Select the file in the navigator and choose **Versioning > Revert**.

To resolve conflicting file versions (assumes use of Perforce merge tool):

1. Open the Perforce client by choosing **Versioning > Resolve.**

2. In the pending changelists for the client, identify the change.

3. Resolve the conflict using the Perforce tools.

If you cannot automerge the conflicts, run the merge tool and use its facilities to create a definitive version from the conflicting data.

4. Accept the merge.
5. Submit the merge.
6. In JDeveloper, use **View > Refresh** to obtain the green dot on the file.

   The file will still be marked as open for edit.
7. Submit the file.

6.5.2.7 How to Resolve Conflicts in File Versions

If there is a conflict between your copy of the file and the one in the Perforce depot, the icon next to the affected file will include an exclamation point. You will not be able to submit such a file to the Perforce depot. To overcome this problem, you should either revert to a non-conflicting version of the file, or resolve the conflict. You can resolve the conflict using the locally installed Perforce client application, or another merge tool of your choice, as specified on the Preferences page (**Tools > Preferences > Versioning > Perforce > Version Tools**).

To revert to a non-conflicting file version:

- Select the file in the navigator and choose **Versioning > Revert.**

To resolve conflicting file versions (assumes use of Perforce merge tool):

1. Open the Perforce client by choosing **Versioning > Resolve**.
2. In the pending changelists for the client, identify the change.
3. Resolve the conflict using the Perforce tools.

   If you cannot automerge the conflicts, run the merge tool and use its facilities to create a definitive version from the conflicting data.
4. Accept the merge.
5. Submit the merge.
6. In JDeveloper, use **View > Refresh** to obtain the green dot on the file.

   The file will still be marked as open for edit.
7. Submit the file.

6.5.2.8 How to Refresh the Status of Files under Perforce Control

The source control status of a file is indicated in the JDeveloper navigators by icon overlays, as listed in.

Table 6–2 Perforce Status Icons

Icon	Meaning
	The file is in the Perforce client workspace but is not yet submitted to the Perforce depot.
	The file will be deleted when next submitted to the Perforce depot.
	The file is out of date compared with the Perforce depot.

Table 6–2 (Cont.) Perforce Status Icons

Icon	Meaning
	The file is up to date compared with the Perforce depot.
	The file is open for edit.
	The file is locked.

If the status of a file is changed outside JDeveloper, for example by using a Perforce client application, the new status might not immediately be shown in JDeveloper. To ensure that the status indicated in the Application Navigator matches the true status of the file in the source control system, you can perform a manual refresh.

To refresh the status of files in JDeveloper:

- Select **View > Refresh**.

6.5.2.9 How to Delete Files

If you wish to delete a file that it is under Perforce control, you should do so using the Perforce facilities within JDeveloper or the Perforce client application. You should not use the Erase From Disk or Delete commands to delete a file that is under Perforce control, as this may cause versioning problems.

To delete a file under Perforce control:

1. Select the file in the navigator and choose **Versioning > Open for Delete**.

   The Delete Files dialog is displayed with the file listed.

2. Click **OK**.

   The file is deleted from the local file system. A black diagonal cross is added to the file's icon in the navigator.

If you need to retrieve a file that has been deleted, you will need to use the Perforce client. To do this, select **Versioning > Perforce > Launch Perforce Client**.

6.5.3 How to Work with Changelists

In Perforce, changelists let you group files together to simplify operations. Once files are grouped in a changelist, you can check them out and submit them all in a single operation.

In Perforce, changes are submitted to a Perforce repository using a changelist. This lets you group changes to several files into a logical unit, and then submit this unit to the Perforce repository in one operation.

You can have more than one changelist. You may find it useful to create changelists for specific projects, for related groups of files, or for any other grouping of files that you find create a logical unit, based on the way you and your team work. You can also move files from one changelist to another.

In general, you use changelists by following this workflow: create a changelist, add files to your changelist, edit your files and submit your changelist with the edited files.

You can also browse existing changelists through the Changelist Browser. The Changelist Browser also lets you create, submit, and move files between changelists. If the submit operation fails on any file in the changelist, then the whole changelist fails. This means the Perforce repository is left in a consistent state.

6.5.3.1 How to Create a Perforce Changelist

A Perforce changelist lets you manipulate a number of changed files and folders in a single operation, simplifying the process when you have several files that you have been working on.

To create a Perforce changelist:

1. From the Versioning menu, select **Perforce > Create Changelist.**
2. From the Connection drop-down list, select the preferred Perforce connection (if you have more than one) for this changelist.
3. Select the files to be added to the changelist, or click **Select All** to add all displayed files to this changelist.
4. Add comments to this changelist, if desired. You can choose a previous comment (with the option of editing it if necessary), or you can select your comment template.
5. When you have set up the changelist as desired, click **OK**.

6.5.3.2 How to Annotate a Perforce Revision or Changelist

Annotating a Perforce revision or changelist lets you store the Perforce revision or changelist as a comment linked to every file in the revision. When you modify these files later in Perforce, you can view the sequence of revisions or changelists to these files, as annotations to the files.

To add annotations to a changelist:

1. From the Versioning menu, select **Versioning > Perforce > Perforce Pending Changelists**.
2. Select the changelist to view by clicking the **Use Changelist** selector.

Any previous annotations will be visible in the Comments field of the changelist.

6.5.3.3 How to Add Files to a Perforce Changelist

A Perforce changelist lets you manipulate a number of changed files and folders in a single operation, simplifying the process when you have several files that you have been working on. When you add files to Perforce, you can select the changelist to which these files will be added at the same time, through the Open for Add menu.

To add files to a changelist:

1. From the Versioning menu, select **Perforce > Open for Add**.
2. Select the changelist to use by clicking the Use Changelist selector.

6.5.3.4 How to Submit a Perforce Changelist

Once you have made a series of edits to your files, you are ready to submit them in Perforce. If you have created a changelist, you can submit all the files on that changelist in a single operation, or select just the ones you have edited and submit them.

To select and submit the files in a changelist:

1. From the Versioning menu, select **Perforce > Submit Changelist**.
2. Enter a description of the changes you have made in the Description field.
3. Check the files you wish to submit. Use the **Select All** and **Deselect All** buttons if required.

6.5.3.5 How to Use the Changelist Browser

The Changelist Browser lets you see, at a glance, the state of all the pending changelists in your Perforce repository. Each pending changelist is shown with its name, description, and contents. The default changelist is always shown at the top of the browser. Under each changelist, you can browse the files that are associated with that changelist. Additionally, the Perforce connection and client are displayed at the top of the browser.

From the Pending Changelist browser, you can create and submit changelists, move files between changelists, and refresh the browser.

To create a changelist with the Changelist Browser:

1. From the Versioning menu, select P**erforce > Create Changelist**.
2. From the Connection drop-down list, select the preferred Perforce connection (if you have more than one) for this changelist.
3. Select the files to be added to the changelist, or click **Select All** to add all displayed files to this changelist.
4. Add comments to this changelist, if desired. You can choose a previous comment (with the option of editing it if necessary), or you can select your comment template.
5. When you have set up the changelist as desired, click **OK**.

To submit a changelist:

1. From the Versioning menu, select **Perforce > Submit Changelist**.
2. Enter a description of the changes you have made in the Description field.
3. Check the files you wish to submit. Use the Select All and Deselect All buttons if required.

To move files between changelists:

1. Click the right mouse button the file in the Changelist Browser and select **Move File to Changelist**.
2. Select the changelist to which you wish to move this file, then click **OK**.

You can also refresh the changelist browser by pressing **F1**.

6.5.4 How to Create and Apply Patches

You may wish to record the changes between two revisions of a file, then apply those changes to a third file. You do this by creating a patch and then applying it.

To create a patch:

This generates a patch comprising the differences between a controlled revision of a file and a revision of the file held locally.

1. In JDeveloper, open the file for which you want to create a patch.
2. Click the History tab.

   The History view lists all the revisions of the file. In the lower portion of the History view, the left pane shows the contents of a local revision, and the right pane shows the contents of the controlled revision.
3. Select the revision combination for which you want to create a patch.
4. From the context menu, choose **Generate Patch**.

   The Select Patch Context dialog may open. For help while using this dialog, press **F1**.

   The Generate Patch dialog opens. Complete the dialog as required. For help while using the dialog, press **F1**.

To apply a patch:

1. In the navigator, select the resource to which you want to apply a patch.

   The resource can be an application, a project, or a source file.
2. Select **Versioning > Apply Patch.**

   If you chose to apply a patch to a project, the Select Patch Context dialog opens, through which you should specify whether you are applying a project file (.jpr) patch, or whether you are updating the contents of a project.

   The Apply Patch dialog is opened.
3. In the grid at the top of the Apply Patch dialog, check that the target resources are correctly identified.
4. Choose the source of the patch. For more information about this and the other options on the dialog, press **F1**.
5. Click Preview. This opens the Apply Patch Preview window, in which you can accept or reject particular changes. For more information about the options in the Apply Patch Preview window, press **F1**.
6. To apply the patch, click **OK**.

6.6 Using Serena Dimensions with Oracle JDeveloper

JDeveloper allows you to use the source control features of Dimensions. JDeveloper integrates the repository management and file access features of Dimensions so that you can access your repository, check files in and out, and view the checked-in versions of files under Dimensions control.

Dimensions is a popular version-control system that is part of a larger content-management and workflow control package. JDeveloper includes an extension which allows you to access the version-control features of Dimensions from within the JDeveloper IDE.

To use Dimensions from within JDeveloper, you need to complete several simple setup operations, to ensure that your Dimensions repository is available and your working files are under Dimensions control.

You can then set the current project for JDeveloper, under the control of Dimensions.

After setting up Dimensions and JDeveloper, your typical workflow will probably follow this basic sequence:

- Check out files to be edited
- Make edits and do other content development
- Check files in to the repository

In addition to the general workflow outlined here, you may also find it useful to undo a file checkout.

6.6.1 How to Set Up Dimensions and JDeveloper

Before you can use Dimensions as your version control system with JDeveloper, you need to perform some initial setup operations. This setup involves connecting to a Dimensions repository, learning how to disconnect if required, and choosing the initial project in JDeveloper.

6.6.1.1 How to Connect to a Dimensions Repository

You connect to a Dimensions repository when you want check out files, synchronize your working copies with the common repository, or check files in before a build. Once you are connected, most file operations in Dimensions are available from the context menu for a file, folder or project.

To create a Dimensions profile:

1. Select **Versioning > Dimensions > Connect to Dimensions**.
2. In the Profile field, type a name for the profile you plan to create. Use a name which will be easy to identify when you are choosing between multiple profiles later
3. Enter the username and password you use to log in to the Dimensions server for the project for which you are creating this profile.
4. Enter the server's URL (for example, myserver.mycompany.com), the database name and the database connection. You should be able to obtain this information from your Dimensions administrator.
5. Click **OK**.

To connect to a Dimensions repository with an existing profile:

1. Select **Versioning > Dimensions > Connect to Dimensions**.
2. Enter the username and password you use to log in to the Dimensions server.

If you are connecting to Dimensions for the first time, you will need to create a profile for this connection. You can have multiple profiles for connecting to different servers and databases, if you use Dimensions for multiple projects.

6.6.1.2 How to Disconnect from a Dimensions Repository

Disconnecting from a Dimensions repository lets you connect to another repository, if your organization uses more than one repository or more than one version control system. You also need to disconnect from Dimensions if you plan to connect with a different profile.

To disconnect from a Dimensions repository:

1. Select **Versioning > Dimensions > Disconnect from Dimensions**.
2. Click **OK**.

6.6.1.3 How to Add Files to Dimensions Control

As you create new files for your projects and applications, you need to add them to Dimensions control so that they are available to your other team members and to the build system.

To add files to Dimensions control:

1. Select the file, then click the right mouse button and select **Versioning > Dimensions > Add**.
2. When JDeveloper displays the Add Items dialog, select the item you wish to add, type a comment (optional), and then click **OK**.

6.6.1.4 How to Remove Files from Dimensions Control

If a file no longer applies to the project you are working on, you can remove it from Dimensions control.

To remove a file from Dimensions control:

1. Click the file in the Application Navigator to select it.
2. Select **Versioning > Dimensions > Remove**.
3. Click **OK**.

6.6.1.5 How to Set the Current Project

Setting the current project lets Dimensions know which of your local directories to monitor for changes versus the Dimensions repository. Once you have set the current project, Dimensions will display files with unsaved changes in the Pending Changes list, as well as making the files in that project available in the Application Resources navigator and more.

Before you can set a project, you must be connected to Dimensions.

To set the current project:

1. Select **Versioning > Set Current Project**.
2. To select a project from your local workspace (such as the default directory, JDeveloper`/mywork`), select **Use Global Project.** If you are working in the default project defined by your Dimensions administrator, select **Use Default Project**.
3. Select the product and project from the drop-down selection boxes.
4. Enter the root directory for this project's files in your local file system, or click Browse to choose from a list.
5. Click **OK**.

6.6.2 How to Work with Files in Dimensions

After initial setup, most of your work in Dimensions will involve files. You will need to add files to (and, occasionally remove files from) the Dimensions repository. You can browse files from the Versioning Navigator, where you can check them out, check them in, and otherwise manipulate them. Additionally, Dimensions lets you view a copy of a file in the repository so you can search for changes from a version of the same file that you are working on.

6.6.2.1 How to Import Files to Dimensions

If you have created a new JDeveloper application, you can add all that application's files to Dimensions in a single operation by using the Import Wizard.

To import files to Dimensions:

1. Create an application (**File > New > Applications > select application type**).
2. Version the application (**right-click selected project > Version Project > Dimensions**).
3. Create a connection to source control (**right-click selected version control system in Versioning Navigator > New Dimensions Connection > enter Dimensions connection data**). This opens the Import to Dimensions wizard, displaying the Welcome screen.
4. From the Welcome screen, click **Next** to continue, or select **Skip this page next time** to proceed to the Destination page the next time you use the Import to Dimensions Wizard.

6.6.2.2 Using Navigator Icon Overlays

JDeveloper uses several overlays on the file navigator icons to represent the state of the associated file in Dimensions, as listed in Table 6–3.

Table 6–3 Dimensions Status Icons

Icons	Description
	**File not under Dimensions control**: The file is not under Dimensions control
	**File extracted by multiple users** The file has been extracted by others, but is available for checkout
	**File extracted by others** The file has been extracted by others and is not available for checkout
	**File extracted by single user** The file has been extracted by a single user, but is available for checkout
	**File not extracted** The file has not been extracted
	**File not authenticated** The user has not logged into the Dimensions server
	**File removed from server** The local file has been removed from the Dimensions server

6.6.2.3 How to Download a Dimensions Project

Downloading files from the Dimensions repository to your local working directory is the key to working in Dimensions. You can do this after you have connected to Dimensions.

Once you have signed in to the Dimensions server, use the Versioning Navigator to navigate to the content you wish to work with. After you select a folder, Dimensions

lets you download the contents of the folder (giving you the option of expanding subfolders) to your local working directory.

Now that you have copied the content from the Dimensions repository to your working directory, you need to set this content as the current project in Dimensions.

To download from the Dimensions repository:

1. Click the + next to the Dimensions entry in the Versioning Navigator. This displays the profile with which you connected to the Dimensions server.
2. Click the + next to your profile name. This expands the list of projects on the Dimensions server that are available to your profile. Depending on the number of projects and the connection speed, this may take a few minutes.
3. Browse the available projects, clicking the + to expand project categories and folders. When you have identified the content you wish to work with, select the folder, click the right mouse button and select Download.
4. Specify the download location and settings as follows:
   - **Destination**

     Enter the location of your local work area. Use the Browse button to select from a directory browser.
   - **Expand Substitution Variables**

     Select if your project uses substitution variables that you wish to expand when downloading to your work area.
   - **Use database timestamps**

     Select if you wish the local file copies to be created with the same timestamps they have in your Dimensions repository. If you leave this unselected, files will be created with timestamps reflecting the date and time you downloaded them.
   - **Recurse**

     Select if you wish Dimensions to expand (recurse) directories, checking out all files in all directories of the project.
   - **Overwrite local files**

     Select to overwrite any local files in your work area with content from the repository. Any changes you have made to the local files and have not yet checked in will be lost.
5. Click **OK**.

6.6.2.4 How to Check Out Files

Checking out files from the Dimensions repository to your local working directory lets you make changes which will be tracked against the changes of others on your team. To do this, use the Dimensions Web client; your Dimensions administrator will have the URL and any login information you need to access the Dimensions Web client.

Once you have signed in to the Dimensions Web client, use the left-hand pane to navigate to the content you wish to work with. The Web client displays this content in the right-hand viewing pane. If you select a folder, Dimensions will check out the contents of the folder (giving you the option of expanding subfolders) to your local working directory.

You can control whether Dimensions automatically checks out files when you open a Dimensions-controlled file in JDeveloper. Select the T**ools > Preferences > Versioning > Dimensions > General** page. The option Automatically Check Out Files lets you specify whether Dimensions is to check out a file from the repository when you edit your local working copy.

To check out content from the Dimensions repository:

1. Select V**ersioning > Dimensions > Check Out**. This displays the Check Out dialog.
2. In the field titled Check out contents of project folder to, enter or browse to the path to which you wish to check out the files (by default, this is JDeveloper`/mywork`).
3. The Dimensions Web client presents the following options; select the ones you require for the project you are working on:
   - **If writable workfile exists**:

     Select **Overwrite** to force Dimensions to overwrite any writable file with the same name in your working directory. Otherwise, select **Don't overwrite**.
   - **Relate to Requests**

     You can associate this checkout (and any changes you make as part of this checkout) to a request (a means of tracking build changes, bug fixes, and more) that is maintained by Dimensions. The Dimensions version control system is part of a much larger project-management system with integrated workflow, bug tracking, and more. For details, refer to the on-line help for the Dimensions Web client.
   - **Include subdirectories**

     Select this if you wish to check out not only the content you have selected, but also the content of subdirectories inside the selected folder. If you perform local builds for testing and verification, and therefore need access to all the files in the project, you should select this.
4. Click **OK**.

Now that you have checked out the content from the Dimensions repository to your working directory, you need to set this content as the current project in Dimensions.

When you have made the required changes and verified them locally, you can check in your changed files.

6.6.2.5 How to Undo a File Checkout

Undoing a file check out essentially leaves the file in the repository untouched, while removing any record from the repository's database that pertains to who checked out the file and when it was checked out. This not only leaves the file unchanged since the last checkin, it also leaves the file available to be checked out by other team members.

Note that undoing a check out will essentially discard any work you've done since checking out the files. Use your judgment as to whether undoing a file checkout is the most effective solution to the situation you find yourself in, and consider whether saving a local copy (outside your local Dimensions directory) will be worthwhile.

To undo a file checkout:

1. Click the right mouse button on the file in the Pending Changelist, then select **Undo Check Out**. This opens the Undo Check Out dialog.

2. In the After Undo Check Out field, select how you want Dimensions to leave your local copies:

Leave workfile as read only
Select this to leave all copies in your local directory unchanged and in read-only mode. This is safest if you have made many changes that you wish to retain for future use.

Replace workfile with latest copy
Select this to have Dimensions update your local directory to the latest versions of all files, from the Dimensions repository. This ensures that you will have the up-to-date versions to work from, but it will overwrite any changes you have made.

Delete workfile
Select this to have Dimensions delete all checked-out files from your local directory. This is the surest way to start with a "clean sheet," by checking out all files in the project again.

3. In the Include subdirectories field, select whether you want Dimensions to apply the previous choice (leave workfile as read only, replace workfile with latest copy, or delete workfile) to all subdirectories under the one you have selected. Depending on how your subdirectories are structured, you can use this to control the granularity of which files and what content you choose to undo.
4. Click **OK**.

After undoing a check out, you will need to check out the latest content before you can work in Dimensions again.

You may also wish to get a copy of the Dimensions-controlled content on your local working directory.

6.6.2.6 How to Check In Files

After making and verifying your changes to the content for which you are responsible, you make your work available to the rest of the team by checking it in to Dimensions. This uploads your work to the Dimensions repository, where it will be available to other team members and to your organization's build process.

To check in files to Dimensions:

1. Click the right mouse button on the file in the Pending Changelist, then select Check In. This opens the Check In dialog.
2. In the Check In dialog, make the appropriate selections for the work you are checking in:

Check In from directory:
Enter the directory in your local file system from which you are checking in new work, or click the button next to the field to open a file system browser.

Include subdirectories
Check this if the work you are checking in includes files in subdirectories under that which you have selected.

If workfile is unchanged:
Check this to determine how Dimensions is to handle unchanged content in your working directory. You can choose to Check in your unchanged files, which will update the time stamp in the Dimensions repository, or Undo checkout, which

removes any record, from the Dimensions database, of your having checked out the unchanged files.

After Check In:
This tells Dimensions how to leave the content in your local working directory. You can select Leave workfile as read-only, in which case Dimensions will leave the file as you left it but mark it read-only to prevent making unrecorded changes to the file while not under Dimensions control. Alternatively, you can select Delete workfile, which removes the local copy of your file (and all other files you are checking in). Whichever you select, you will need to check out the files from Dimensions again before resuming work.

Description
Enter a description of the changes you are making. You can include bug-tracking numbers or other information to help identify and track the specific checkin you are making at this time; however, tracking the files against the Dimensions request system is a more effective way of keeping tabs on changes.

3. Click **OK**.

6.6.2.7 About the Pending Changes List

The Pending Changes list displays files, folders and other elements that have been created or modified and not yet added to Dimensions. This includes:

- files which you have edited in JDeveloper
- all files in a new project that you have created in JDeveloper and not yet placed under Dimensions control
- any file for which some Dimensions-related activity needs to be performed.

Files are added to the Pending Change list when you save a copy locally.

6.7 Using Rational ClearCase with Oracle JDeveloper

JDeveloper allows you to use the source control features of Rational ClearCase release 4.0 onwards (including ClearCase 2002). JDeveloper works in a seamless manner with ClearCase so that once you have it configured you can add files to source control, and check them in and out from the navigators

The JDeveloper Rational ClearCase extension allows you to use the source control features of Rational ClearCase inside JDeveloper. Once you have JDeveloper configured to work with your ClearCase installation, you can add files to source control, and check them in and out from the navigators.

To work in ClearCase, you have to store your workspaces, projects and files on your ClearCase view; before new projects and files are under ClearCase source control, you have to explicitly add them to ClearCase. Once your files are added to your ClearCase view, you can check them in and out, compare versions, review file histories, and (if necessary) remove files from ClearCase

6.7.1 How to Configure JDeveloper to Use Rational ClearCase

To use ClearCase with JDeveloper, you must have ClearCase 4.0 or greater client installed on the same machine as JDeveloper.

To configure JDeveloper to use ClearCase:

1. Select **Tools > Preferences**, then unfold the Versioning and ClearCase nodes.
2. Set the preferences as required. Note that there are several pages of preferences for ClearCase. For information about the specific preferences, press F1.
3. Close the Preferences dialog by clicking **OK**.

If you have not chosen (in the preferences) to have connections made automatically, make sure that you have a connection to the ClearCase server, then select **Versioning > Connect to ClearCase**.

Now that you have configured JDeveloper to work with ClearCase, you can access files and folders that already exist in a mounted ClearCase view. New files and folders must be created in or copied to your ClearCase view.

To add new files to ClearCase, see Section 6.7.2, "How to Add a File to ClearCase".

6.7.2 How to Add a File to ClearCase

To work in ClearCase, you have to store your workspaces, projects and files on your ClearCase view, and before new projects and files are under ClearCase source control, you have to explicitly add them to ClearCase.

The comment pane in the Add to ClearCase dialog allows you to build up comments for different groups of files. The comments you type apply to the files you have selected. For example, you can select all the files and type a global comment. Next, select a smaller number of files. The first comment is displayed and you can add to it. Then you can select just a single file in this group and add another comment specific to that file.

To add one or more files to ClearCase:

1. Select the files in the navigator, and choose **Versioning > Add**. The Add to ClearCase dialog is displayed, listing the items you have selected.

   If you want to continue working on the files, leave the **Check In** box clear. If you check the box, the files will be checked in and you must check them out when you want to work on them.
2. To add the files in the list to ClearCase source control, click **OK**. You may see one or more messages asking whether you should add folders to ClearCase. Click **Yes**.

6.7.3 How to Refresh the Status of Objects under ClearCase Control

The source control status of an object is indicated in the Application Navigator by an icon overlay, as below.

Table 6–4 Status icons for ClearCase

Icon	Description
	The object is checked out and can be modified.
	The object is checked in and must be checked out before it can be modified.

If the status of an object is changed outside JDeveloper, for example by checking in an object using external source control software, the new status might not immediately be

shown in JDeveloper. To ensure that the status indicated in the Application Navigator matches the true status of the object in the source control system, you can perform a manual refresh.

To refresh the status of objects in JDeveloper, select **Versioning > Refresh States**.

6.7.4 How to Remove a File From ClearCase

You can remove a file from ClearCase if you no longer need it. The files are removed from the current version of the directory which contains them.

To remove a file from ClearCase:

1. With the file or files selected in the navigator, choose **Versioning > Remove**. The Remove from ClearCase dialog is displayed with the files listed.
2. To remove all the listed files from ClearCase, click **OK**.

The files are removed from the current version of the directory, and you will no longer be able to work with them.

6.7.5 How to Check In a File to ClearCase

When you have finished working on a file, you should check it into ClearCase.

To check in files shown in the Application Navigator:

- Select the files in the Application Navigator, and choose **Versioning > Check In**.

The Check In to ClearCase dialog is displayed listing the files that you selected.

If you want to check in the files even though they are identical to the previous versions in ClearCase, check the **Force Check In Where Files Are Identical** box. If you do not check this box, the file may remain checked out, depending on how ClearCase handles files of that type. If the file remains checked out, you can use the Undo Checkout command to return the file to its previous checked in state.

Type comments for this checkin into the Comments box or, to use the same comments for the check in that were used for the checkout, check the Use Checkout Comments box.

To check in the listed files, click **OK**. The files in the list are checked in to ClearCase source control.

To check in files shown in the Checked Out Files window:

1. Select the files in the Checked Out Files window that you want to check in. To obtain more information about the Checked Out Files window, press F1.
2. In the button bar of the viewer, click the **Check In** button.

6.7.6 How to Check Out a File From ClearCase

To work on a file, it must be checked out from ClearCase.

Files will be checked out automatically when you start to change them, if the Automatically Check Out Files preference is set on the ClearCase preferences page (available by choosing **Tools > Preferences** and selecting ClearCase). This preference applies to data files, workspace files and project files. For data files, the file is checked out when you begin to edit in the source view. If you check out a file unintentionally, immediately use **Versioning > Undo Checkout** to revert.

To check out one or more files manually:

1. Select the files in the Application Navigator, and choose **Versioning > Check Out**. The Check Out From ClearCase dialog is displayed listing the items you have selected.

   If you want to prevent another user from checking out the same files and then checking them in before you do, check the **Check Out Reserved To User** box.

2. Type comments about this checkout into the Comments box.
3. To check out the listed files, click **OK**.

The files in the list are checked out from ClearCase source control and you can work on them.

Checked out files are shown in the Checked Out Files window. This window opens when files are first checked out. You can open it at other times by selecting **Versioning > View Checked Out Files**.

6.7.7 How to Undo a ClearCase Checkout

If you have checked out a file but not made any changes to it, or if you want to discard the changes you have made, you can undo the last check out of that file.

Caution: You may lose you work if you use this on a file that you have changed since it was checked out.

To undo checkout for one or more files:

1. Select the files in the navigator, and choose **Versioning > Undo Checkout**. The Undo Clearcase Checkout dialog is displayed listing the items you have selected.
2. To undo the checkout for all the listed file, and lose any changes that you have made to those files, click **OK**.

6.7.8 How to List ClearCase Checkouts

You may want to see all the files that you have checked out from ClearCase, for example to see which files need to be checked in before performing another action.

To list ClearCase checkouts:

- Choose **Versioning > View Checked Out Files**. The Checked Out Files viewer is displayed.

6.7.9 How to Compare Files Checked In to ClearCase

Use these procedures to compare versions of files that are under ClearCase source control. You can compare a file with: its immediate predecessor any of the file's previous revisions any other file on your file system.

You can choose which Compare Viewer to use (JDeveloper or ClearCase), by setting an option on the ClearCase Version Tools preference page under **Tools > Preferences**.

To compare a file with its immediate predecessor:

- With the file selected in the Application Navigator, choose **Versioning > Compare with Previous Version**.

If there are no differences, a message is displayed. Otherwise, a Compare Viewer is displayed through which you can find and reconcile the differences.

To compare a file with another revision:

- With the file selected in the Application Navigator, choose V**ersioning > Compare.** Ensure that the Predecessor File Revision option is chosen.

Previous versions of the file are listed in the Compare ClearCase File dialog. Select the version you want to compare the current file with and click **OK**.

If there are no differences, a message is displayed. Otherwise, a Compare Viewer is displayed through which you can find and reconcile the differences.

To compare a file with a file outside ClearCase source control:

- With the file selected in the Application Navigator, choose **File > Compare With > Other File**.

The Select File to Compare With dialog is opened.

Browse to and select the file you want to compare the current file with, and then click **Open**.

6.7.10 How to Display the History of a ClearCase File

Use this procedure to display the history of a file that is under ClearCase source control.

To display the history:

- With the file selected in the navigator, choose **Versioning > View History.**

6.7.11 How to Display the Description of a ClearCase File

Use this procedure to display the description of a project or a file that is under ClearCase source control.

To display the description:

- With the file selected in the navigator, choose **Versioning > View Description.**

The description appears in the Messages Log window.

6.8 Using Team System with Oracle JDeveloper

Oracle JDeveloper's Team System extension allows you to use the source control features of Microsoft Visual Team System inside JDeveloper. Once you have JDeveloper configured to work with Team System, you can add files to source control, and check them in and out from the navigators.

To begin using Team System with JDeveloper, you must first create a workspace using Team System software, and then populate this workspace with content from the Team System server. Files are checked out to the workspace, where they can be worked on. Files newly created within JDeveloper must be added to version control. Changed and new files are made available to other users by checking them in to the Team System server.

6.8.1 How to Set Up Team System and JDeveloper

Before beginning to use Team System with JDeveloper, there are some initial steps you need to follow:

1. Set up the Team System client software. See Section 6.8.1.1, "How to Set Up Team System for Use with JDeveloper."
2. Configure JDeveloper for use with Team System, including the preferences and other settings for making Team System the source control system recognized by JDeveloper. See Section 6.8.1.2, "How to Configure JDeveloper for Use with Team System."

In practice, Team System (like any version control system) consists of operations that you use at varying times depending on the place in the product lifecycle. For example, if you create a new file, you'll need to add it to Team System control.

Other operations you may perform, depending on the stage of development, include:

- Checking out files from the server so that you can work on them. See Section 6.8.2.3, "How to Check Out Files."
- Making changes to a file saved in your Team System workspace, and make them available to other users. See Section 6.8.2.6, "How to Check In Files."
- Using Team System's Shelving feature to save file changes in the Team System server without having to check the files in. See Section 6.8.2.12, "How to Shelve and Unshelve Files."
- Resolving conflicts between your changes and changes made by your team mates to your Team System files
- Checking in files to your Team System server.

6.8.1.1 How to Set Up Team System for Use with JDeveloper

To set up Team System for use with JDeveloper, follow these steps:

1. Install the Team System server.
2. Install the Team System client software.
3. Connect the Team System client software to the Team System server.
4. Use the Team System client software to create one or more workspaces.
5. Use the Team System client software to populate the workspace(s) with content from the Team System server.

Instructions for doing the above are given in the Team System online help.

6.8.1.2 How to Configure JDeveloper for Use with Team System

Once you have set up Team System for use with Oracle JDeveloper, you are ready to configure JDeveloper to use Team System. In addition to the steps in Section 6.8.1.1, "How to Set Up Team System for Use with JDeveloper," make sure you have already i * Installed the JDeveloper Team System VCS extension (from the Official Oracle Extensions and Updates center).

To configure JDeveloper for use with Team System, carry out the following activities in JDeveloper:

- Set preferences.
- Select Team System as the JDeveloper versioning system.

- Set the workspace to use with JDeveloper.
- Create a JDeveloper project to hold the workspace files.
- Refresh the workspace folders in JDeveloper.

To set JDeveloper preferences for use with Team System:

1. Choose **Tools > Preferences**, then select **Extensions** in the left pane of the Preferences dialog.
2. In the right pane, make sure that Versioning Support *n* is checked, then click Configure.
3. Ensure that Versioning Support for Team System *n* is checked
4. In the left pane of the Preferences dialog, open the Versioning node and then the Team System node. The main Team System preferences panel is shown. Other Team System preferences panels are shown when you click on the items beneath the Team System node.
5. Make changes to the preferences as required.

   For more information about the specific preferences, press F1.
6. Click **OK** to close the Preferences dialog.

To select Team System as the versioning system:

- Choose **Versioning > Version System [...] > Team System**.

To set the workspace to use with JDeveloper:

1. Choose **Versioning > Set Workspace**.
2. Select the required workspace from the list.

To create a JDeveloper project to hold the workspace files:

1. Select **File > New** to open the New Gallery.
2. Use the New Gallery to create a new application and project.
3. In the Application Navigator, select the newly created project and click the **Add to Project Content** button in the toolbar.

   This opens the Project Content page of the Project Properties dialog.
4. Use the **Add** button in the Java Content area to add the location of the workspace.

   If your workspace contained Java sources, a dialog is displayed through which you should confirm that you want the sources added to the project content.

   To avoid confusion, you may wish to remove non-workspace locations from the Java Content list.
5. Click **OK** to close the Project Properties dialog.

To refresh the workspace folders in JDeveloper:

- Choose **Versioning > Refresh** Workspace Folders.

6.8.2 How to Work with Files in Team System

In addition to the file system operations you are probably familiar with from most version control systems (that is, checking files in and out, adding files to the repository, etc.), Team System lets you specify individual file versions from the server.

6.8.2.1 How to Get Versions of Files from the Team System Server

JDeveloper lets you get (from the Team System server) a version of a file that is in the Application Navigator. You must previously have used the get command in the Team System client software to populate your workspace with source files.

You can use this procedure to obtain the following versions of files: the latest version; files from a previously saved named changelist; files with a particular date stamp; files from a previously created named label; files from a particular workspace version.

The version obtained from the Team System server will replace the version currently in the Application Navigator.

To get versions of files from the Team System server:

1. In the Application Navigator, select the application, project or files to set the scope of the Get operation.
2. Select **Versioning > Get.**

   The Get dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.

6.8.2.2 How to Add FIles to Team System Control

Use to bring files under Team System source control. The files will be added to the Team System server and made available to other users when you next check in the file.

To add files to Team System Control

1. In the Application Navigator, select the file that you want to add to Team System control.
2. Select **Versioning > Add**.

   The Add dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.
4. To add the file to the server and make it available to other users, check in the file.

6.8.2.3 How to Check Out Files

Use to check out files so that you can work on them. The files must already by under Team System source control.

To check out files:

1. In the Applications Navigator, select the application, project or file that you want to check out.
2. Select **Versioning > Check Out.**

   The Check Out dialog is opened.
3. Complete the dialog.

   For information while using the dialog, press **F1**.

6.8.2.4 How to View the Status of a File

Use this procedure to check the status of a file that is under Team System source control. See also Section 6.8.2.5, "How to Refresh the Status of Files."

To view the status of a file:

1. With the file selected in the Application Navigator, open the context menu and select **Versioning > Properties**.
2. Select the Versioning tab.

The status labels shown are those used by Team System to describe the source control status of the file.

The main statuses are:

- **Edited** - In JDeveloper, the file is checked out and may have been modified.
- **Unchanged** - In JDeveloper, the file is currently checked in.
- **Scheduled for addition** - In JDeveloper, the file has been added (that is, brought under source control) but not yet checked in.

6.8.2.5 How to Refresh the Status of Files

The source control status of a file is indicated in the JDeveloper navigators by icon overlays, as below.

Table 6–5 File status icons in Team System

Icon	Description
	The object is checked in and must be checked out before it can be modified.
	The object is checked out and can be modified.
	The object is not under source control.
	The file has been brought under source control but has not yet been checked in to the Team System server.
	The object has been scheduled for removal from the Team System server the next time it is checked in.

To refresh the status of files in JDeveloper:

- Select **View > Refresh**.

6.8.2.6 How to Check In Files

Use to check in a file to the Team System server. A checked in version of a file can be seen and worked on by other users.

To check in files:

1. In the Application Navigator, select the file that you want to check in.
2. Select **Versioning > Check In**.

   The Check In dialog is opened.

3. Complete the dialog.

   For information while using the dialog, press **F1**.

6.8.2.7 How to Resolve Conflicts in File Versions

If there is a conflict between your copy of the file and the one in the Team System server when you attempt to check it in, you will see a message box saying that the operation cannot be completed. To overcome this problem, you must first cancel the check-in operation, then do one of the following:

- Revert to a non-conflicting version of the file.
- Resolve the conflict using the merge tool in the Team System client software.

To revert to a non-conflicting file version:

- Select the file in the Application Navigator and choose **Versioning > Undo**.

6.8.2.8 How to Undo Changes to Files

Use to undo the most recent change to a file.

To undo changes:

1. In the Application Navigator, select the file whose last change you want to undo.
2. Select **Versioning > Undo**.

   The Undo dialog is opened.

   The change will be undone when you click **OK**.

6.8.2.9 How to Replace a File with the Team System Base Version

Use this procedure to replace a file with the base version. The base version is the version from which the one you are currently working on originated.

To replace a file with the Team System base revision:

1. In the Application Navigator, select the file to be replaced.
2. Choose **File > Replace With > Base Version**.

   The Replace With Base Version dialog opens.
3. Check that the file that you want to replace is shown in the dialog.
4. To replace the file, click **OK**.

6.8.2.10 How to View the History of a File

Use this procedure to open the History Viewer and view the history of files held under Team System control.

To view the history of a file:

- With the file selected in the Application Navigator, choose **Versioning > Version History** from the context menu.

For more information while using the History Viewer, press **F1**.

6.8.2.11 How to Compare Files In Team System

Use these procedures to compare files that are under Team System control with other versions of the same files, or with other files.

To compare versions of a file:

1. From the context menu for the file, choose **Compare With.**
2. Select either **Previous Version**, **Latest Version** or **Other Version**.

If there are no differences, a message is displayed. Otherwise the version or versions are shown in the History tool.

To compare a file with another file:

1. From the context menu for the file, choose **Compare With > Other File**.

   The Select File to Compare With dialog is opened.

2. Select the file to be compared.

   The files are shown in the Compare tool.

To compare two files:

1. Select the two files in the navigator.
2. From the context menu for one of the files, choose **Compare With > Each Other**.

The files are shown in the Compare tool.

6.8.2.12 How to Shelve and Unshelve Files

Shelving lets you save file changes in the Team System server without having to check the files in. As part of the shelving process, you can choose either to continue to work on the changed files or to remove them from view and revert to unchanged versions.

When you later want to make use of the file changes that were shelved, you can unshelve them.

If you decide you no longer want to keep changes that were shelved, you can delete the shelveset that you put them in.

To shelve a set of file changes that have not been checked in:

To shelve a set of file changes that have not been checked in:

1. In the Application Navigator, select the versioned project containing the files.
2. Select **Versioning > Shelve**.

   The Shelve dialog opens.

3. Complete the dialog.

For information while completing the dialog, click **F1**.

The file changes will be shelved when you click **OK**.

The file icons in the Application Navigator will change to reflect the new file statuses, if any.

To unshelve a set of file changes:

1. In the Application Navigator, select the versioned project into which you want to unshelve the file changes.
2. Select **Versioning > Unshelve**.

   The Unshelve dialog opens.

3. Select the shelveset name for the shelveset containing the file changes.

The file changes will be unshelved when you click **OK**.

Files deleted since the shelveset was created will be reinstated and the file icons in the Application Navigator will change to reflect the new file statuses.

To delete a shelveset:

1. Select **Versioning > Delete Shelveset.**

   The Delete Shelveset dialog opens.

2. Select the name of the shelveset that you want to delete.

The shelveset will be deleted when you click **OK**.

6.8.2.13 How to Delete Files

Use to delete files from your workspace and from the Team System server.

To delete a file:

1. Select the file in the Application Navigator and choose **Versioning > Delete**.

   The Delete dialog is displayed with the file listed.

2. Click **OK**.

   On the Outgoing tab of the Pending Changes window (**Versioning > Pending Changes**), the file will be indicated as ready for deletion: a black diagonal cross is added to the file's icon.

3. To complete the deletion of the file, select it in the Pending Changes window and choose **Versioning > Check In**.

   The Check In dialog is opened.

4. Add your comments, if any, and click **OK**.

The file is deleted from your workspace and from the Team System server.

6.8.3 How to Use Import and Export Features

The JDeveloper import and export features allow you to create and apply patches from just the changes or revisions between two versions of a file.

6.8.3.1 How to Create Patches

You may wish to record the changes between two revisions of a file, then apply those changes to a third file. You do this by creating a patch and then applying it.

To create a patch:

This generates a patch comprising the differences between a controlled revision of a file and a revision of the file held locally.

1. In JDeveloper, open the file for which you want to create a patch.
2. Click the History tab.

   The History view lists all the revisions of the file. In the lower portion of the History view, the left pane shows the contents of a local revision, and the right pane shows the contents of the controlled revision.

3. Select the revision combination for which you want to create a patch.
4. From the context menu, choose **Generate Patch**.

The Select Patch Context dialog may open. For help while using this dialog, press **F1**.

The Generate Patch dialog opens. Complete the dialog as required. For help while using the dialog, press **F1**.

6.8.3.2 How to Apply Patches

You may wish to record the changes between two revisions of a file, then apply those changes to a third file. You do this by creating a patch and then applying it.

To apply a patch:

1. In the navigator, select the resource to which you want to apply a patch.

   The resource can be an application, a project, or a source file.

2. Select **Versioning > Apply Patch**.

   If you chose to apply a patch to a project, the Select Patch Context dialog opens, through which you should specify whether you are applying a project file (.jpr) patch, or whether you are updating the contents of a project.

   The Apply Patch dialog is opened.

3. In the grid at the top of the Apply Patch dialog, check that the target resources are correctly identified.

4. Choose the source of the patch. For more information about this and the other options on the dialog, press **F1**.

5. Click **Preview**. This opens the Apply Patch Preview window, in which you can accept or reject particular changes. For more information about the options in the Apply Patch Preview window, press **F1**.

6. To apply the patch, click **OK**.

6.9 Using WebDAV with JDeveloper

Web-based Distributed Authoring and Versioning, or WebDAV, is an extension to HTTP which allows users to edit and manage files on WebDAV-enabled servers in a collaborative fashion. WebDAV connections in JDeveloper allow you to view files hosted on WebDAV servers in the same way as you would files on the local file system. Files located on WebDAV servers, accessed using WebDAV connections in JDeveloper, can be viewed in the same way as files stored on the local file system or LAN.

As WebDAV clients provide access using HTTP, files can be accessed through firewalls (configured to support WebDAV extensions) that would otherwise prevent FTP file transfer. The JDeveloper read-only implementation of WebDAV supports the current WebDAV 1.0 standard, which does not support versioning. As a WebDAV client, JDeveloper can connect directly to any Oracle Internet File System, allowing you to view WebDAV files from the database.

6.9.1 WebDAV Server Requirements

You must run a WebDAV server to use JDeveloper as a WebDAV client. The WebDAV server must be one of the following:

- Oracle Internet File System 8.1.7 (or above)
- Apache 1.3.19 (or above)

> **Note:** If the Apache server is version 1.x, the `mod_dav` module must also be installed.

- A server that conforms to the WebDAV 1.0 standard

> **Note:** If you access the Internet through a firewall, it must be configured to process the extended HTTP commands used by WebDAV.
>
> If your web server is configured to redirect URLs to a different server (for example, if you are using `JkMount` in Apache to redirect requests for certain file extensions to Tomcat), be aware that WebDAV will not be available for those resources if the server you are redirecting to does not support WebDAV in that context.

If you'd like to find out more about WebDAV, see the following Web sites:

- `http://www.webdav.org`
- `http://httpd.apache.org/docs-2.1/mod/mod_dav.html`

6.9.2 How to Create a WebDAV Connection

WebDAV connections created in JDeveloper allow you to view files and folders as part of a JDeveloper project.

> **Note:** The same URL cannot be used for more than one WebDAV connection on the same JDeveloper client.

To create a WebDAV connection in JDeveloper:

1. In the New Gallery, choose **General > Connections > WebDAV Connection**, then click **OK**.
2. Use the WebDAV Connection dialog to create a connection.

   For more information while using the dialog, press **F1**.

6.9.3 How to Access a WebDAV-Enabled Server Via a Proxy Server

If you access the internet via a proxy server you need to configure JDeveloper before accessing WebDAV-enabled servers on the internet.

To access a WebDAV-enabled server via a proxy server:

1. Check with your network administrator to ensure that your proxy server is WebDAV-enabled.
2. In JDeveloper choose **Tools > Preferences**, click **Web Browser and Proxy** in the left pane of the Preferences dialog box, make sure that the **Use HTTP Proxy Server** checkbox is checked, then enter the details for the proxy.
3. If the WebDAV-enabled server you want to access is inside your firewall and you do not need to go through your proxy server to access it, add the name of the WebDAV server to your default web browser's proxy exceptions list. This is

normally set on the browser's preferences/settings page with the other proxy settings.

6.9.4 How to Modify a WebDAV Connection

WebDAV connections are shown in the Application Resources section of the Application Navigator, listed under the Connections node.

Existing WebDAV connections can be modified.

To modify a WebDAV connection:

1. Right-click the WebDAV connection that you want to modify.
2. Choose **Properties**.
3. On the WebDAV Connection Properties dialog, change the details of the WebDAV connection.

   For help while using the dialog, press **F1**.
4. Click **OK**.

6.9.5 How to Refresh a WebDAV Connection

WebDAV connections are shown in the Application Resources section of the Application Navigator, listed under the Connections node.

To ensure that the folders and files accurately reflect the current contents of the WebDAV server, you can manually refresh the display of a WebDAV connection.

Note: All folders and files listed for the WebDAV connection are refreshed. The properties of the folders and files, and their contents, are refreshed.

To refresh the entire contents of a WebDAV connection:

1. Right-click the WebDAV connection that you want to refresh.
2. Choose **Refresh**.

6.9.6 How to Delete a WebDAV Connection

WebDAV connections are shown in the Application Resources section of the Application Navigator, listed under the Connections node.

Deleting a WebDAV connection from JDeveloper does not affect any of the files or folders on the WebDAV server itself.

To delete a WebDAV connection:

1. Right-click the WebDAV connection you want to delete.
2. Choose **Delete**.

You can subsequently recreate the connection, in which case the files and folders that were part of it will be shown beneath it again.

7

Building, Running and Debugging Applications

This chapter provides an overview of the building, running, and debugging features in JDeveloper. These features are explained in greater detail in the subsequent chapters of this guide.

This chapter includes the following sections:

7.1 About Building, Running and Debugging Applications

After you have completed the design time aspects of your application, you are ready to build, run, and debug your project.

JDeveloper provides three ways to build projects: using the Make and Rebuild, Apache Ant, or Maven.

When you run your project, the Run Manager manages the processes that are run, debugged, or profiled. The Run Manager window is automatically displayed when two or more such processes are active at the same time,. When a process has completed, it is automatically removed from the Run Manager.

The debugger enables you to investigate your code, and identify and fix problem areas. Two types of debugging are available: local and remote debugging.

7.2 Building Applications

You can build your application using one of these ways:

- Make and Rebuild
- Apache Ant
- Apache Maven

Additionally, you can also clean your application and generate Javadoc for it.

7.2.1 Make and Rebuild

The Make and Rebuild commands execute standard operations for compiling projects in JDeveloper.

Make operations compile source files that have changed since they were last compiled, or have dependencies that have changed. Rebuild operations, in contrast, compile source files unconditionally. You can invoke make on individual source files, on working sets, or on containers such as packages, projects, and workspaces.

7.2.2 Apache Ant

Apache Ant is a build tool similar in functionality to the Unix `make` utility. Ant uses XML formatted buildfiles to both describe and control the process used to build an application and its components. Ant supports cross-platform compilation and is easily extensible. Apache Ant is a product of the Apache Software Foundation. For more information, see the website `http://ant.apache.org/index.html`.

For more information about Apache Ant, see Section 18.6.5, "Building with Apache Ant".

7.2.3 Apache Maven

Apache Maven is a software project management and comprehension tool. Maven can manage a project's build, reporting and documentation from a central piece of information, the project object model (POM). You can build the project using its POM and a set of plugins that are shared by all projects using Maven, providing a uniform build system.

Maven can be extended by plugins to use a number of other development tools for reporting or the build process. For more information about Maven, see `http://maven.apache.org/index.html`.

For more information about Apache Maven, see Section 18.6.6, "Building and Running with Apache Maven".

7.3 Running Applications

JDeveloper offers several techniques to monitor and control the way applications are run. The Run Manager enables you to manage all running processes.

7.3.1 Run Manager

The Run Manager keeps track of processes that are run, debugged, or profiled. When two or more such processes are active at the same time, the Run Manager window is automatically displayed. When a process has completed, it is automatically removed from the Run Manager.

For more information about the Run Manager, see Section 19.2, "Understanding the Run Manager".

7.4 Debugging Applications

JDeveloper provides you with a comprehensive debugger to assess and repair your code. Debugging can be of two types -- local and remote.

A local debugging session is started by setting breakpoints in source files, and then starting the debugger. When debugging an application such as a servlet in JDeveloper,

you have complete control over the execution flow and can view and modify values of variables. You can also investigate application performance by monitoring class instance counts and memory usage. JDeveloper will follow calls from your application into other source files, or generate stub classes for source files that are not available

Remote debugging requires two JDeveloper processes: a debugger and a debuggee which may reside on a different platform. Once the debuggee process is launched and the debugger process is attached to it, remote debugging is similar to local debugging.

7.4.1 How to Use the Debugger

The Debugger provides a number of special-purpose debugging windows that enable you to efficiently identify the problematic areas in your code.

You can control what type of information is displayed in each of the debugger windows. To see what options are available in each window such as which columns to display, right-click in a window and choose **Preferences** from the context menu. Or, you can choose **Tools** > **Preferences** from the main menu and expand the Debugger node to display a preferences page for each debugger window. You can also save the debug information as text or HTML output file.

For more information on using the debugger, see Section 19.7, "Using the Debugger Windows".

7.4.2 Technologies that Use Debugging

Several technologies use debugging facilities provided by JDeveloper. The following table lists these technologies and the corresponding link to their debugging documentation.

Table 7–1 Technologies that Use Debugging

Technology	Documentation Link
ADF Components	Section 19.6.3, "How to Debug ADF Components"
Java Servlet	Section 11.5.7, "How to Debug a Servlet"
JSP Pages	Section 11.4.2, "How to Debug and Deploy JSPs"
Java Programs	Section 19.6, "About the Debugger"
JavaScript	Section 19.6.21, "How to Debug a Javascript Program"
Web Services	Section 16.11.2, "How to Debug Web Services"
Integrated WebLogic Server	Section 9.2.3.2, "How to Run and Debug with an Integrated Application Server"
PL/SQL Programs and Java Stored Procedures	Section 29.3.3, "Debugging PL/SQL and Java Stored Procedures Prerequisites"
Extensions	*Oracle Fusion Middleware Developer's Guide for Oracle JDeveloper Extensions*

8

Auditing and Profiling Applications

This chapter describes the auditing and profiling capabilities of Oracle JDeveloper.

This chapter includes the following sections:

- Section 8.1, "About Auditing and Profiling Applications"
- Section 8.2, "Auditing Applications"
- Section 8.3, "Monitoring HTTP Using the HTTP Analyzer"
- Section 8.4, "Profiling Applications"

8.1 About Auditing and Profiling Applications

Use the auditing and profiling tools that JDeveloper provides to analyze the health and performance of your applications. These tools help you improve the quality of your code. You can use the JDeveloper auditing feature to analyze Java code for conformance to programming standards.

Use the profiler to gather statistics on your program that enable you to more easily diagnose performance issues, such as bottlenecks by identifying methods consuming more time, which method is called the most, how memory is used, and what kind of objects are being created.

8.2 Auditing Applications

Auditing is the static analysis of code for adherence to rules and metrics that define programming standards. Auditing finds defects that make code difficult to improve and maintain. The JDeveloper auditing tools help you find and fix such defects. Code can be audited even when it is not compilable or executable.

You can create and customize profiles, choose the rules to be used, and set parameters for individual rules. Browse the audit rules and metrics to learn more about them.

An audit report displays rule violations and measurements organized as a tree. A row of the tree corresponds to either a construct or a violation, and includes any measured values for the construct or theoretical violation. A construct is a method, class, file, project, or workspace. For more information on auditing tools and steps to audit your code, see Section 18.9, "Optimizing Application Performance".

8.3 Monitoring HTTP Using the HTTP Analyzer

The HTTP Analyzer allows you to monitor HTTP traffic, for example, to:

- Monitor request/response traffic between a web service client and the service.

- Monitor HTTP requests between Java applications and web resources.

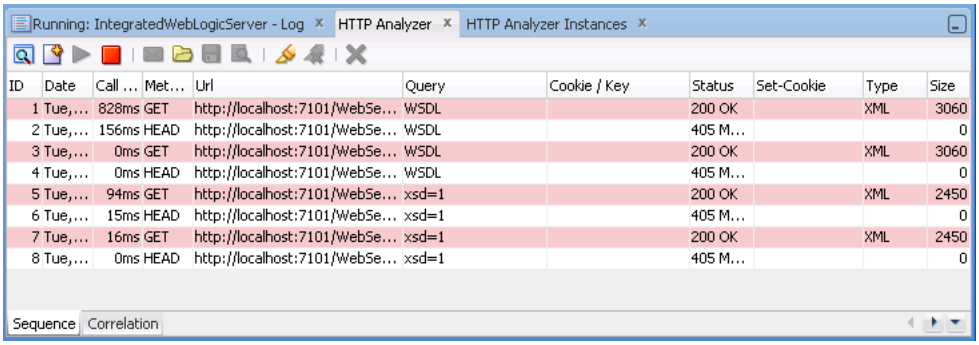
The HTTP Analyzer acts as a proxy between code in JDeveloper and the HTTP resource that the code is communicating with, and helps you to debug your application in terms of the HTTP traffic sent and received.

When you run the HTTP Analyzer, there are a number of windows that provide information for you.

8.3.1 How to Use the Log Window

When you open the HTTP Analyzer from the Tools menu, the HTTP Analyzer log window appears, illustrated in Figure 8–1. By default its position is at the bottom center of JDeveloper, alongside the other log windows.

Figure 8–1 HTTP Analyzer Log Window



When HTTP Analyzer runs, it outputs request/response messages to the HTTP Analyzer log window. You can group and reorder the messages:

- To reorder the messages, select the Sequence tab, then sort using the column headers (click on the header to sort, double-click to secondary sort).
- To group messages, click the Correlation tab.
- To change the order of columns, grab the column header and drag it to its new position.

Table 8–1 HTTP Analyzer Log Window Toolbar Icons

Icon	Name	Function
	**Analyzer Preferences**	Click to open the HTTP Analyzer Preferences dialog where you can specify a new listener port, or change the default proxy. An alternative way to open this dialog is to choose **Tools > Preferences**, and then navigate to the HTTP Analyzer page. For more information, see
	**Create New Request**	Click to open the HTTP Analyzer Test window, where you enter payload details, and edit and resend messages.
	**Start HTTP Analyzer**	Click to start the HTTP Analyzer running. The monitor runs in the background, and only stops when you click **Stop** or exit JDeveloper. If you have more than one listener defined clicking this button starts them all. To start just one listener, click the down arrow and select the listener to start.
	**Stop HTTP Analyzer**	Click to stop the HTTP Analyzer running. If you have more than one listener running, clicking this button stops them all. To stop just one listener click the down arrow and select the listener to stop.

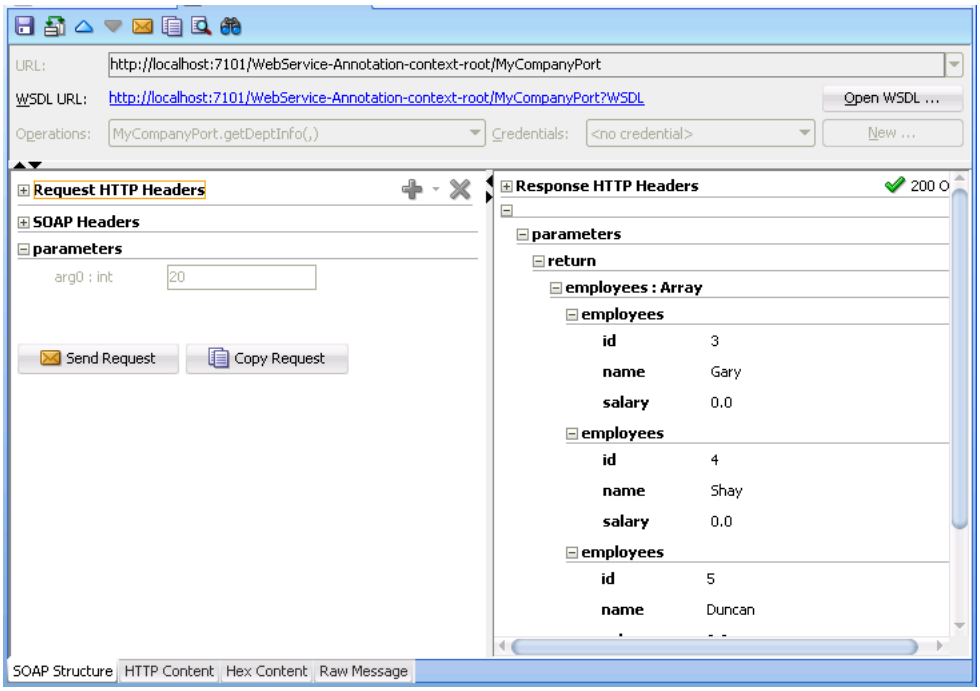
Table 8–1 (Cont.) HTTP Analyzer Log Window Toolbar Icons

Icon	Name	Function
	**Send Request**	Click to resend a request when you have changed the content of a request. The changed request is sent and you can see any changes in the response that is returned.
	**Open WS-I log file**	Click to open the Select WS-I Log File to Upload dialog, where you can navigate to an existing WS-I log file. For more information, see Section 16.13, "Monitoring and Analyzing Web Services."
	**Save Packet Data**	Click to save the contents of the HTTP Analyzer Log Window to a file.
	**WS-I Analyze**	Click to invoke the WS-I Analyze wizard which allows you to examine a web service at packet level. For more information, see Section 16.13, "Monitoring and Analyzing Web Services."
	**Select All**	Click to select all the entries in the HTTP Analyzer Log Window.
	**Deselect All**	Click to deselect all the entries in the HTTP Analyzer.
	**Clear Selected History (Delete)**	Click to clear the entries in the HTTP Analyzer.

8.3.2 How to Use the Test Window

An empty HTTP Analyzer test window appears when you click the **Create New Request** button in the HTTP Analyzer Log window. A test window showing details of the request/response opens when you choose **Test Web Service** from the context menu of a web service container in the Application Navigator, or when you double-click a line in the HTTP Analyzer Log Window, illustrated in Figure 8–2. By default, its position is in the center of JDeveloper, in the same place that the source editor appears.

Figure 8–2 HTTP Analyzer Test Window



The test window allows you examine the headers and parameters of a message. You can test the service by entering a parameter that is appropriate and clicking **Send Request**.

The tabs along the bottom of the test window allow you choose how you see the content of the message. You can choose to see the message as:

- The SOAP structure, illustrated in Figure 8–2.
- The HTTP code, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://annotation/">
    <env:Header/>
    <env:Body>
        <ns1:getDeptInfo>
            <arg0/>
        </ns1:getDeptInfo>
    </env:Body>
</env:Envelope>
```

- The hex content of the message, for example:

```
[000..015] 3C 3F 78 6D 6C 20 ... 3D 22 31    <?xml version="1
[016..031] 2E 30 22 20 65 6E ... 22 55 54    .0" encoding="UT
[032..047] 46 2D 38 22 3F 3E ... 6E 76 65    F-8"?> <env:Enve
[048..063] 6C 6F 70 65 20 78 ... 76 3D 22    lope xmlns:env="
```

- The raw message, for example:

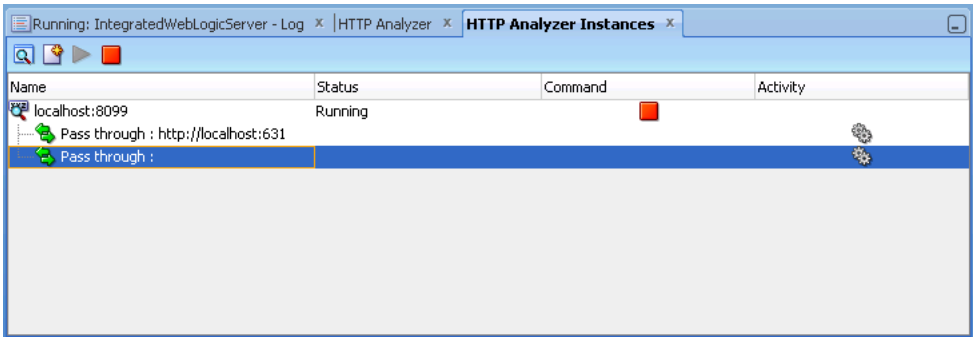
```
POST http://localhost:7101/WebService-Annotation-context-root/MyCompanyPort
HTTP/1.1
SOAPAction: ""
Content-Type: text/xml; charset=UTF-8
Host: localhost:7101
```

```
Content-Length: 277

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://annotation/">
    <env:Header/>
    <env:Body>
        <ns1:getDeptInfo>
            <arg0/>
        </ns1:getDeptInfo>
    </env:Body>
</env:Envelope>
```

8.3.3 How to Use the Instances Window

When you open the HTTP Analyzer from the **Tools** menu, the HTTP Analyzer Instances window appears. By default, its position is at the bottom center of JDeveloper, as a tab alongside the HTTP Analyzer log window. This window provides information about the instances of the HTTP Analyzer that are currently running, or that were running and have been stopped. The instance is identified by the host and port, and any rules are identified. You can start and stop the instance from this window.

Figure 8–3 HTTP Analyzer Instances Window



You create a new instance in the HTTP Analyzer dialog, which opens when you click the Create New Request button.

Table 8–2 HTTP Analyzer Instances Window Toolbar Icons

Icon	Name	Function
	**Analyzer Preferences**	Click to open the HTTP Analyzer dialog where you can specify a new listener port, or change the default proxy.
	**Create New Request**	Click to open a new instance of the HTTP Analyzer Test window, where you enter payload details, and edit and resend messages.
	**Start HTTP Analyzer**	Click to start the HTTP Analyzer running. The monitor runs in the background, and only stops when you click **Stop** or exit JDeveloper. If you have more than one listener defined clicking this button starts them all. To start just one listener, click the down arrow and select the listener to start.
	**Stop HTTP Analyzer**	Click to stop the HTTP Analyzer running. If you have more than one listener running, clicking this button stops them all. To stop just one listener click the down arrow and select the listener to stop.

8.3.4 What Happens When You Run the HTTP Analyzer

When you start the HTTP Analyzer, all Java processes and application server activity with JDeveloper will send their traffic via the HTTP Analyzer, using the proxy settings in the HTTP Analyzer dialog, which opens when you click the Start HTTP Analyzer button in the Instance or Log window, or from the HTTP Analyzer page of the Preferences dialog. By default, the HTTP Analyzer uses a single proxy on an analyzer instance (the default is 8099), but you can add additional proxies of your own if you need to.

Each analyzer instance can have a set of rules to determine behavior, for example, to redirect requests to a different host/URL, or to emulate a web service.

8.3.5 How to Specify HTTP Analyzer Settings

By default, the HTTP Analyzer uses a single proxy on an analyzer instance (the default is 8099), but you can add additional proxies of your own if you need to.

To set HTTP Analyzer preferences:

1. Open the HTTP Analyzer preferences dialog by doing one of the following:
   - Click the Start HTTP Analyzer button in the HTTP Analyzer Instances window or Log window.
   - Choose **Tools > Preferences** to open the Preferences dialog, and navigating to the HTTP Analyzer page.

   For more information at any time, press F1 or click **Help** from the HTTP Analyzer preferences dialog.
2. Make the changes you want to the HTTP Analyzer instance. For example, to use a different host and port number, open the Proxy Settings dialog by clicking **Configure Proxy**.

8.3.6 How to Use Multiple Instances

You can have more than one instance of HTTP Analyzer running. Each will use a different host and port combination, and you can see a summary of them in the HTTP Analyzer Instances window.

To add an additional HTTP Analyzer Instance:

1. Open the HTTP Analyzer preferences dialog by doing one of the following:
   - Click the Analyzer Preferences button in the HTTP Analyzer Instances window or Log window.
   - Choose **Tools > Preferences** to open the Preferences dialog, and navigating to the HTTP Analyzer page.

   For more information at any time, press F1 or click Help from the HTTP Analyzer preferences dialog.
2. To create a new HTTP Analyzer instance, that is a new listener, click **Add**. The new listener is listed and selected by default for you to change any of the values.

8.3.7 How to Configure External Web Browsers

You can use external web browsers to route messages through the HTTP Analyzer so that you can see the traffic between the web browser and client. This section describes how you can use a profile in Firefox so that when you start the HTTP Analyzer and

run an HTML or JSP or JSF page from within JDeveloper, a new instance of Firefox using the Debugger profile is started.

> **Note:** The steps below use the command `firefox`, which is correct for Linux. If you are using Windows, use `firefox.exe`.

To configure a Firefox profile for the HTTP Analyzer:

1. First you create a new Firefox profile. By default, starting Firefox from the command line opens a window on your currently open instance of Firefox, so you need to use -no-remote to create a separately configured instance Run the following from the command line

   ```
   firefox -no-remote -CreateProfile Debugging
   ```
2. Start Firefox using this profile

   ```
   firefox -no-remote -P Debugging
   ```
3. Next you configure JDeveloper to start this version of Firefox. From the main menu, choose **Tools > Preferences**.
4. In the Preferences dialog, select the Web Browser and Proxy node. For more information, press F1 or click **Help** from within the dialog page.
5. In the Browser Command Line, enter or browse to the correct location, and enter `firefox -no-remote -P Debugging`. JDeveloper underlines this in red, and when you close the dialog you will see a Command Line Validation Error warning which you can safely ignore.
6. Click **OK**. When you start the HTTP Analyzer and run an HTML or JSP or JSF page from within JDeveloper, a new instance of Firefox using the Debugger profile is started.

Click **OK**. When you start the HTTP Analyzer and run an HTML or JSP or JSF page from within JDeveloper, a new instance of Firefox using the Debugger profile is started.

8.3.8 Using SSL

You can use the HTTP Analyzer with secured services or applications, for example, web services secured by policies. JDeveloper comes with a set of preconfigured credentials, `HTTPS Credential`, which is always present. You cannot delete or edit `HTTPS Credential`, but you can copy it to create a new credential of the same type.

Once you have configured the credentials, you can choose which to use in the HTTP Analyzer Test window.

8.3.8.1 HTTPS Keystore

HTTPS encrypts an HTTP message prior to transmission and decrypts it upon arrival. It uses a public key certificate signed by a trusted certificate authority. When the integrated application server is first started, it generates a `DemoIdentity` that is unique to your machine, and the key in it is used to set up the HTTPS channel.

The client keystore identity is used for configuring HTTPS. The server keystore identity is used when the HTTP Analyzer is acting as a server; it is not used when connecting to a remote server.

For more information about keystores and keystore providers, see *Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server*.

When the default credential `HTTPS Credential` is selected, you need to specify the keystores that JDeveloper and the HTTP Analyzer should use when handling HTTPS traffic. Two keystores are required to run the HTTP Analyzer:

- The client keystore, containing the certificates of all the hosts to be trusted by JDeveloper and the Analyzer (client trust) when it makes onward connections.
- The server keystore, containing a key that the Analyzer can use to authenticate itself to calling clients (server keystore).

The client keystore is only required when mutual authentication is required.

8.3.8.2 Username Token

Username token is a way of carrying basic authentication information. You supply a username/password to provide authentication.

8.3.8.3 X509 Certificates

X509 is a PKI standard for single sign-on, where certificates are used to provide identity, and to sign and encrypt messages. You enter details of an X509 certificate. When you supply a valid keystore and the password for the keystore, the client key aliases are populated.

If JDeveloper has any problems finding and opening the keystore, error messages will be displayed.

8.3.8.4 STS Configuration

A Secure Token Service (STS) is a web service that issues and manages security tokens over HTTPS. You enter the Security Token Server provider URL and optionally a policy URL.

> **Note:** The client truststore must contain the server public key, otherwise when the HTTP Analyzer requests the SAML token it will fail.

8.3.8.5 How to Use HTTPS

To configure the HTTP Analyzer to use different HTTPS values:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select the Credentials node. For more information, press F1 or click **Help** from within the dialog page.
3. Enter the new keystore and certificate details you want to use.

8.3.8.6 How to Configure Credentials for Testing Web Service Policies

You can use the HTTP Analyzer to test web services that are secured using policies.

> **Note:** You cannot use the HTTP Analyzer with JAX-RPC web services that are secured with WebLogic 9.x policies. WebLogic 9.x policies are deprecated for JAX-RPC.

The HTTP Analyzer supports:

- HTTPS. The message is encrypted prior to transmission using a public key certificate that is signed by a trusted certificate authority. The message is decrypted on arrival.
- Username token. This is a way of carrying basic authentication information using a token based on username/password.
- X509. This is a PKI standard for single sign-on authentication, where certificates are used to provide identity, and to sign and encrypt messages.
- STS. Security Token Service (STS) is a web service which issues and manages security tokens.

You choose the credentials to use in the HTTP Analyzer Test window.

To add authentication information to the HTTP Analyzer:

1. Choose **Tools > Preferences** to open the Preferences dialog, and navigate to the Credentials page. For more information at any time, press F1 or click **Help** from the Preferences dialog.
2. Enter the authentication information that is appropriate for the web service.

8.3.9 How to Run the HTTP Analyzer

The HTTP Analyzer allows you to view the content of request and response HTTP messages.

To monitor HTTP packets:

1. Open the HTTP Analyzer by choosing **Tools > HTTP Analyzer**. The HTTP Analyzer docked window opens.
2. Start the HTTP Analyzer by clicking the Start HTTP Analyzer button. By default, this starts the listener on your localhost's hostname on port 8098. You can add new listeners, and use different hosts and ports, configure HTTPS, or set up rules to determine how the analyzer works.
3. Run the class, application, web service and so on that you want to analyze in the usual way.

   Each request and response packet is listed in the HTTP Analyzer Log window, and detailed in the HTTP Analyzer Test Window.

If you are using the HTTP Analyzer to examine how a web service developed in JDeveloper works, the HTTP Analyzer starts automatically when you choose Test Web Service from the context menu of the web service in the Application Navigator.

8.3.10 How to Debug Web Pages Using the HTTP Analyzer

You can use the HTTP Analyzer when you are debugging Web pages, such as HTML, JSP, or JSF pages. This allows you to directly examine the traffic that is sent back and forth to the browse.

To debug Web pages using the HTTP Analyzer:

1. Configure a browser to route messages through the HTTP Analyzer so that you can see the traffic between the web browser and client.
2. Start the HTTP Analyzer running.
3. Run the class, application, or Web page that you want to analyze in the usual way.

Each request and response packet is listed in the HTTP Analyzer Log window, and detailed in the HTTP Analyzer Test Window.

8.3.11 How to Edit and Resend HTTP Requests

You can edit the contents of a HTTP request and resend it. You can then examine the response to see whether the changes you expect have occurred.

To send a request:

1. In the Request pane of the HTTP Analyzer Test window, enter parameter values.
2. Click the Send Request button.
3. The processed value is returned in the Response pane.

To edit and resend a request:

1. In the Request pane of the HTTP Analyzer Test window, click **Copy Request**. This opens a new test window, where you can enter a new parameter to send.

   Alternatively, you can open a new test window by double-clicking a line in the HTTP Analyzer Log window.

8.3.12 How to Use Rules to Determine Behavior

You can set rules so that the HTTP Analyzer runs using behavior determined by those rules. You can set more than one rule in an HTTP Analyzer instance. If a service's URL matches a rule, the rule is applied. If not, the next rule in the list is checked. If the service does not match any of the rules the client returns an error. For this reason, you should always use a Pass Through rule with a blank filter (which just passes the request through) as the last rule in a list to catch any messages not caught by the preceding rules.

The types of rule available are:

- Pass Through Rule
- Forward Rule
- URL Substitution Rule
- Tape Rule

8.3.12.1 Using the Pass Through Rule

The Pass Through simply passes a request on to the service if the URL filter matches. When you first open the Rule Settings dialog, two Pass Through Rules are defined:

- The first has a URL filter of `http://localhost:631` to ignore print service requests.
- The second has a blank URL filter, and it just which just passes the request to the original service. This rule should normally be moved to end of the list if new rules are added.

8.3.12.2 Using the Forward Rule

The Forward rule is used to intercept all URLs matched by the filter and it forwards the request on to a single URL.

8.3.12.3 Using the URL Substitution Rule

The URL Substitution rule allows you to re-host services by replacing parts of URL ranges. For example, you can replace the machine name when moving between the integrated application server and Oracle WebLogic Server.

8.3.12.4 Using the Tape Rule

The tape rule allows you to run the HTTP Analyzer in simulator mode, where a standard WS-I log file is the input to the rule. When you set up a tape rule, there are powerful options that you can use:

- Loop Tape, which allows you to run the tape again and again.
- Skip to matching URL and method, which only returns if it finds a matching URL and HTTP request method. This means that you can have a WSDL and an endpoint request in the same tape rule.
- Correct header date and Correct Content Size, which allow you change the header date and content size of the message to current values so that the request does not fail.

An example of using a tape rule would be to test a web service client developed to run against an external web service.

To test a web service client developed to run against an external web service:

1. Create the client to the external web service.
2. Run the client against the web service with the HTTP Analyzer running, and save the results as a WS-I log file.

   You can edit the WS-I file to change the values returned to the client.
3. In the HTTP Analyzer page of the Preferences dialog, create a tape rule.

   Ensure that it is above the blank Pass Through rule in the list of rules.
4. In the Rule Settings dialog, use the path of the WS-I file as the Tape path in the Rule Settings dialog.

   When you rerun the client, it runs against the entries in the WS-I file instead of against the external web service.

   There are other options that allow you to:

   - Correct the time and size of the entries in the WS-I log file so the message returned to the client is correct.
   - Loop the tape so that it runs more than once.
   - Skip to a matching URL and HTTP request method, so that you can have a WSDL and an endpoint request in the same tape rule.

   Note: Tape Rules will not work with SOAP messages that use credentials or headers with expiry dates in them.

8.3.13 How to Set Rules

You can set rules so that the HTTP Analyzer runs using behavior determined by those rules. Each analyzer instance can have a set of rules to determine behavior, for example, to redirect requests to a different host/URL, or to emulate a web service.

To set rules for an HTTP Analyzer instance:

1. Open the HTTP Analyzer by choosing **Tools > HTTP Analyzer**. The HTTP Analyzer docked window opens.

   Alternatively, the HTT Analyzer automatically opens when you choose **Test Web Service** from the context menu of a web service container in the Application Navigator.

2. Click the Analyzer Preferences button to open the HTTP Analyzer preferences dialog, in which you can specify a new listener port, or change the default proxy.

   Alternatively, choose **Tools > Preferences**, and then navigate to the HTTP Analyzer page.

3. Click **Configure Rules** to open the Rule Settings dialog in which you define rules to determine the actions the HTTP Analyzer should take. For more help at any time, press F1 or click **Help** in the Rule Settings dialog.

4. In the Rule Settings dialog, enter the URL of the reference service you want to test against as the Reference URL. This will help you when you start creating rules, as you will be able to see if and how the rule will be applied.

5. Define one or more rules for the service to run the client against. To add a new rule, click the down arrow next to **Add**, and choose the type of rule from the list. The fields in the dialog depend on the type of rule that is currently selected.

6. The rules are applied in order from top to bottom. Reorder them using the up and down reorder buttons. It is important that the last rule is a blank Pass Through rule.

8.3.14 Using the HTTP Analyzer with Web Services

This section contains information about using the HTTP Analyzer with web services developed in JDeveloper. In general, you can use HTTP Analyzer to examine the content of web services in the same way as using it to examine any packets across HTTP.

Note: You cannot use the HTTP Analyzer to test JAX-RPC web services that have WebLogic Server 9.x policies attached. WebLogic 9.x policies have been deprecated in JAX-RPC.

8.3.14.1 Testing Web Services with the HTTP Analyzer

JDeveloper allows you to test web services using the HTTP Analyzer to examine the network traffic of a proxy connecting to a web service developed in JDeveloper.

To test a web service:

1. Run the web service on the integrated application server and open the HTTP Analyzer by right-clicking the web service node in the Application Navigator, and choosing **Test Web Service**. JDeveloper automatically:

   - Starts the integrated application server, if it is not already running.
   - Compiles and binds the web service application to the integrated application server, which you can see in the Application Server Navigator.
   - Displays a Log window for the integrated application server (if there is not one already open).

2. Enter a parameter to test the service in the Request pane of the HTTP Analyzer Test window and click **Send Request**.

   The response from the deployed web service is displayed in the Response pane of the HTTP Analyzer Test window.

You can examine the contents of the HTTP headers of the request and response packets to see the SOAP structure, the HTTP content, the Hex content or the raw message contents by choosing the appropriate tab at the bottom of the HTTP Analyzer Test window.

8.3.14.2 Using the HTTP Analyzer with RESTful Web Services

You can use the HTTP Analyzer to interact with RESTful web services. Representational State Transfer (REST) describes any simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. REST provides a set of design rules for creating stateless services that are viewed as resources, or sources of specific information, and can be identified by their unique URIs. A client accesses the resource using the URI, a standardized fixed set of methods, and a representation of the resource is returned. The client is said to transfer state with each new resource representation.

When using the HTTP protocol to access RESTful resources, the resource identifier is the URL of the resource and the standard operation to be performed on that resource is one of the HTTP methods: `GET`, `PUT`, `DELETE`, `POST`, or `HEAD`.

The HTTP Analyzer has support for Hypermedia as the Engine of Application State (HATEOAS), and so you can examine and test RESTful web services using the HTTP Analyzer.

Jersey and WADL

Before you can create RESTful web services in JDeveloper, you need to download and add to your project the Jersey JAX-RS Reference Implementation (RI).

A Web Application Description Language (WADL) is an XML file created by Jersey that provides a description of the resources in the servlet. For more information about WADL, see https://wadl.dev.java.net/.

Testing a RESTful Service

An outline of testing a RESTful service using WADL is given here, with more detailed steps in the procedure below. Not all RESTful services work this way. The HTTP Analyzer reads a WADL created by Jersey for the RESTful web service, and you examine the WADL in the HTTP Analyzer Test window. From the WADL, you can open an instance of the HTTP Analyzer Test window directly from a method, and test the method by entering a parameter and posting it to the service. The HTTP Analyzer redirects the response to a new URL which it displays, and when you click on it another instance of the HTTP Analyzer Test window opens with the response. Once you have finished, you use the WADL to locate the new resource that the HTTP Analyzer created to test the service and delete it.

Example 8–1 provides an example of a WADL document which uses `POST`, `GET` and `DELETE`.

Example 8–1 Simple Example of WADL

```
  <?xml version = '1.0' encoding = 'UTF-8' standalone = 'yes'?>
  <application xmlns="http://research.sun.com/wadl/2006/10">
     <doc xmlns:jersey="http://jersey.dev.java.net/"  jersey:generatedBy="Jersey: 1.1.0-ea
04/30/2009 04:46 PM"/>
     <resources  base="http://localhost:7101/RESTDemo-ContainerProject-context-root/jersey/">
```

```
        <resource path="buckets">
           <method name="POST" id="createNewBucket">
              <request>
                 <representation mediaType="*/*"/>
              </request>
              <response>
                 <representation mediaType="*/*"/>
              </response>
           </method>
           <method name="GET" id="getBuckets">
              <response>
                 <representation  mediaType="application/buckets+xml"/>
              </response>
           </method>
           <resource path="/{id}">
              <param xmlns:xs="http://www.w3.org/2001/XMLSchema"  type="xs:int" style="template"
name="id"/>
              <method name="DELETE" id="delete">
                 <response>
                    <representation mediaType="*/*"/>
                 </response>
              </method>
              <method name="GET" id="getBucket">
                 <response>
                    <representation mediaType="*/*"/>
                 </response>
              </method>
           </resource>
        </resource>
     </resources>
  </application>
```

To test a REST web service

To test a REST web service requires that you:

- examine the RESTful service
- test the service
- work with the resource

To examine the RESTful service:

1. Run the REST web service on the integrated application server.
2. Right-click the web service node in the Application Navigator, and choose Test Web Service. JDeveloper automatically:
   - Starts the integrated application server, if it is not already running.
   - Compiles and binds the web service application to the integrated application server instance, which is the IntegratedWebLogicServer node in the Application Server Navigator.
   - Displays a Log window for the integrated application server (if there is not one already open).
3. Click the HTTP Content tab in the HTTP Analyzer Test window. RESTful web services do not use SOAP, so you will not use the SOAP Structure tab.
4. In the Log window for the integrated application server, click the link next to Target Application WADL. A second instance of the test window opens. Notice that the URL displays the WADL, and the Method is GET.

5. Click **Send Request**. The GET method is used to return the content of the WADL so that it is displayed in the Response pane.

   If necessary, use the left arrow to maximize the width of the pane to see the code more clearly.

To test the RESTful service:

1. In the WADL displayed in the Response pane, press Ctrl+mouse-click to use the Go to declaration feature to reveal parts of the HTTP message that can be accessed. Click on a POST method that is now revealed as a link. This opens a new instance of the test window.
2. Enter a parameter in the Request pane, and click **Send Request**. The POST method is used, and the Request pane displays a `201 Created HTTP` status code along with the location of the URL that contains the response.
3. Click on the URL in the Response pane. Another instance of the test window opens. Notice that the URL displays the redirected URL, and the Method is GET. Click **Send Request**, and the response to the parameter you entered is displayed in the Request pane.

> **Note:** When you click on the WADL, the correct content-type and accept headers will be generated.

To work with the resource:

1. Select the test window instance for the WADL, and navigate to the GET method. Press Ctrl+mouse-click to open a new instance of the test window. Notice that the URL displays the redirected URL, and the Method is GET.
2. You can update the resource by choosing PUT from the Method list, and click **Send Request**.
3. In order to delete this resource, choose DELETE from the Method list, and click Send Request.

8.3.15 Using the HTTP Analyzer with WebSockets

The HTTP Analyzer will pass unsecured WebSockets requests via a proxy.

The content of the request response stream will be available in the HTTP Analyzer after you close and reopen the message. The WebSockets messages are those with a response code of 101.

8.3.16 Reference: Troubleshooting the HTTP Analyzer

This section contains information to help resolve problems that you may have when running the HTTP Analyzer.

8.3.16.1 Running the HTTP Analyzer While Another Application is Running

If you have an application waiting for a response, do not start or stop the HTTP Analyzer. Terminate the application before starting or stopping the HTTP Analyzer.

8.3.16.2 Changing Proxy Settings

When you use the HTTP Analyzer, you may need to change the proxy settings in JDeveloper. For example:

- If you are testing an external service and your machine is behind a firewall, ensure that the JDeveloper is using the HTTP proxy server.
- If you are testing a service in the integrated application server, for example when you choose **Test Web Service** from the context menu of a web service in the Application Navigator, ensure that JDeveloper is not using the HTTP proxy server.

If you run the HTTP Analyzer, and see the message

```
500 Server Error
The following error occurred: [code=CANT_CONNECT_LOOPBACK] Cannot connect due to
potential loopback problems
```

you probably need to add `localhost|127.0.0.1` to the proxy exclusion list.

To set the HTTP proxy server and edit the exception list:

1. Choose **Tools > Preferences**, and select **Web Browser/Proxy**.
2. Ensure that **Use HTTP Proxy Server** is selected or deselected as appropriate.
3. Add any appropriate values to the Exceptions list, using | as the separator.

   In order for Java to use localhost as the proxy `~localhost` must be in the Exceptions list, even if it is the only entry.

8.4 Profiling Applications

The Profiler monitors and logs a running program's use of processor and memory resources. It gathers statistics that enables you to more easily diagnose the performance issues and correct the inefficiencies in your code.

JDeveloper offers two kinds of profilers: The CPU Profiler and the Memory Profiler, for local as well as remote profiling.

- The CPU Profiler is used to analyze your application's impact on the processor. Use the CPU Profiler to test functions of your application, such as startup and initialization, repainting, and compiling.
- The Memory Profiler provides a visual and statistical analysis of how your program utilizes memory in the Java heap. Use the Memory Profiler to track down and isolate memory leaks in your program.

For more information on the profiler, including steps to profile your application and project, see Section 18.10, "Profiling a Project".

9 Deploying Applications

This chapter describes how to run and debug applications using the JDeveloper integrated application server, and how to deploy applications to a target application server, for example to Oracle WebLogic Server or to a third-party server.

This chapter includes the following sections:

9.1 About Deploying Applications

Deployment is the process of packaging application files as an archive file and transferring it to a target application server. You can use JDeveloper to deploy Java or Java EE applications directly to the application server (such as Oracle WebLogic Server or IBM WebSphere), or indirectly to an archive file as the deployment target, and then install this archive file to the target server. For application development, you can also use JDeveloper to run an application in the integrated application server. JDeveloper supports deploying to server clusters, but you cannot use JDeveloper to deploy to individual Managed Servers that are members of a cluster.

If you are using Oracle® Fusion Middleware extensions, refer to the appropriate developer's guide for deployment information specific to the product. For example:

- If you are deploying an ADF Fusion Web application, see the "Deploying Fusion Web Applications" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- If you are deploying an ADF Java EE application, see the "Deploying an ADF Java EE Application" chapter in the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

You can deploy applications in the following ways:

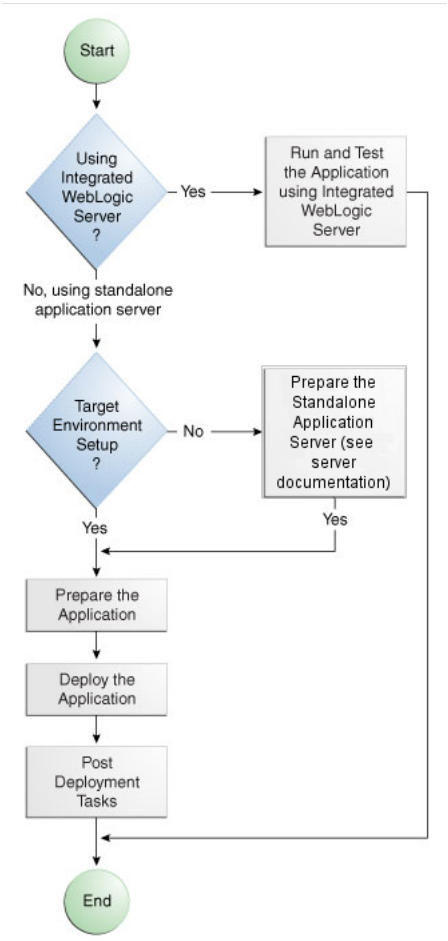
- Directly to an application server through an application server connection.
- To an archive file. You can deploy applications indirectly by choosing an archive file as the deployment target. The archive file can subsequently be installed on the target Java EE application server.
- To a test environment using the JDeveloper integrated application server, a Java EE runtime service used for running and testing JDeveloper applications and projects as Java EE applications and modules within a Java EE container.

Note: Normally, you use JDeveloper to deploy applications for development and testing purposes. If you are deploying applications for production purposes, you can use Enterprise Manager or scripts to deploy to production-level application servers.

For more information about deployment to later-stage testing or production environments, see the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

Figure 9–1 shows the flow diagram that describes the overall deployment process. Note that preparing the target application server for deployment is outside the scope of this guide; you should refer to the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework* for deployment to Oracle WebLogic Server, or to the appropriate documentation for a third-party application server.

Figure 9–1 Deployment Overview Flow Diagram



Java and Java EE applications are based on standardized, modular components and can be deployed to the following application servers:

- Oracle WebLogic Server

  Oracle WebLogic Server provides a complete set of services for those modules and handles many details of application behavior automatically, without requiring programming.

- A third-party application server, that is an application server provided by a vendor other than Oracle:

  – Apache Tomcat

  – IBM WebSphere

  – JBoss

  For information about which versions of Oracle WebLogic Server, Tomcat, WebSphere, or JBoss are compatible, see the JDeveloper Certification Information at `http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html`.

You can use JDeveloper to:

- Run applications in the integrates application server

You can run and debug applications using Integrated WebLogic Server and then deploy to a standalone WebLogic Server or to a third party server.

- Deploy directly to the standalone application server

  You can deploy applications directly to the standalone application server by creating a connection to the server and choosing the name of that server as the deployment target.

- Deploy to an archive file

  You can deploy applications indirectly by choosing an EAR file as the deployment target. The archive file can subsequently be installed on a target application server.

Deployment can be an iterative process where refinements to the application, or corrections to issues in the deployed application, require redeployment to either the test deployment environment, archive file, or application server. The process of deploying an application from JDeveloper can involve a number of processes.

9.1.1 Developing Applications with the Integrated Application Server

JDeveloper is bundled with an integrated application server called Integrated WebLogic Server and a default connection called `IntegratedWebLogicServer` is defined for it. The integrated application server is a Java EE runtime for services using deployment optimized for the iterative code development cycle. You can use it for running and testing JDeveloper applications and projects as Java EE applications and modules within a Java EE container, as well as for post-run services such as launching a browser or tester. JDeveloper has a default connection to the integrated application server and does not require any deployment profiles or descriptors. In most cases, deploying to the integrated application server is a one-click operation, for example, running a web service by choosing **Run** from the right-mouse menu of the web service in the Application Navigator, or running an application by choosing **Run** from the JDeveloper main menu.

You debug the application using the features described in Chapter 7, "Building, Running and Debugging Applications."

9.1.2 Developing Applications to Deploy to Standalone Application Servers

Typically, for deployment to standalone application servers, you test and develop your application by running it in the integrated application server. You can then test the application further to more closely simulate the production environment by deploying it to standalone Oracle WebLogic Server in development mode or to a third-party application server.

In general, you use JDeveloper to prepare the application or project for deployment by:

- Creating a connection to the target application server
- Creating deployment profiles (if necessary)
- Creating deployment descriptors (if necessary, and that are specific to the target application server)
- Updating *application.xml* and *web.xml* to be compatible with the application server (if required)
- Migrating application-level security policy data to a domain-level security policy store

You must already have an installed application server. For Oracle WebLogic Server, you can use the Oracle 11*g* Installer or the Oracle Fusion Middleware 11*g* Application Developer Installer to install one. For other applications servers, follow the instructions in the applications server documentation to obtain and install the server.

If necessary, you must prepare the application server by creating a global JDBC data source for applications that require a connection to a data source.

After the application and application server have been prepared, you can:

- Use JDeveloper to:
  - Directly deploy to the application server using the deployment profile and the application server connection.
  - Deploy to an EAR file using the deployment profile. WAR and MAR files can be deployed only as part of an EAR file.
- Use Enterprise Manager, scripts, or the application server's administration tools to deploy the EAR file created in JDeveloper. For more information, see the *Oracle Fusion Middleware Administrator's Guide for Oracle Application Development Framework*.

9.1.3 Understanding the Archive Formats

A Java EE archive file contains a Java EE module or application. A module consists of one or more JDeveloper projects of a common component type, which have been configured for deployment. An application is comprised of one or more modules. An archive also contains a deployment descriptor, which is an XML file that describes the configuration of the module or application to the server, and is specific to the type of server. A deployment descriptor can be server specific or generic for Java EE servers.

JAR, EJB JAR, and WAR files each contain a module consisting of one or more components. An Enterprise Archive (EAR file) contains an application consisting of one or more modules.

When you create a web (servlet, JSP, JSF, and ADF Faces) or EJB application and deploy it via an application server connection, JDeveloper packages it as a WAR or EJB JAR, which you can optionally wrap in an EAR file. If your application consists of components of differing types, the components will be packaged into multiple modules, which you can deploy independently or assembled as an EAR file.

9.1.4 Understanding Deployment Profiles

Deployment profiles are application or project properties that govern the deployment of a project or application. A deployment profile names the source files, deployment descriptors, and other auxiliary files that will be packaged, the type and name of the archive file to be created, dependency information, platform-specific instructions, and other information.

9.1.5 Understanding Deployment Descriptors

Deployment descriptors define the content and organization of the deployed applications. Deployment descriptor files that are required by an application depend on the technologies the application uses and on its target application server.

9.1.6 Configuring Deployment Using Deployment Plans

You can control how an application is deployed using a deployment plan which allows you to make configuration adjustments in the application deployment descriptors `web.xml`, `weblogic.xml`, `application.xml`, and `weblogic-application.xml`.

Deployment plans are controlled using a descriptor called `plan.xml`. Only Weblogic deployment descriptor configuration can be customized using `plan.xml`. The primary use case for deployment customization is to modify Weblogic specific application configuration for different servers being deployed without requiring modification of the base Weblogic descriptor. For more information, see the section on Deployment Plans in *Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server*.

9.1.7 Deploying from the Java Edition

If you are using the Java edition of JDeveloper, which contains only the core Java and XML features, the only deployment actions you can perform are:

- Creating a simple JAR archive which you can then manually deploy to a server.

  JDeveloper Java Edition provides the facility to package applications into a JAR file. The deployment dialog in Java Edition allows for only limited configuration of standard JAR options such as specifying JAR name, file groupings, or dependencies on other deployment profiles. Any application that requires more configuration than this must be deployed from the Studio edition of JDeveloper.

- Creating deployment profiles as part of extension development. For more information about creating extensions to JDeveloper, see *Oracle Fusion Middleware Developer's Guide for Oracle JDeveloper Extensions*

9.2 Running Java EE Applications in the Integrated Application Server

JDeveloper is installed with Integrated WebLogic Server, an integrated application server which you can use to test and develop your application. For most development purposes, the integrated application server will suffice. When your application is ready to be tested, you can select the run target and then choose the **Run** command from the main menu.

> **Note:** The first time you start the integrated application server by running or debugging a project, file, or web service, a dialog is displayed where you enter a password for the administrator ID on the default domain. When you click **OK**, the default domain is created. You only need to do this once.

When you run the application target, JDeveloper detects the type of Java EE module to deploy based on artifacts in the projects and application workspace. JDeveloper then creates an in-memory deployment profile for deploying the application to the integrated application server. JDeveloper copies project and application workspace files to an "exploded EAR" directory structure. This file structure closely resembles the EAR file structure that you would have if you were to deploy the application to an EAR file. JDeveloper then follows the standard deployment procedures to register and deploy the "exploded EAR" files into the integrated application server. The "exploded EAR" strategy reduces the performance overhead of packaging and unpackaging an actual EAR file.

In summary, when you select the run target and run the application in the integrated application server, JDeveloper:

- Detects the type of Java EE module to deploy based on the artifacts in the project and application
- Creates a default deployment profile (that is, without customizations) in memory
- Copies project and application files into a working directory with a file structure that simulate the "exploded EAR" file of the application.
- Performs the deployment tasks to register and deploy the simulated EAR into the integrated application server
- Automatically migrates identities, credentials, and policies. If you plan to deploy the application to a standalone Oracle WebLogic Server instance, you will need to migrate this security information.

> **Note:** When you run the application in the integrated application server, JDeveloper ignores the deployment profiles that have been created for the application.

The application will run in the base domain in the integrated application server. The base domain has the same configuration as a base domain in a standalone Oracle WebLogic Server instance. In other words, this base domain is the same as if you had used the Oracle® Fusion Middleware Configuration Wizard to create a base domain with the default options in a standalone Oracle WebLogic Server instance.

JDeveloper extends this base domain with the necessary domain extension templates, based on the JDeveloper technology extensions. For example, if you have installed JDeveloper Studio, JDeveloper will automatically configure the integrated application server environment with the ADF runtime template (JRF Fusion Middleware runtime domain extension template).

You can explicitly create additional default domains for the integrated application server which you can use to run and test your applications in addition to using the default domain. Open the Application Server Navigator, right-click **IntegratedWebLogicServer** and choose **Create Default Domain**.

9.2.1 Understanding the Integrated Application Server Log Window

The output messages generated when running or debugging an application in the integrated application server are displayed in a log window which has a title of either Running: IntegratedWebLogicServer or Debugging: IntegratedWebLogicServer.

The content of the Integrated WebLogic Server Log Window includes:

- Status log messages about the server and the applications running on the server
- Output from the integrated application server instance's console (in color)
- Messages generated from deploying the application to the integrated application server
- Messages that log the Java EE archives (EAR, WAR, and EJB JAR) as they are created. You can click on the links in the log window to browse the generated archives.

The generated log files are located at `jdeveloper-user-home/DefaultDomain/server/DefaultServer/logs`.

You can configure diagnostic logging parameters in the `logging.xml` file. Transient loggers can only be added while the server is running in debug mode.

You can control the level of information sent to the log file using the `-verbose` element in the `jsp-descriptor` and `logging` elements of `weblogic.xml`. For more information, see the weblogic.xml descriptor elements information in *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*

9.2.2 Rules Governing Deployment to the Integrated Application Server

Deployment to the integrated application server uses default deployment profiles which rely on project metadata for the default mappings. Default contributors to the profiles are based on project dependencies, and the rules governing dependencies are:

1. If project A depends on the build output of project B, then the build output of project B is merged into project A. If project A is a web application, this means the build outputs of project A and project B are both copied into `WEB-INF/classes` of the resulting WAR.

   Merging implies that you can only have one copy of any particular URI, because it can only exist once within `WEB-INF/classes`.

2. If project A depends on the deployment profile of project B, for example a JAR profile, then the result of that deployment profile is included in the `WEB-INF/lib` of the resulting WAR.

3. A project containing a `WEB-INF/web.xml` is recognized as a web project and a default WAR profile is created for it.

4. A project that contains at least one session EJB bean is recognized as an EJB project and a default EJB JAR profile is created for it.

5. All libraries marked Deploy by Default for a web project are deployed as a web application library (in the `WEB-INF/lib` of the WAR).

6. All libraries marked Deploy by Default for an EJB project are deployed as an application library (in the `lib` of the EAR).

7. If an EJB Project A depends on the build output of Project B, the build output (e.g. `classes` directory) of Project B is merged with the build output of Project A and deployed in the root directory of the EJB JAR.

9.2.3 Working with Integrated Application Servers

The definition of an integrated application server controls the interaction of the instance with JDeveloper and your computer system.

JDeveloper is bundled with an integrated application server called Integrated WebLogic Server, and a default instance called `IntegratedWebLogicServer` is defined for it. All applications are bound by default to `IntegratedWebLogicServer`.

You can modify the properties of the integrated application server that an application is bound to.

Note: WebLogic Server domains used as integrated application servers must be collocated on the same host as the JDeveloper process.

To modify the properties of the integrated application server that an application is bound to:

1. In the Application Navigator, select a project.
2. Choose **Application > Application Properties**.
3. Select **Run** from the left panel.
4. Select an existing integrated application server in **Bind Application to Server Instance**, or click **Application Server Properties** to open the Application Server Properties dialog, where you can change some properties for the integrated application server.
5. Define the other options for the integrated application server, including startup and shutdown options. For more information, press F1 or click **Help** from within the dialog.

You can create a new integrated application server instances.

9.2.3.1 How to Create a New Integrated Application Server Connection

To define an integrated server connection:

1. In the Application Server Navigator, right-click **Application Servers** and choose **New Application Server**. The Create Application Server Connection wizard opens. For more information at any time, press F1 or click **Help** from within the wizard.
2. On the Usage page, select **Integrated Server**. If you want to manage the server from within JDeveloper, select **Let** JDeveloper **manage the lifecycle for this Server Instance** on the Name and Domain page, and provide the **Domain** and **Server Instance** directories.
3. Complete the wizard.

9.2.3.2 How to Run and Debug with an Integrated Application Server

By default, the integrated application server is automatically started when you run or debug an EJB, servlet, HTML, web service, or JSP project. Alternatively, you can start the integrated application server by clicking **Start Server Instance** or **Debug Server Instance** from the **Run** menu.

After it has been started, an integrated application server does not terminate automatically when you terminate a running Java EE application. Therefore, you can select an object, such as a JSP or a servlet in the Application Navigator, and choose an option from the **Run** menu.

You can run or debug a working set, which is a group of files created by applying a named filter to a project, by choosing the **Use Current Working Set (Java JEE Only)** option from the **Run** menu.

Once this is enabled, when you select **Run** or **Debug** from the context menu of the source editor or from a node in the Application Navigator, it is the current working set that is run or debugged.

Only a single integrated application server can be run at any given time. Thus, if you attempt to start another instance of the server, JDeveloper will shut down the previous instance and restart the instance in order to perform the requested task on the selected icon in the Navigator. After an integrated application server is started, multiple applications can run on it independently of each other. If an application is running, rerunning the application redeploys the up-to-date version of the application.

To run in an integrated application server, an application must be bound to a server instance. JDeveloper is supplied with a WebLogic Server domain, and a default server instance named `DefaultServer` is defined for it. The unique integrated application server connection defined for this integrated application server is called `IntegratedWebLogicServer`, and has the Domain Home defined as the system directory `$SYSTEM_ROOT/DefaultDomain`. All applications are bound by default to `IntegratedWebLogicServer`.

9.2.3.3 Working with the Default Domain

If you have not explicitly created the integrated application server's default domain, it will automatically be created with default settings when you start the server by running or debugging an application.

Alternatively, you can explicitly create the default domain from the Application Server Navigator.

If necessary, you can delete the existing default domain so that you can create it again to use new values.

To explicitly create the integrated application server's default domain:

1. If necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click the integrated application server connection `IntegratedWebLogicServer` and choose **Create Default Domain**. The Configure Default Domain dialog opens, where you can accept the defaults, or explicitly set other values, such as choosing a different listen address. For more information at any time, click **Help** or press F1 from the Configure Default Domain dialog.

When you install extensions to JDeveloper you may have to update the integrated application server's default domain.

To update the integrated application server's default domain:

1. necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click the integrated application server connection `IntegratedWebLogicServer` and choose **Update Default Domain**.

If you have already created the default domain, but you need to use specific settings you can delete the existing default domain and create it again.

To delete the integrated application server's default domain:

- With JDeveloper closed, locate the system folder in the file system and delete it. When you restart JDeveloper, you can create a new default domain for the integrated application server.

After the server has started, click the Run Manager tab in the navigator, or select Run Manager from the View menu, to display the integrated application server process.

Note: You can run more than one application simultaneously on a server in run mode, however you can only debug one application at a time in debug mode. To return JDeveloper back into non-debug editing mode, the integrated application server must be shut down.

9.2.3.4 One-Click Running of Applications in the Integrated Application Server

You can test an application by running it in the integrated application server. You can also set breakpoints and then run the application with the integrated application server in debug mode. For more information about running and debugging, see Chapter 7, "Building, Running and Debugging Applications."

To run an application in the integrated application server:

1. In the Application Navigator, select the run target, for example a project, web service, unbounded task flow, or JSF page.
2. Right-click the run target and choose **Run** or **Debug**. Alternatively, choose **Run** or **Debug** from the main menu.

   The first time you start the integrated application server by running or debugging an application, a dialog is displayed where you enter a password for the default user `weblogic` on the default domain. When you click **OK**, the default domain is created. You only need to do this once.

Application-level and Global Data Sources

If you are deploying to an integrated application server, you can use application level data sources or global data sources.

For both one-click deployment to an integrated application, JDeveloper ensures that your web application `web.xml`, or EJB application `ejb-jar.xml`, contains the necessary `<resource-ref>` entry to identify an application resource name. The name is `jdbc/connection-nameDS`, where `connection-name` is the name of the application resources connection.

The application looks up this data source using the application-specific resource JNDI namespace of `java:comp/env/jdbc/connection-nameDS`, and it finds this resource because `web.xml` contains the `<resource-ref>` entry for `jdbc/connection-nameDS`.

To use application level data sources in one-click deployment to Integrated WebLogic Server, select **Auto Generate JDBC Connections When Running Application** in JDeveloper on the WebLogic page of the Application Properties dialog (available from the **Application** menu). This:

- Generates a file called `connection-name-jdbc.xml` in the /META-INF directory of the application's EAR file
- Creates a corresponding `<module>` entry in the `weblogic-application.xml` file in META-INF that references this JDBC module

If the application uses more than one application resources database connection, then a `connection-name-jdbc.xml` file will be created for each, and there will be a similar number of `<module>` entries in the `weblogic-application.xml` file.

To use global data sources in one-click deployment to Integrated WebLogic Server, deselect Auto **Generate JDBC Connections When Running Application** in JDeveloper on the WebLogic page of the Application Properties dialog (available from the **Application** menu), and:

1. Connect to the Integrated WebLogic Server Administration Console, described in Section 9.2.3.9, "How to Log In to the Integrated WebLogic Server Administration Console"
2. Create the global data source in a similar manner to creating one on Oracle WebLogic Server, see Section 9.3.6.4, "Setting Up JDBC Data Sources on Oracle WebLogic Server"

9.2.3.5 How to Start the Integrated Application Server

By default, the integrated application server is automatically started when you run or debug an EJB, servlet, or JSP project. Therefore, you can select an object, such as a JSP or a servlet in the Navigator, and choose an option from the **Run** menu.

> **Note:** The first time you start the integrated application server by running or debugging a project, file, or web service, a dialog is displayed where you enter a password for the administrator ID on the default domain. When you click **OK**, the default domain is created. You only need to do this once.

Only a single integrated application server can be run at any given time. Thus, if you attempt to start another instance of the server, JDeveloper will shut down the previous instance and restart the instance in order to perform the requested task on the selected icon in the Navigator.

After the server has started, click the Run Manager tab in the Navigator to display the integrated application server process. You can open the Run Manager by choosing **View > Run Manager** from the main menu.

To start an integrated application server:

1. If necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click the Integrated WebLogic Server connection and choose **Start Server Instance**.

Alternatively, choose **Run > Start Server Instance** from the main menu.

To start an integrated application server in debug mode:

1. If necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click the Integrated WebLogic Server connection and choose **Debug Server Instance**.

Alternatively, choose **Run > Debug Server Instance** from the main menu.

9.2.3.6 How to Cancel a Running Deployment

If you are running a large application on the integrated application server, you can cancel it before it has finished deploying.

To cancel a running deployment:

- n the Log Window, click the Terminate button and choose the profile or application you wish to cancel.

9.2.3.7 How to Terminate an Integrated Application Server

After an integrated application server has started, the integrated application server process appears in the Run Manager. For more information, see Section 19.2, "Understanding the Run Manager."

You can open the Run Manager by choosing **View > Run Manager** from the main menu.

Note: Applications deployed on an integrated application server are automatically undeployed whenever the integrated application server is terminated.

The default behavior is to undeploy all the applications, but you can change the behavior.

To shutdown the running integrated application server:

Do one of the following:

- Choose **Run > Terminate > IntegratedWebLogicServer** (or the integrated application server connection name) from the main menu.
- Select the integrated application server name from the **Terminate** dropdown list in the toolbar.
- Choose **View > Run Manager** from the main menu. Right-click the integrated application server name and choose **Terminate**.
- Choose **File > Exit** to exit JDeveloper. Click **Yes** when prompted to terminate the instance's process.
- In the Application Server Navigator, right click on the integrated application server connection and select **Terminate Server Instance**.

To force shutdown of Integrated WebLogic Server:

- If you need to force shutdown of Integrated WebLogic Server, press the **Terminate** button twice.

9.2.3.8 How to Configure Startup and Shutdown Behavior for Integrated Application Servers

You can configure startup and shutdown behavior for integrated application server connections.

To configure the startup and shutdown behavior for an integrated application server:

1. If necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click the integrated application server connection and choose **Properties** to open the Application Server Properties dialog. For more information at any time, press F1 or click **Help** from within the dialog.

   If you are viewing the properties of the default integrated application server, you can only change settings on the Configuration, Shutdown and Launch Settings tabs in the dialog. Otherwise you can edit everything except the connection name.

9.2.3.9 How to Log In to the Integrated WebLogic Server Administration Console

The integrated application server is an implementation of Oracle WebLogic Server and as such you can connect to the server's Administration Console.

Note: To log in to the Administration Console, you must have the integrated application server running from JDeveloper, for example:

- By starting Integrated WebLogic Server from the Application Server Navigator.
- By running an application.

To launch and log in to the integrated application server Administration Console:

1. If necessary, open the Application Server Navigator by choosing **View > Application Server Navigator**.
2. Right-click **IntegratedWebLogicServer** and select **Launch Administrative Console**. A browser instance opens at the login page, which is `http://host:port/console`.

   For example, if the default configuration is used, the browser uses `http://localhost:7001/console`.
3. Log in using the username for the default domain and password you used when the integrated application server was launched for the first time.

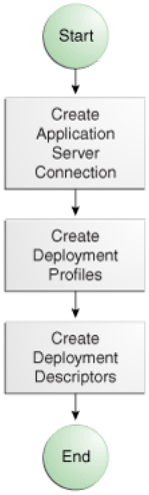
The integrated application server is an implementation of Oracle WebLogic Server, so for more information about the integrated application server Administration Console refer to the Administration Console Online Help, which is available from the WebLogic Server online documentation in your JDeveloper installation, or from the Administration Console.

9.3 Connecting and Deploying Java EE Applications to Application Servers

Before you deploy an application to a standalone application server, you must perform prerequisite tasks within JDeveloper to prepare the application for deployment.

Figure 9–2 show the process flow to prepare the application for deployment. After the application has been prepared and the application server has been prepared, you can proceed to deploy the application.

Figure 9–2 Preparing the Application for Deployment Flow Diagram



9.3.1 How to Create a Connection to the Target Application Server

You can deploy applications to the application server via JDeveloper application server connections.

Before you begin:

- Ensure that the application server is installed and started.
- If you are working behind a proxy server you need to configure JDeveloper to recognize the proxy server:
  1. Choose **Tools** > **Preferences** to open the Preferences dialog.
  2. Select **Use HTTP Proxy Server** and enter the hostname and port for the proxy server.

     Exceptions is populated with values from your machine. If you are deploying to a Oracle WebLogic Server configured to use SSL you should add `*.company_name.com` to the exception list.

To create a connection to an application server:

1. Launch the Application Server Connection wizard.

   You can:

   - In the Application Server Navigator, right-click **Application Servers** and choose **New Application Server Connection**.
   - In the New Gallery, expand **General**, select **Connections** and then **Application Server Connection**, and click **OK**.

     On the Usage page of the wizard ensure that **Standalone Server** is selected, and click **Next**.
   - In the Resource Palette, choose **New** > **New Connections** > **Application Server**.
2. In the Create AppServer Connection dialog Usage page, select **Standalone Server**.
3. In the Name and Type page, enter a connection name.
4. In the Connection Type dropdown list, choose:

- **WebLogic 10.3** to create a connection to Oracle WebLogic Server
- **JBoss 5.x** to create a connection to JBoss
- **Tomcat 6.x** to create a connection to Tomcat
- **WebSphere Server 7.x** to create a connection to IBM WebSphere Server

5. Click **Next**.
6. On the Authentication page, enter a user name and password for the administrative user authorized to access the application server.
7. Click **Next**.
8. On the Configuration page, enter the information for your server:

   For WebLogic:

   - The Oracle WebLogic host name is the name of the WebLogic Server instance containing the TCP/IP DNS where your application (`.jar`, `.war`, `.ear`) will be deployed.
   - In the **Port** field, enter a port number for the Oracle WebLogic Server instance on which your application (`.jar`, `.war`, `.ear`) will be deployed.

     If you don't specify a port, the port number defaults to 7001.
   - In the **SSL Port** field, enter an SSL port number for the Oracle WebLogic Server instance on which your application (`.jar`, `.war`, `.ear`) will be deployed.

     Specifying an SSL port is optional. It is required only if you want to ensure a secure connection for deployment.

     If you don't specify an SSL port, the port number defaults to 7002.
   - Select **Always Use SSL** to connect to the Oracle WebLogic Server instance using the SSL port.
   - Optionally enter a **WebLogic Domain** only if Oracle WebLogic Server is configured to distinguish non administrative server nodes by name.

   For JBoss:

   - Enter or browse to the location of the JBoss deploy directory, where your application files (`.jar`, `.war`, `.ear`) are.
   - If you are using JMX, Select **Enable JMX for this connection**. (optional).

     Note: JMX configuration is optional and is not required for connecting to the JBoss Application Server. JMX is only needed for deploying SOA applications.

     You must use the Oracle JMX RMI connector (`oracle-jboss-remoting.sar`) on the JBoss server; the standard JBOSS JMX connector (`jmx-remoting.sar`) does not work with JDeveloper.

   - In the **Host Name** field, enter host name of the target server. The default is the machine name.
   - In the **RMI Port** field, enter the port number of JBoss's RMI connector port. The default is 19000.

For Tomcat:

- In the **Webapps Directory** field enter or browse to the location of the webapps directory where you place the application `.war` files.

For WebSphere:

- In the **Host Name** field, enter the name of the WebSphere server containing the TCP/IP DNS where your Java EE applications (`.jar`, `.war`, `.ear`) are deployed. If no name is entered, the name defaults to `localhost`
- In the **SOAP Connector Port** field, enter the port number. The host name and port are used to connect to the server for deployment. The default SOAP connector port is `8879`
- In the **Server Name** field, enter the name assigned to the target application server for this connection.
- In the **Target Node** field, enter the name of the target node for this connection. A node is a grouping of Managed Servers. The default is `machineNode01`, where machine is the name of the machine the node resides on
- In the **Target Cell** field, enter the name of the target cell for this connection. A cell is a group of processes that host runtime components. The default is `machineNode01Cell`, where machine is the name of the machine the node resides on.
- In the **Wsadmin script location** field, enter, or browse to, the location of the wsadmin script file to be used to define the system login configuration for your IBM WebSphere application server connection. Note that you should not use the wsadmin files from the `ORACLE_HOME/oracle_common/common/bin` directory, which are not the correct version. The default location is `websphere-home/bin/wsadmin.sh` for Unix/Linux and `websphere-home/bin/wsadmin.bat` for Windows.

9. Click **Next**.
10. If you have chosen WebSphere, the JMX page appears. On the JMX page, enter the JMX information (optional):

Note: JMX configuration is optional and is not required for connecting to the WebSphere Application Server. JMX is only needed for deploying SOA applications.

- Select **Enable JMX for this connection** to enable JMX.
- In the **RMI Port** field, enter the port number of WebSphere's RMI connector port. The default is 2809.
- In the **WebSphere Runtime Jars Location** field, enter or browse to the location of the WebSphere runtime JARs.
- In the **WebSphere Properties Location (for secure MBEAN access)** field, enter or browse to the location of the file that contains the properties for the security configuration and the mbeans that are enabled. This field is optional.

11. Click **Next**.
12. On the Test page, click **Test Connection** to test the connection.

JDeveloper performs several types of connections tests. The JSR-88 test must pass for the application to be deployable. If the test fails, return to the previous pages of the wizard to fix the configuration.

13. Click **Finish**.

How to Launch Oracle WebLogic Server Administration Console

You can launch and connect to the Oracle WebLogic Server Administration Console from the Application Server Navigator.

> **Note:** To log in to the console, the server must be started.

1. In the Application Server Navigator, right-click the name of the connection to the Oracle WebLogic Server instance, and choose **Launch Admin Console**. A browser instance opens at the login page, which is `http://host:port/console`.

   For example, if the default configuration is used, the browser uses `http://localhost:7001/console`.

2. Log in using the username and password you used when creating the connection to the Oracle WebLogic Server instance. If you are launching the Administration Console for Integrated WebLogic Server, the default user is `weblogic` and password you entered when the default domain was created.

For more information about the WebLogic Server Administration Console, refer to the Administration Console Online Help, which is available from the WebLogic Server online documentation in your JDeveloper installation, or from the Administration Console.

9.3.2 How to Create and Edit Deployment Profiles

A deployment profile defines the way the application is packaged into the archive that will be deployed to the target environment. The deployment profile:

- Specifies the format and contents of the archive file that will be created
- Lists the source files, deployment descriptors, and other auxiliary files that will be packaged
- Describes the type and name of the archive file to be created
- Highlights dependency information, platform-specific instructions, and other information

9.3.2.1 About Deployment Profiles

Deployment to application servers uses deployment profiles which rely on project metadata for the default mappings. Default contributors to the profiles are based on project dependencies, although you can customize the deployment profiles to change them.

The rules governing dependencies are:

1. If project A depends on the build output of project B, then the build output of project B is merged into project A. If project A is a web application, this means the build outputs of project A and project B are both copied into the WEB-INF/classes of the resulting WAR.

   Merging implies that you can only have one copy of any particular URI, because it can only exist once within WEB-INF/classes.

2. If project A depends on the deployment profile of project B, for example a JAR profile, then the result of that deployment profile is included in the WEB-INF/lib of the resulting WAR.
3. All libraries marked Deploy by Default for a web project are deployed as a web application library (in the WEB-INF/lib of the WAR).
4. All libraries marked Deploy by Default for an EJB project are deployed as an application library (in the lib of the EAR).
5. If an EJB Project A depends on the build output of Project B, the build output (e.g. classes directory) of Project B is merged with the build output of Project A and deployed in the root directory of the EJB JAR.

Application level deployment profiles are:

- **EAR files:** Used to deploy the Java EE enterprise archive (EAR) file. The EAR file consists of the application's assembled WAR, EJB JAR and client JAR files.
- **MAR files:** Used for deploying metadata archive files for seeded customizations or base metadata in an MDS repository in the application server. For more information about MAR files, refer to the appropriate developer's guide for the Oracle® Fusion Middleware product you are using.

Project level Java EE deployment profiles are:

- **ADF Library JAR file:** Used for deploying ADF components as an application JAR file, which can be reused in ADF applications, or it can be used to build other ADF libraries.
- **Business Components archive file:** Creates a simple archive file for deploying ADF Business Components.
- **Business Components EJB Session Bean:** Creates a profile for deploying ADF Business Components as an EJB session bean.
- **Business Components Service Interface:** Creates a profile for deploying ADF Business Components as a service interface.
- **Client JAR files:** Used for deploying the standard Java EE client JAR file.
- **EJB JAR files:** Used to deploy the Java EE EJB module (EJB JAR). The EJB JAR contains the EJB components and the corresponding deployment descriptors.
- **Extension JAR file:** Creates a profile for deploying an extension as a JAR file.
- **JAR file:** Creates a simple JAR archive from a project.
- **OSGi bundle:** Creates an OSGi bundle that can be deployed to an OSGi container. You use this when you create extensions to JDeveloper.
- **RAR file:** Creates a profile for deploying a Java EE connector RAR file.
- **Shared Library JAR file:** Creates a profile for deploying a simple archive, which can be a JAR of ZIP file, to the file system or as a shared library to a remote server.
- **Taglib JAR file:** Creates a profile for deploying custom tag libraries to a JAR file.
- **WAR files:** Used to deploy the JAVA EE web module (WAR). The WAR consists of the web components (JSPs and servlets) and the corresponding deployment descriptors.

9.3.2.2 Creating Deployment Profiles

Deployment profiles can be created in various ways:

- Use the Deployment page of the Application Properties dialog, which can be opened from:
  - The Application menu on the JDeveloper toolbar.
  - The context menu of an application.
  - The dropdown list on the Application Navigator toolbar.
- Use the Deployment page of the Project Properties dialog, which can be opened from:
  - Select the project in the Application Navigator, and choose Project Properties from the Application menu on the JDeveloper toolbar.
  - The context menu of a project in the Application Navigator.
- Use one of the wizards from the General - Deployment Profiles category of the New Gallery (choose New from the File menu to open the New Gallery). The new deployment profile will be added to your project properties. To create application level profiles, invoke the New Gallery at application level. To create project level profiles, invoke the New Gallery at project level.
- If your project has a `web.xml` file, you can right-click it and choose **Create WAR Deployment Profile**.
- If your project is EJB 3.0, you can right-click the bean class and choose **Create EJB JAR Deployment Profile**.
- If your project has an `application.xml`, you can right-click it and choose **Create EAR Deployment Profile**.

To modify an existing deployment profile

- right-click the project in the Application Navigator and choose **Project Properties** then choose **Deployment** in the tree structure in the wizard, then select the deployment profile and choose **Edit**.
- right-click the application in the Application Navigator and choose **Application Properties** then choose **Deployment** in the tree structure in the wizard, then select the deployment profile and choose **Edit**.

To activate a deployment profile:

- For a project level deployment profile, right-click the project in the Application Navigator then choose **Deploy > deployment profile.**
- For an application deployment profile, right-click the application in the Application Navigator then choose **Deploy > deployment profile**. Alternatively,
  - Right-click the application in the Application Navigator then choose **Deploy > deployment profile**.
  - Choose **Deploy > deployment profile** from the context menu of an application.
  - Choose **Deploy > deployment profile** from the dropdown list on the Application Navigator toolbar.

The project and any projects on which it depends will be compiled and packaged.

You may find that the application you created already contains the deployment profile you need, for example if you create a web-based project you should already have a default WAR deployment profile which includes the dependent model projects it requires.

To create a deployment profile:

1. For an application level deployment profile, in the Application Navigator, right-click the application and choose **New**.

   For a project level deployment profile, in the Application Navigator, right-click the project that you want to deploy and choose **New**.

2. In the New Gallery, expand **General**, select **Deployment Profiles** and then choose the deployment profile type you want, and click **OK**.

   If you don't see **Deployment Profiles** in the **Categories** tree, click the **All Features** tab.

3. Choose the deployment profile type you want to create, and click **OK**. For example, for an EAR deployment profile:

   - Select **Application Assembly** and then in the **Java EE Modules** list, select all the project profiles that you want to include in the deployment, including any WAR profiles.
   - Select **Platform**, select the application server you are deploying to, and then select the target application connection from the **Target Connection** dropdown list.

4. In the Edit Deployment Profile Properties dialog, configure the profile by setting property values. For example, you may want to change the file groups that are included in the profile. When you have finished, click **OK**.

Deployment profiles are available from the Application Properties dialog, for application level deployment profiles, or from the Project Properties dialog, for project level deployment profiles, and you can edit them or delete them.

9.3.2.3 Viewing and Changing Deployment Profile Properties

After you have created a deployment profile, you can view and change its properties.

To edit or delete a deployment profile:

1. For an application level deployment profile, choose **Application** > **Application Properties** to open the Application Properties dialog.

   For a project level deployment profile, choose **Application** > **Project Properties** to open the Project Properties dialog.

2. Click **Deployment** in the left panel to open the Deployment page.

3. Choose the deployment profile you want to edit or delete, and click:

   - **Edit** to open the Edit Deployment Profile Properties dialog.
   - **Delete** to delete the deployment profile.

9.3.2.4 Configuring Deployment Profiles

Configuring is the process of assembling an archive file from its component files. Configuring is specified in the `File Groups` branch of deployment profile properties dialogs.

The `File Groups` branch consists of a list of file groups, each specifying some components. The packaged archive will be the union of all the file groups. The order of the file groups resolves name collisions: if two files have the same name, the one from the file group higher in the list is included, and the one from the lower file group is omitted.

A newly created deployment profile will include one or more predefined file groups. You can add, delete, and edit file groups.

File groups are defined by a set of contributors pruned by a set of filters. Contributors are source files, JAR files, and directories that are selected for inclusion. Filters are rules that are applied to the contributors or contributor's component subdirectories and files to identify the set and files that will be packaged. There are three kinds of file groups:

- The Packaging file group type allows you to select contributors, project directories and other directories and JAR files, and filters. The file group mechanism is flexible and transparent, and is appropriate for most projects.
- The Dependency analysis file group type allows you to select contributors that are project files and their dependencies. Profiles migrated from previous versions will contain a Dependency Analysis file group.
- The Libraries file group type allows you to select contributors that are project libraries. A libraries file group is created for WAR deployment profiles. Libraries files groups are useful in other projects that need to repackage existing JAR files.

9.3.3 How to Create and Edit Deployment Descriptors

Deployment dependencies between the components of an application are stated in their project's deployment profiles. In a project's deployment profile, name the profiles for the projects that are immediately upstream. When a deployment profile is activated for deployment, its dependencies will first be deployed.

Set deployment profile dependencies on the deployment profile's Profile Dependencies page. Only deployment profiles in the current workspace are listed and available for selection. Click the **Help** button for more information. The various profile dependencies you can select include:

- Profile-to-profile dependency
- Profile-to-JAR dependency
- Profile-to-WAR dependency
- Profile-to-RAR (Resource Archive) dependency

When deploying a profile contained in a project that has project-to-profile dependencies on other profiles, at deploy-time the profile incorporates the dependencies specified in the project. For example, if `Project1.jpr` contains `Servlet1.java` and depends on `ejb1.jar`, and `project2.jpr` contains `MySessionEJB` and `ejb1.jar`, then deploying the first project will result in an EAR file containing both `webapp1.war` and `ejb1.jar`.

When creating profile dependencies between JAR, WAR, and EJB JAR modules that share common JAR files, you can use the `META-INF/MANIFEST.MF Class-Path` attribute to link JAR files together at deploy-time. From the deployment profile properties JAR options page, select **Include Manifest File (META-INF/MANIFEST.MF)**. Doing so causes a single shared copy of any common JARs to be included in the EAR file.

Dependency projects can have dependencies of their own, but cyclical dependencies should be avoided. When JDeveloper encounters a circular dependency it will attempt to deploy anyway, but a warning will be displayed in the log window.

9.3.3.1 About Deployment Descriptors

Deployment descriptors are server configuration files that define the configuration of an application for deployment and that are deployed with the Java EE application as needed. The deployment descriptors that a project requires depend on the technologies the project uses and on the type of the target application server. Deployment descriptors are XML files that can be created and edited as source files, but for most descriptor types, JDeveloper provides dialogs or an overview editor that you can use to view and set properties. If you cannot edit these files declaratively, JDeveloper opens the XML file in the source editor for you to edit its contents.

In addition to the standard Java EE deployment descriptors (for example, `application.xml` and `web.xml`), you can also have deployment descriptors that are specific to your target application server. For example, if you are deploying to Oracle WebLogic Server, you can also have `weblogic.xml`, `weblogic-application.xml`, and `weblogic-ejb-jar.xml`.

The essential descriptors are created by the wizards that create deployment profiles. Add other descriptors only if you wish to override default behavior. In some cases descriptors will be created and included in archive files as they are deployed.

Deployment descriptors can also be created from the New Gallery. Deployment descriptors are placed in a META-INF subfolder of a project's Application Sources or WEB-INF subfolder of a project's Web Contents folders.

Each Java EE standard deployment descriptor is extended by a corresponding Oracle WebLogic Server-specific descriptor. Table 9–1 provides a description of these files and illustrates how they relate to one another.

Table 9–1 Deployment Descriptors

Java EE Standard Descriptors	Oracle WebLogic Server Proprietary Descriptors
`application-client.xml` Describes the EJB modules and other resources used by a Java EE application client deployed as an archive.	`weblogic-appclient.xml` The file format is defined in `weblogic-appclient.xsd`. For more information, see the chapter about client application deployment descriptor elements in *Oracle Fusion Middleware Programming Stand-alone Clients for Oracle WebLogic Server*.
`application.xml` Specifies the components of a Java EE application, such as EJB and web modules, and can specify additional configuration for the application as well. This descriptor must be included in the /META-INF directory of the application's EAR file.	`weblogic-application.xml` The file format is defined in `weblogic-application.xsd`. For more information, see *Oracle Fusion Middleware Programming XML for Oracle WebLogic Server*.

Table 9–1 (Cont.) Deployment Descriptors

Java EE Standard Descriptors	Oracle WebLogic Server Proprietary Descriptors

`ejb-jar.xml`
 	
Defines the specific structural characteristics and dependencies of the Enterprise JavaBeans within a JAR, and provides instructions for the EJB container about how the beans expect to interact with the container.
	`weblogic-ejb-jar.xml`
	
	The format of this file is defined in `weblogic-ejb-jar.xsd`.
	
	For more information, see *Oracle Fusion Middleware Programming Enterprise JavaBeans for Oracle WebLogic Server*.
	
	`persistence-configuration.xml`
	
	For EJB 3.0 modules. The format of this file is defined in `persistence-configuration.xsd`.
	
	For more information, see *Oracle Fusion Middleware Programming Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.
	
	`weblogic-cmp-rdbms-jar.xml`
	
	For EJB 2.1 modules. The format of this file is defined in `weblogic-rdbms20-persistence.xsd`.

`ra.xml` Contains information on implementation code, configuration properties and security settings for a resource adapter packaged within a RAR file.	`weblogic-ra.xml` The format of this file is defined in `weblogic-ra.xsd`. For more information, see t*Oracle Fusion Middleware Programming Resource Adapters for Oracle WebLogic Server*.
`web.xml` Specifies and configures a set of Java EE web components, including static pages, servlets, and JSP pages. It also specifies and configures other components, such as EJBs, that the web components might call. The web components might together form an independent web application and be deployed in a standalone WAR file.	`weblogic.xml` The format of this file is defined by `weblogic-web-app.xsd`. For more information, see *Oracle Fusion Middleware Programming XML for Oracle WebLogic Server*.

Table 9–1 (Cont.) Deployment Descriptors

Java EE Standard Descriptors	Oracle WebLogic Server Proprietary Descriptors

None.	`module-name-jdbc.xml`
	Defines data sources to be used in the deployed application.
	The format of this file is defined by `weblogic-jdbc.xsd`.
	For more information, see *Oracle Fusion Middleware Configuring and Managing JDBC for Oracle WebLogic Server*.
	`plan.xml`
	The format of this file is defined by `deployment-plan.xsd`.
	Contains a list of name/value pairs, and a description of the various deployment descriptors in an application. It allows administrators to override values in deployment descriptors.
	For more information, see *Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server*.
	`weblogic-diagnostics.xml`
	The format of this file is defined by `weblogic-diagnostics.xsd`.
	Used in the WebLogic Server Administration Console to create or modify diagnostic monitors in the diagnostic application module.
	For more information, see *Oracle Fusion Middleware Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.
	`weblogic-jms.xml`
	The format of this file is defined by `weblogic-jms.xsd`.
	Used to configure JMS drivers in the Oracle WebLogic Server.
	For more information, see *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.
	`weblogic-webservices.xml`
	The format of this file is defined by `weblogic-webservices.xsd`.
	For more information, see *Oracle Fusion Middleware WebLogic Web Services Reference for Oracle WebLogic Server*.

9.3.3.2 About Library Dependencies

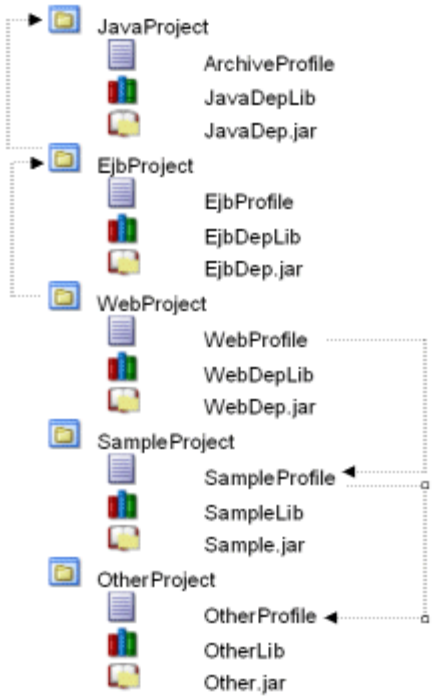
Dependent libraries are any library needed for a module to compile and run. In the Libraries and Classpath page of the Project Properties dialog for the project containing the library, dependent libraries are shown as available for export.

In an application, dependent libraries can be in the following projects:

- Projects of the current module's profile, that is the profile container.
- Projects that the profile container depends on.
- Projects associated with any profile dependency for this module's profile (recursively to its profile and project dependencies).

The example below illustrates project dependencies (arrows on the left) and profile dependencies (arrows on the right).

Figure 9–3 Dependencies Between Projects and Deployment Profiles



Project dependencies are recursive at deployment time, even though they are not at compile time, which is why the libraries from `JavaProject` are considered dependent libraries. `WebProfile`, which represents a web module, has the following dependent libraries:

- `EjbDepLib` (a library from a project dependency to `WebProject`)
- `EjbDep.jar` (a library jar from a project dependency to `WebProject`)
- `JavaDepLib` (a library from a recursive project dependency to `JavaProject`)
- `JavaDep.jar` (a library jar from a recursive project dependency to `JavaProject`)
- `SampleLib` (a library from a profile dependency)
- `Sample.jar` (a library jar from a profile dependency)
- `OtherLib` (a library from a recursive profile dependency)
- `Other.jar` (a library jar from a recursive profile dependency)

9.3.3.2.1 Resolved and Unresolved Libraries Dependent Libraries can either be resolved or unresolved. Dependent libraries are considered unresolved until they are included in an archive and placed on the classpath, thereby making the library content available to classes that need to reference it.

For example, a WAR profile resolves libraries by selecting those libraries in a library file group contributor where the target output directory is `WEB-INF\lib`. This ensures that the WAR archive created will include those libraries in the archive's `WEB-INF\lib` directory and thereby including the library content on the WAR archive's classpath.

When a library is not resolved by a deployment profile, this profile will expose the unresolved library in the application hierarchy so that it can be resolved at a higher level. Consider the situation where libraries contained in an EJB project remain

unresolved from the EJB profile's perspective. This information will be exposed so that an EAR profile can ensure that those libraries are resolved at the EAR level (in the EAR profile's library file group).

In the illustration above, WebProject has a project dependency to JavaProject, and JavaProject includes a library called JavaDepLib. You can define a web application which creates a WAR deployment profile on WebProject. You can then resolve JavaDepLib in the web module by ensuring that this library is selected in the `WEB-INF\lib` library file group of the WAR deployment profile.

9.3.3.2.2 Manifest Entries for Libraries When libraries are included in an EAR archive in a directory other than the standard `\lib` or `APP-INF\lib`, JDeveloper automatically inserts the required manifest entries into the modules that refer to those libraries.

9.3.3.3 Creating Deployment Descriptors

JDeveloper automatically creates many of the required deployment descriptors for you. If they are not present, or if you need to create additional descriptors, you can explicitly create them.

Before you begin:

Check to see whether JDeveloper has already generated deployment descriptors.

To create a deployment descriptor:

1. In the Application Navigator, right-click the project for which you want to create a descriptor and choose **New**.
2. In the New Gallery, expand **General**, select **Deployment Descriptors** and then a descriptor type, and click **OK**.

   If you can't find the item you want, make sure that you chose the correct project, and then choose the **All Features** tab or use the **Search** field to find the descriptor. If the item is not enabled, check to make sure that the project does not already have a descriptor of that type. A project is allowed only one instance of a descriptor.

   JDeveloper starts the Create Deployment Descriptor wizard and then opens the file in the overview or source editor, depending on the type of deployment descriptor you choose.

   Note: For EAR files, do not create more than one of any type of deployment descriptor per application or workspace. Only the application resources descriptors or descriptors generated at the EAR level will be used by the runtime. If multiple projects in an application have the same deployment descriptor, the one belonging to the launched project will supersede the others. This restriction applies to `application.xml`, `weblogic-jdbc.xml`, `jazn-data.xml`, and `weblogic.xml`.

   The best place to create an application-level descriptor is in the Descriptors node of the Application Resources panel in the Application Navigator. This ensures that the application is created with the correct descriptors.

To inspect or change deployment descriptor properties:

1. In the Application Navigator, select the deployment descriptor.

2. Right-click and choose **Open**.

   The file will open in an overview editor specific to that descriptor type, or in an XML editor window.

9.3.3.4 Viewing or Modifying Deployment Descriptor Properties

After you have created a deployment descriptor, you can change its properties by using JDeveloper dialogs or by editing the file in the source editor. The deployment descriptor is an XML file (for example, `application.xml`) typically located under the Application Sources node.

To view or change deployment descriptor properties:

1. In the Application Navigator or in the Application Resources panel, double-click the deployment descriptor.
2. In the overview editor, select either the **Overview** tab or the **Source** tab, and configure the descriptor by setting property values.

   If the overview editor is not available, JDeveloper opens the file in the source editor.

9.3.4 How to Configure Global Deployment Preferences

You can set global deployment options in the Deployment page of the Preferences dialog.

To configure the deployment preferences:

1. Choose **Tools > Preferences** from the main menu.
2. Select the **Deployment** node. Configure the deployment options as required. For more information, click **Help**.
3. Click **OK**.

> **Note:** Set application-specific and project-specific deployment profile options via the application properties or project properties. The Application Properties and Project Properties dialogs are available from the Application menu

9.3.5 How to Pass Options to Target Connections When Deploying

When deploying in JDeveloper, you can directly access the target application server connection in order to pass command line options. For example, you can specify the client JAR which contains the necessary stubs and skeletons on the client side to support RMI-IIOP deployment. These options would overwrite or bypass the server's default settings.

To pass options to target application server connections when deploying:

1. If not already done, create the appropriate deployment profile.
2. In the Application Navigator, right-click the project, then choose **Properties**.
3. Select Deployment in the panel on the left of the Project Properties dialog.
4. Select the deployment profile you want to edit, and click **Edit**.

5. Open the page which corresponds to the target connection type for which you want to pass command options.
6. Edit the page, or click **Restore Default** to revert to the default settings for the target server.

   For instructions click **Help**.
7. Click OK when you are finished editing the deployment profile properties.

9.3.6 How to Configure Applications for Deployment

This section describes the tasks you may have to perform for the application to deploy successfully to an application server.

9.3.6.1 How to Configure an Application for Deployment to Oracle WebLogic Server

When you create applications in JDeveloper You can deploy the packaged application to Oracle WebLogic Server through an application server connection. A packaged application will contain a deployment profile that names the files to be deployed, describes their organization, and specifies the target server. The target Oracle WebLogic Server instance must be installed locally or mapped to a network drive.

To configure an application for deployment to Oracle WebLogic Server:

1. Set up any JDBC data sources you need on the server. For more information, see Section 9.3.6.4, "Setting Up JDBC Data Sources on Oracle WebLogic Server."
2. For clients that access EJBs on Oracle WebLogic Server, the following code is required in the client.

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
  "weblogic.jndi.WLInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "system");
env.put(Context.SECURITY_CREDENTIALS, "welcome1");
env.put(Context.PROVIDER_URL, "t3://localhost:7001");
```

3. When deploying to Oracle WebLogic Server, if you use the same EJB in two or more different applications, the second deployment will usually cause a JNDI name collision. Therefore, you must rename the JNDI name of the EJB for the second and any subsequent deployments:
   - Right-click `weblogic-ejb-jar.xml`, and choose **Open**.
   - Under Enterprise Java Beans, select the relevant ModuleBM bean. The EJB tab is displayed on the right.
   - In the EJB tab, change the **JNDI Name** so that it is different from any other JNDI Name in `weblogic-ejb-jar.xml` and any other EJBs that are already deployed to Oracle WebLogic Server.
   - Deploy the application accessing the EJB to Oracle WebLogic Server. During deployment, the IDE automatically fills in `weblogic.xml` with appropriate EJB references.

9.3.6.2 How to Configure a Client Application for Deployment

A Java EE Client module is packaged as a client JAR file which contains one or more Java application components and a client deployment descriptor file named `application-client.xml`. After you have created the deployment profile and the deployment descriptor file, you can deploy the client JAR to the application server.

To package a client application for deployment:

1. Create a Client JAR File deployment profile for your project.

   A profile may have already been created for your project. If you wish to deploy to multiple targets, create a separate profile for each.

2. Create the `application-client.xml` deployment descriptor file, if not already present in your project.

   Normally, this file is created with the application client.

9.3.6.3 How to Configure an Applet for Deployment

A standalone applet is packaged as a web archive (WAR) file which contains the applet, the Applet HTML file, as well as the standard Java EE web deployment descriptor, web.xml and possibly target-specific deployment descriptors, as well. After you have created the deployment profile and the appropriate deployment descriptor files, you can deploy the application to an application server, or as an archive file.

To configure a web application for deployment:

1. Create a WAR file deployment profile for your project.

   A profile may have already been created for your project. If you wish to deploy to multiple targets, create a separate profile for each.

2. Add a `web.xml` deployment descriptor to your project, if it is not already present.

   Normally, this file is created with the WAR file deployment profile.

> **Note:** If you encounter problems when deploying a Swing applet (JApplet), for example, the error `Class not found` is displayed, this may indicate that JDeveloper cannot locate the Swing libraries. Your clients may need to use Sun's Java SE browser plugin or bundle the Swing libraries for JVMs version 1.1 with your applet. Deployed applet files must reside in a separate location from any other web application files you have deployed.

9.3.6.4 Setting Up JDBC Data Sources on Oracle WebLogic Server

To avoid passwords being present in plain text in deployed files, JDeveloper uses password indirection, which means that passwords for the data sources must be set on the server before the application will run correctly.

You do this using global data sources, which are set up in the Oracle WebLogic Server Administration Console using the **Data Sources** link under **JDBC**.

JDeveloper ensures that your web application `web.xml`, or EJB application `ejb-jar.xml`, contains the necessary `<resource-ref>` entry to identify an application resource name. The name is `jdbc/connection-nameDS`, where connection-name is the name of the application resources connection.

The application looks up this data source using the application-specific resource JNDI namespace of `java:comp/env/jdbc/connection-nameDS`, and it finds this resource because web.xml contains the `<resource-ref>` entry for `jdbc/connection-nameDS`.

An important control for the files that are generated is the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** field on the WebLogic page of the Application Properties dialog.

When the **Auto Generate** field is selected, JDeveloper does the following:

- Generates an `application-name-jdbc.xml` file for each connection in the application resources, and sets the indirect password attribute

  ```
  <jdbc-driver-params>
  <use-password-indirection>true</use-password-indirection>
  </jdbc-driver-params>
  ```

  Upon deployment, JDeveloper determines the JDBC connection password from the username in `application-name-jdbc.xml`, and populates the JDBC connection password using an Mbean.

- `weblogic-application.xml` is updated to add each `application-name-jdbc.xml` as a module.
- `web.xml` (if it exists) has a resource reference to each JDBC JNDI name.

How to Create a Global Data Source on Oracle WebLogic Server

You create a global data source on Oracle WebLogic Server Administration Console.

To set up a global data source:

1. Login to the Oracle WebLogic Server Administration Console. For more information, see Section 9.3.1, "How to Create a Connection to the Target Application Server."
2. Click on the **Data Sources** link under **JDBC**.
3. On the Summary of JDBC Data Sources page, click **New**.
4. In the Create a New JDBC Data Source page, enter details of the data source.

   The name can be anything.

   The JNDI name must be of the form `jdbc/connection-nameDS`. For example, if the application has a connection name `connection1`, the JNDI name is `jdbc/connection1DS`.
5. Ensure that the database type is `Oracle` and that the driver is `Oracle's Driver (Thin) for Service Connections;Version 9.0.1,9.2.0,10,11`.
6. Click **Next** twice to navigate to the Create a New JDBC Data Source page, where you enter the connection details.

   The database name is the `Oracle SID`.

   The host name is the name of the machine the database is on.

   The default port is `1521`.
7. Enter the user name and password, for example `hr/hr`.
8. Click **Next** and click **Test Configuration**.
9. Click **Next** to navigate to the Select Targets page, where you select a target for this data source. If you fail to select a target, the data source is created but not deployed.
10. Click **Finish**.

Deploying to an EAR File to Run on Oracle WebLogic Server

To deploy an application to an EAR file to run on Oracle WebLogic Server, you can:

- Select the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** field, and set up passwords using application level credential mapping.
- Alternatively, you can deselect the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** field and set up passwords by creating a global data source on Oracle WebLogic Server.

If you are deploying using ojdeploy:

- You can use the `-nodatasources` switch, in which case you can set up passwords on Oracle WebLogic Server by either:
  - Creating a global data source.
  - Manually creating application level data sources.
- If you do not use the `-nodatasources` switch, you can only set up passwords using application level credential mapping.

9.3.6.5 Preparing an Application for Deployment to a Third Party Server

There may be specific tasks that you have to perform so that your application will run on a third party server.

Deploying to Tomcat:

- Stop and restart the Tomcat server after deployment.
- Make sure that you have the `tools.jar` library in the Tomcat classpath, located in `jdeveloper_install/jdk/lib`. This file must be the same version of the JDK being used to run Tomcat. Otherwise, you may encounter problems when running applications in Tomcat.
- The recommended deployment for web applications is `tomcat_install/webapps/`subdirectory. Set this option in the General page of the WAR File deployment profile.
- The system administrator of the Tomcat application server must assign a context path to your application in the `conf/server.xml` file:

  ```
  <DefaultContext crossContext="true"/>
  ```

  See Tomcat system administration documentation for more information.
- You may get the following error message when running a JSP application deployed to Tomcat:

  ```
  Only one of the two parameters ... or ... should be defined.
  ```

  Because Tomcat does not release tags after pooling, subsequent uses of the same tag with incompatible attributes defined will cause this error.

  To avoid the error, you must disable tag pooling in Tomcat:

  1. Open the file `tomcat_home/conf/web.xml` in a text editor.
  2. Find the following element:

     ```
     <init-param>
       <param-name>enablePooling</param-name>
       <param-value>true</param-value>
     </init-param>
     ```

     Change the value of `<param-value>` to false

Deploying to WebSphere

WebSphere deployment on Windows does not work when the directory containing the JDeveloper generated EAR contains spaces.

9.3.7 How to Use Deployment Plans

You can use deployment plans to override deployment values. One reason you might want to do this is to change settings so that an application that has finished testing can be run in a production environment without having to change the deployment profiles.

When an EAR, WAR, or EJB JAR archive configured to use a deployment plan is deployed, both the archive and the deployment plan are sent to the application server. You can use multiple deployment plans in an application.

For more information, see the section about deployment plans in the chapter about configuring applications for production deployment in Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server.

The following is an example of a deployment plan for an EAR called `application.ear`. Note that the `module-name` element must contain the name of the deployment profile that it is associated with.

Example 9–1 Example of a Deployment Plan

```
<deployment-plan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/technology/weblogic/10.3/deployment-plan
http://www.oracle.com/technology/weblogic/10.3/deployment-plan/1.0/deployment-plan
.xsd"
xmlns="http://www.oracle.com/technology/weblogic/10.3/deployment-plan">
  <application-name>DeployPlan</application-name>
  <variable-definition>
    <variable>
      <name>SessionDescriptor_timeoutSecs</name>
      <value>888</value>
    </variable>
    <variable>
      <name>SessionDescriptor_invalidationIntervalSecs</name>
      <value>888</value>
    </variable>
    <variable>
      <name>SessionDescriptor_cookieMaxAgeSecs</name>
      <value>888</value>
    </variable>
  </variable-definition>
  <module-override>
    <module-name>application.ear</module-name>
    <module-type>ear</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-application</root-element>
      <uri>META-INF/weblogic-application.xml</uri>
      <variable-assignment>
        <name>SessionDescriptor_timeoutSecs</name>
        <xpath>/weblogic-application/session-descriptor/timeout-secs</xpath>
      </variable-assignment>
      <variable-assignment>
        <name>SessionDescriptor_invalidationIntervalSecs</name>

<xpath>/weblogic-application/session-descriptor/invalidation-interval-secs</xpath>
      </variable-assignment>
```

```
      <variable-assignment>
        <name>SessionDescriptor_cookieMaxAgeSecs</name>

<xpath>/weblogic-application/session-descriptor/cookie-max-age-secs</xpath>
      </variable-assignment>
    </module-descriptor>
  </module-override>
</deployment-plan>
```

When an EAR, WAR, or EJB JAR archive configured to use a deployment plan is deployed, both the archive and the deployment plan are sent to the application server. You can use multiple deployment plans in an application.

9.3.7.1 How to Create and Use Deployment Plans

You can create a deployment plan from the New Gallery and edit it in the XML editor.

Once created, a deployment plan can be associated with an EAR, WAR, or EJB JAR archive.

Alternatively, you can generate a deployment plan in Oracle WebLogic Server, then use it in JDeveloper.

To create a deployment plan:

1. In the Application Navigator, select the project for which you want to create a deployment plan.
2. Choose **File > New** to open the New Gallery.
3. In the Categories tree, expand **General** and select **Deployment Descriptors**. In the Items list, select **WebLogic Deployment Descriptor** and click OK.
4. In the Select Descriptor page of the Create WebLogic Deployment Descriptor wizard, choose plan.xml.

   If this is the first deployment descriptor you are creating in the application, you can choose Finish to create a deployment plan with the default name of plan.xml.

   If you already have a deployment plan called plan.xml in the application, Navigate to the Select Name page and enter a new name for the deployment plan, then click Finish. The deployment plan will be created and added to the project, and it will be opened in an XML editor window.
5. Open the Deployment Profile Properties of the EAR, WAR or EJB JAR.
6. Enter the path to the deployment plan in the **Deployment Plan** field.

9.3.7.2 How to Generate Deployment Plans

Deployment plans enable you to export an application's configuration for deployment to multiple WebLogic Server environments.

You can create a deployment plan from scratch in JDeveloper.

Alternatively, you can generate a deployment plan which you can then add to your application in JDeveloper and edit it to suit your purposes, described in this topic. There are two ways to do this:

- Deploy the application to a Oracle WebLogic Server, make changes to the application using the Administration Console, and save the resulting deployment plan. You can then copy the deployment plan back into your source in JDeveloper, and if necessary you can modify it. For more information, see the section about deployment plans in the chapter about configuring applications for production

deployment in Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server.

- Use the weblogic.PlanGenerator command-line tool to generate a deployment plan for an application that uses an EAR. For more information, see the reference chapter about weblogic.PlanGenerator command line tool in Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server.

To generate a deployment plan using WebLogic Server Administration Console:

1. Deploy the application to Oracle WebLogic Server.
2. Open the WebLogic Server Administration Console.

   The WebLogic Server Administration Console automatically generates (or updates) a valid deployment plan for an application when you interactively change deployment properties for an application that you have installed to the domain. You can use the generated deployment plan to configure the application in subsequent deployments, or you can generate new versions of the deployment plan by repeatedly editing and saving deployment properties.

To use the weblogic.PlanGenerator command-line tool to generate a deployment plan:

1. From the command line, navigate to `install/wlserver_10.3/server/bin/` and run either `setWLSEnv.sh` or `setWLSEnv.cmd` script, to add the WebLogic Server classes to the CLASSPATH environment variable on your machine, and ensure that the correct JDK binaries are available in your PATH.
2. From the command line, navigate to the location of the EAR file, and run `java weblogic.PlanGenerator -plan plan.xml application-name.ear -all`.

   The switch `-all` specifies that the deployment plan is generated containing elements for all possible attributes in your EAR file. If you remove this switch, the generated deployment plan will only contain elements for the existing attributes in your descriptor files.

9.4 Deploying Java Applications

JDeveloper supports deployment of applications containing a variety of technologies to a variety of application servers. This section provides instructions for deploying an application to an executable JAR file on your file system. If you wish to deploy an application containing Java EE technologies, or you wish to deploy to the integrated application server, Oracle WebLogic Server, or another supported application server, be sure to verify that you have performed the necessary configuration and preparation steps as outlined in Section 9.3, "Connecting and Deploying Java EE Applications to Application Servers."

9.4.1 Deploying to a Java JAR

Applications can be deployed indirectly by choosing an archive file as the deployment target. The archive file can subsequently be installed on a target Java EE application server

JDeveloper has various deployment modes for different applications. However, you may want to quickly and simply deploy your application as a JAR file to your file system.

Note: Before deploying an executable JAR file you must first create a deployment profile.

To deploy a simple archive in JDeveloper:

1. Select and right-click the project in the Application Navigator.
2. Choose **Deploy deployment profile**, where deployment profile is the deployment profile that you created earlier.
3. In the Deployment Action page of the Deploy dialog, choose **Deploy to JAR file**, and finish the wizard.

You can make your simple archive or Java EE Client Module into an executable JAR file that you can launch with the **java** command.

Note: Before deploying an executable JAR file you must first create a deployment profile.

To deploy an executable JAR file:

1. Right-click the project in the Application Navigator and choose **Project Properties**.
2. Select the name of the profile in the Deployment section of the Project Properties dialog and click **Edit**.
3. Click **JAR Options** in the tree.
4. Select **Include Manifest File (META-INF/MANIFEST.MF)**.
5. In the Main Class field, enter the fully qualified name of the application class that is to be invoked.
6. Click **OK**.
7. Launch the executable JAR file from the command line

   ```
   java -jar myapp.jar
   ```
   where `myapp` represents your JAR file name.

9.4.2 Deploying to an OSGi Bundle

Applications can be deployed as OSGi bundles which can then be deployed to an OSGi container.

JDeveloper has various deployment modes for different applications. However, you may want to quickly and simply deploy your application as a JAR file to your file system.

Note: Before deploying an OSGi bundle you must first create a deployment profile. For more information, see Section 9.3.2, "How to Create and Edit Deployment Profiles."

To deploy an OSGi bundle in JDeveloper:

1. Select and right-click the project in the Application Navigator.
2. Choose **Deploy > deployment profile**, where deployment profile is the OSGi bundle deployment profile that you created earlier.

3. In the Deployment Action page of the Deploy dialog, choose **Deploy to OSGi bundle**, and finish the wizard.

9.5 Deploying Java EE Applications

You can use JDeveloper to deploy Java EE applications directly to the standalone application server or create an archive file and use other tools to deploy to the application server.

9.5.1 How to Deploy to the Application Server from JDeveloper

The Java EE Enterprise Archive (EAR) deployment profile provides you with centralized control over the process of application assembly. This assembling task involves selecting which already-configured Java EE deployment profiles to include with the EAR file. You can mix and match any combination of configured WAR, EJB JAR, and/or client JAR profiles in projects within the same workspace. When you deploy an application to an application server connection, JDeveloper assembles a minimal EAR file which includes the profile combinations and deploys it with the EAR file to the target application server.

To deploy an application as a Java EE Enterprise Archive (EAR File):

1. Create an EAR File deployment profile.
2. Create a connection to the target application server.
3. Right-click the project in Application Navigator and choose **Deploy > deployment profile**.
4. In the Deployment Action page of the Deploy dialog, choose one of the deployment options:
   - Deploy to application server connection to package the web module as an EAR file, and deploy it to the application server connection you select or create on the Select Server page of the Deploy dialog.
   - Deploy to EAR file to package the web module as an EAR file and save to the location specified in the EAR deployment profile.

To reopen the EAR deployment profile later to make changes, right-click the application in the Application Navigator toolbar and choose **Application Properties**, then select the name of the profile in the Deployment section of the Application Properties dialog and click **Edit**.

- If you have an existing EAR file, you can use the JDeveloper EAR import facility to import the EAR into any project.
- JAR and WAR files to be included in an EAR file must be created before the EAR file is deployed. For the included application's deployment profiles, choose Deploy to JAR file or Deploy to WAR file in the Deploy dialog to create these subordinate archives.
- The EAR file does not contain passwords so if, for example, you are creating an EAR file to run on Oracle WebLogic Server, you must set up a data source on the server.

9.5.2 How to Deploy a RAR File

Stored in a Resource Adapter Archive (RAR) file, a resource adapter may be deployed on any Java EE server, much like the EAR file of a Java EE application. A RAR file may be contained in an EAR file or it may exist as a separate file

To deploy a resource adapter archive in JDeveloper:

1. Create a deployment profile.

   Note: To reopen a project deployment profile later to make changes, right-click the project in the Application Navigator and choose **Project Properties**, then select the name of the profile in the Deployment section of the Project Properties dialog and click **Edit**.

2. Right-click the project in the Application Navigator, then choose **Deploy > deployment profile**.
3. In the Deployment Action page of the Deploy dialog, choose **Deploy to RAR file**.

9.5.3 How to Add a Resource Adapter Archive (RAR) to the EAR

The EAR profile supports Resource Adapter Archive files (RAR or .rar) in a JDeveloper project. A RAR file is typically provided by an Enterprise Intelligence Server (EIS) vendor, similar to a JDBC driver. Java EE developers may need to package a RAR file into their EAR file if their Java EE application makes use of the EIS services supported by the RAR. JDeveloper does not directly support RAR file creation, but RAR files can be assembled using the File Groups feature of a JAR file deployment profile.

The ra.xml file is the deployment descriptor for the RAR file for the J2EE Connector Architecture (JCA). For more information, see

`http://www.oracle.com/technetwork/java/javaee/tech/entapps-138775.html`

To add a RAR to an EAR deployment profile:

1. In JDeveloper, add an existing RAR file to a project.
2. Create an EAR deployment profile in the same project as the RAR file.
3. Right-click the project in the Application Navigator and choose **Project Properties**.
4. Select the name of the profile in the Deployment section of the Project Properties dialog and click **Edit**.
5. Click the **Application Assembly** node to display all the Java EE modules (WAR and EJB JAR) currently available and saved in your project.
6. Select the checkbox next to the RAR (.rar) file that you want to assemble and package with the EAR file.
7. Click **OK**.
8. Deploy the Java EE EAR.

At deploy-time, the EAR file's application.xml contains a <connector> element which is automatically added to the RAR file.

9.5.4 How to Deploy a Metadata Archive (MAR) File

Metadata Archive (MAR) profiles are application level deployment profiles which are used to package seeded customizations or place base metadata in the MDS repository. In a MAR profile, selections can only be done at the package level, not at the file level.

There are two uses for a MAR profile

- The first use is to create a MAR profile. Once you have created it you can include it in an application's EAR for deployment.
- The second use is to export MAR contents to MDS repository configured for a deployed application in a remote server. This procedure is for applying ADF Library customizations changes to an application that has already been deployed to a remote application server. It is not for the initial packaging of customizations into a MAR that will eventually be a part of an EAR.

To deploy a MAR profile in an EAR:

1. Create a MAR deployment profile.
2. Choose **Application > Deploy > deployment-profile**. In the Deploy profile dialog, select Deploy to MAR. For more information at any time, press F1 or click `Help` from within the Deploy profile dialog.

   The Deploy to MAR option creates a metadata archive file, which is a convenient option that can be used to verify the MAR contents. The created metadata file should have the same MAR contents as the Export to Deployed Application option.
3. To include a MAR profile in a new EAR, in the Application Properties dialog that is displayed when you finish creating the MAR profile, click **New** to create an application level EAR deployment profile.

   Alternatively, to add the MAR profile to an existing EAR profile, open the EAR profile.
4. In the Edit EAR Deployment Profile Properties dialog, go to the Application Assembly page and ensure that the MAR profile is listed under Java EE Modules.
5. Click **OK**.
6. Deploy the Java EE EAR.

To export MAR contents to MDS repository and deploy it to a deployed application in a remote server:

1. Create a MAR deployment profile.
2. Choose **Application > Deploy > deployment-profile**. In the Deploy profile dialog, select Deploy to MAR. For more information at any time, press F1 or click **Help** from within the Deploy profile dialog.

   The Deploy to MAR option creates a metadata archive file, which is a convenient option that can be used to verify the MAR contents. The created metadata file should have the same MAR contents as the Export to Deployed Application option.
3. To export the MAR contents to the MAR profile, choose **Application > Deploy > deployment-profile**. In the Deploy profile dialog, choose **Export to a Deployed Application**. For more information at any time, press F1 or click **Help** from within the Deploy profile dialog.

4. Continue through the Deploy profile dialog. You can choose the server to deploy to, and then the deployed application in that server. You can also choose to use a sandbox instance before committing the deployment.

9.5.5 How to Deploy an Applet as a WAR File

You can deploy web application components including applets as a WAR or EAR file to the target application server.

To deploy an applet as a WAR file:

1. If not already done, configure the applet for deployment.
2. If not already done, create an application server connection.
3. In the Navigator, right-click the project and choose **Deploy > deployment profile.**
4. Deploy to application server connection to create the archive type specified in the deployment profile, and deploy it to the application server connection you select or create on the Select Server page of the Deploy dialog.
   - **Deploy to application server connection** to create the archive type specified in the deployment profile, and deploy it to the application server connection you select or create on the Select Server page of the Deploy dialog.
   - **Deploy to EAR file** to deploy the project and any of its dependencies (specified in the deployment profile) to an EAR. JDeveloper puts the EAR file in the default directory specified in the deployment profile.
   - **Deploy to WAR file** to deploy the project to a WAR. JDeveloper puts the WAR file in the default directory specified in the deployment profile.

   Note: The deployed applet files must reside in a separate location from any other web application files you have deployed.

You can test the deployed web application by running it in a browser. For more information, see Section 9.7, "Testing the Application and Verifying Deployment."

If you encounter problems when deploying a Swing applet (JApplet), for example, the error "Class not found" is displayed, this may indicate that JDeveloper cannot locate the Swing libraries. You may need to force your clients to use Sun's Java SE browser plugin or bundle the Swing libraries for JVMs version 1.1 with your applet.

9.5.6 How to Deploy a Shared Library Archive

Shared Java EE libraries provides an easy way to share one or more different types of Java EE modules among multiple Enterprise Applications. You can deploy shared libraries as JAR files to the application server.

To deploy create and deploy a shared library archive:

1. Create a shared library deployment profile.
2. Add the libraries to the profile in the Edit JAR Deployment Profile Properties dialog. Choose File Groups, and click **New** to open the Create File Group dialog, where you define a new file group.
3. Create a connection to the target application server.

4. Right-click the project in Application Navigator and choose **Deploy > shared library deployment profile**.
5. On the Deployment Action page of the Deploy shared library dialog, choose **Deploy to a Weblogic Application Server** and click **Next**.
6. On the Select Server Page, choose the application server connection and select **Deploy as a shared Library**. Click **Finish**.

9.5.7 How to Deploy to a Managed Server That Is Down

For successful deployment, the Administration Server for the WebLogic Server domain has to be up as it is handling the deployment process. When you deploy to a server that is down, the deployment log window messages indicate that the server is currently down but the application will be installed when it is brought back up. The log messages will be similar to:

```
[02:27:21 PM] ----  Deployment started.  ----
   [02:27:21 PM] Target platform is  (Weblogic 10.3).
   [02:27:23 PM] Retrieving existing application information
   [02:27:23 PM] Running dependency analysis...
   [02:27:23 PM] Building...
   [02:27:26 PM] Deploying 2 profiles...
   [02:27:26 PM] Wrote Web Application Module to
/scratch/.../jdev/mywork/Application1/Project1/deploy/webapp1.war
   [02:27:26 PM] Wrote Enterprise Application Module to
/scratch/.../jdev/mywork/Application1/application1.ear
   [02:27:26 PM] Deploying Application...
   [02:27:27 PM] [Deployer:149195]Operation 'deploy' on application 'application1'
has been deferred since   'Server-2' is unavailable
   [02:27:27 PM] [Deployer:149034]An exception occurred for task
[Deployer:149026]deploy application application1   on Server-2.: .
   [02:27:27 PM] Application Deployed Successfully.
   [02:27:27 PM] Elapsed time for deployment:  5 seconds
 [02:27:27 PM] ----  Deployment finished.
```

One situation that can occur is that deployment appears to succeed, but as the server is brought back up the deployment cannot successfully terminate, for example, because some validation that is part of the deployment process was not performed, or because a library that needs to be present for deployment to be successful is missing. In these cases, when the server is brought back up and deployment resumes, it fails.

You can only deploy an application once to a server that is down. If you attempt to redeploy the same application to the same down server a second time, an error is displayed.

9.6 Post-Deployment Configuration

After you have deployed your application to Oracle WebLogic Server, you can migrate it from one Oracle WebLogic Server to another.

You may need to perform some of the same steps you did for a first time deployment.

In general, to migrate an application to another application server, you would:

- Configure the target application server with the correct database or URL connection information.
- Migrate security information, for example JDBC data sources, from the source to the target.

- Deploy the application to the new server.

9.7 Testing the Application and Verifying Deployment

After you deploy the application, you can test it from Oracle WebLogic Server.

The deployment log window displays the context root URLs for any Web applications deployed. You can access a deployed web application by entering the application URL in a browser. The URL of the deployed web application appears in the deployment log window, for example:

```
[03:08:20 PM] The following URL context root(s) were defined and can be used as a
starting point to test your application:
[03:08:20 PM] http://12.345.678.912:7101/Project1
[03:08:21 PM] Elapsed time for deployment: 7 seconds
[03:08:21 PM] ---- Deployment finished. ----
```

You can copy the URL and paste it into a browser to test the deployed web application.

Depending on your browser proxy settings, you may need to specify the full domain name of the host machine. If the servlet engine and the browser used to view a deployed application are on the same machine, you may use localhost for the host name.

9.8 Deploying from the Command Line

Applications or modules can be deployed from JDeveloper without starting the JDeveloper IDE.

Before deploying from the command line, you need to run JDeveloper at least once to create a deployment profile for either the application by choosing the Deployment page of the Application Properties dialog or the Project Properties dialog, both of which are available from the Application menu.

Deployment profiles are stored as part of either the application or project properties.

9.8.1 How to Deploy from the Command Line

Applications or modules can be deployed from JDeveloper without actually starting the JDeveloper IDE itself. Deploying from the command line, using OJDeploy, is especially useful where you need to deploy existing projects or applications using a batch file or other script.

OJDeploy can run a deployment locally in-process, or submit to a background server, OJServer, using the `-ojserver` option.

9.8.1.1 Command Usage

Example 9–2 Using ojdeploy to Deploy from the Command Line

```
ojdeploy -profile <name> -workspace <jws> [ -project <name> ] [ <options> ]
ojdeploy -buildfile <ojbuild.xml> [ <options> ]
ojdeploy -buildfileschema
```

Table 9–2 Arguments That Can be Used With ojdeploy

Argument	Description
`profile`	Name of the profile to deploy.
`workspace`	Full path to the JDeveloper application file (`.jws`)
`project`	Name of the JDeveloper project within the `.jws` where the deployment profile can be found. If omitted, the profile is assumed to be in the application.
`buildfile`	Full path to a build file for batch deploy.
`buildfileschema`	Print XML Schema for the build file.

Note: Deployment profiles can be classified into two broad categories, those that are defined at the application (workspace) level, and those defined at the project level. To deploy an application profile, OJDeploy takes the application location, and the name of the profile. To deploy a project profile it takes an additional `-project` argument.

Table 9–3 Options Available to Use with ojdeploy

Option	Description
`-address`	The listen address for OJServer. Defaults to `localhost:2010`. The default parts of the address may be omitted, for example, `-address :2001` or `-address fasup-pls01`.
`-basedir`	Interpret path for application relative to a base directory. The built-in macro `${base.dir}` captures the value of `-basedir`.
`-clean`	Deletes all files from the project output directory before compiling. Deployment will stop for that profile, if a file or directory could not be deleted.
`-define`	Allows for additional macros to be defined on the command-line. This can also be done in an XML build file using the `<variable>` element. Macros and options defined on the command line supplement, or override, those found in the `<defaults>` section of a build file if one is being used.
-failonwarning	Stop deployment on warnings.
-forcerewrite	Ensures output file is rewritten even if the contents have not changed in this run of OJDeploy.
-nocompile	Prevents the build system from being invoked. This is useful if an application or project just needs to be packaged, and not compiled at this time.
-nodatasources	For Java EE applications, this prevents the `weblogic-jdbc.xml` file from being updated with connection information found in the JDeveloper IDE. This option is ignored for non-Java EE applications.
-nodependents	Dependent profiles are not deployed.
-ojserver	Runs the deployment job on an OJServer. All paths referenced by the other options should be accessible on the server.

Table 9–3 (Cont.) Options Available to Use with ojdeploy

Option	Description
-outputfile	Specifies an alternate location for any JAR files created from the profile. The default is within a `/deploy` directory inside the project or application. If this parameter does not specify a file extension, the extension is determined from the original file name in the deployment profile.
-statuslogfile	Full path to an output file for status summary. No macros allowed. The `-statuslogfile` option creates an XML file that stores a list of all the profiles processed and the status of each. A summary section at the end can be checked to quickly determine the exit status for the entire script.
-stdout, -stderr	Lets these streams be redirected to a file for each profile and project. You can use macros in the name or path of the files.
-timeout	Specify the number of seconds after which deployment of a single profile should be aborted.
-updatewebxmlejbrefs	Update EJB references in `web.xml`.

Table 9–4 Macros Available to Use with ojdeploy

Macro	Description
`workspace.name`	Name of the application (without the `.jws` extension).
`workspace.dir`	Directory of the application `.jws` file.
`project.name`	Name of the project (without the `.jpr` extension).
`project.dir`	Directory of the `project.jpr` file.
`profile.name`	Name of the profile being deployed.
`deploy.dir`	Default deploy directory for the profile.
`base.dir`	Override the current OJDeploy directory using this parameter. You can also override the current OJDeploy directory using the basedir attribute in the build script.

Note: The project.name and project.dir macros are only available when a project-level profile is being deployed.

Examples

Deploy a project-level profile:

- `ojdeploy -profile webapp1 -workspace /usr/jdoe/Application1/Application1.jws -project Project1`
- `ojdeploy -profile webapp1 -workspace Application1/Application1.jws -basedir /usr/jdoe -project Project1`

Deploy an application-level profile:

- `ojdeploy -profile earprofile1 -workspace /usr/jdoe/Application1/Application1.jws`

Deploy all profiles from all projects of an application:

- ojdeploy -workspace /usr/jdoe/Application1/Application1.jws -project * -profile *

Build in batch mode from a ojbuild file:

- ojdeploy -buildfile /usr/jdoe/ojbuild.xml

Build using ojbuild file, pass into, or override default variables in, the build file:

- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -define myhome=/usr/jdoe,mytmp=/tmp
- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -basedir /usr/jdoe

Build using ojbuild file, set or override parameters in the default section:

- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -nocompile
- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -outputfile '${workspace.dir}/${profile.name}.jar'
- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -define mydir=/tmp -outputfile '${mydir}/${workspace.name}-${profile.name}'

More examples:

- ojdeploy -workspace Application1/Application1.jws,Application2/Application2.jws -basedir /home/jdoe -profile app*
- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -define outdir=/tmp,rel=11.1.1 -outputfile '${outdir}/built/${workspace.name}/${rel}/${profile.name}.jar'
- ojdeploy -workspace Application1/Application1.jws -basedir /home/jdoe -nocompile -outputfile '${base.dir}/${workspace.name}-${profile.name}'
- ojdeploy -workspace /usr/jdoe/Application1.jws -project * -profile * -stdout /home/jdoe/stdout/${project.name}.log
- ojdeploy -buildfile /usr/jdoe/ojbuild.xml -ojserver

9.8.1.2 How to Override Without Editing a Build Script

To pass in macro values or override the ones defined in a build script, use the `-define` option to supply a new value:

```
ojdeploy -buildfile /home/jdoe/ojbuild.xml -define "mycustomdir=/tmp"
```

This will add the `mycustomdir` variable to the <defaults> section of the build script, or replace it if it already is defined with the value '/tmp'.

To pass in parameter values or override the ones defined in a build script, use the appropriate parameter option:

```
ojdeploy -buildfile /home/jdoe/ojbuild.xml -nocompile -nodatasources
```

This will add the `-nocompile` and `-nodatasources` parameters to the default section of the build file.

9.8.2 How to Deploy Multiple Profiles from the Command Line

Command-line deployment supports deployment of multiple applications in a single invocation. If more complex control is required, OJDeploy can take an XML build script and process it, running all deploy tasks found in it. Macros and wild cards can

be used both in command-line and batch mode. Macros can be strung together or nested.

Each profile to be deployed is qualified by an application and a project. In addition each profile's output can be directed to a different output file/location. Further to this, the calling script assumes no knowledge of the projects within a workspace, only deploying all or a subset of them matching a criteria. The command-line syntax for specifying such inputs and criteria can quickly become cumbersome and inflexible.

A build file can be passed to OJDeploy. The build file will contain multiple `<deploy>` tasks, along with a shared `<defaults>` section which allows for setting up an environment. Each deploy-task specifies the type of deployment (the set mentioned before) and customizes any defaults as required. Each task also allows wild cards as applicable within parameter arguments that apply to the scope of that task. A pre-processor will parse the build file and pass it to OJDeploy, expanding wild cards and substituting variables as necessary.

The build file approach has the following advantages over the command-line syntax:

- It lets more parameters be added to OJDeploy without forcing the implementor for that parameter to be aware of a batch-build concept.
- It keeps the command-line syntax simple, for the degenerate case.
- It allows parameters to be dynamically evaluated based on the current context, and access to a predefined list of pre-processor macros. For example, OutputFile location may be specified as `c:\temp\${profile.name}` where the macro `${profile.name}` is added automatically.

A sample build file is shown below. To invoke all of these deploy actions, the command-line would be `ojdeploy ojdeploy-build.xml`. The file is processed from top to bottom.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<ojdeploy-build basedir="/usr/jdoe/">
  <!-- Defines default parameters for all deploy tasks.
       Also defines some variables strictly for use within this file
       in macros
  -->
  <defaults>
    <parameter name="profile" value="*"/>
    <parameter name="nocompile" />
    <-- define a macro -->
    <variable name="customdir" value="/var/projects/fin/"/>
  </defaults>
  <!-- Select all .jws files in location ${customdir} called absoluteFile1.jws,
absoluteFile2.jws.
       Open all projects.
       Deploy profiles p1, p2, p3 in each project, in each workspace.
  -->
  <deploy>
    <parameter name="workspace"
value="${customdir}/absoluteFile1.jws,${customdir}/absoluteFile2.jws"/>
    <parameter name="project" value="*"/>
    <-- Override default profile parameter -->
    <parameter name="profile" value="p1,p2,p3"/>
  </deploy>
  <!--
    Open relativeFile1.jws in the base directory
    Open all projects.
    Deploy all profiles (default for "profile" parameter is "*")
  -->
```

```
  <deploy>
    <parameter name="workspace" value="relativeFile1.jws"/>
    <parameter name="project" value="*"/>
  </deploy>
  <!--
    Open relativeFile2.jws in base directory.
    Open all Projects
    Deploy profiles matching the patter "web*"
  -->
  <deploy>
    <parameter name="workspace" value="relativeFile2.jws"/>
    <parameter name="project" value="*"/>
    <parameter name="profile" value="web*"/>
  </deploy>
</ojdeploy-build>
```

9.8.2.1 How to Use Wildcard Samples

Project and profile names can be specified as "*" or "name*" or "name1,name2,name3,..." or any combination of these. Workspace names need to be enumerated, so "*" is not allowed in workspace names, but workspace names can be specified as "workspace1" or "workspace1,workspace2,workspace3".

For example:

- adf* (Profile)
- View* (Project)
- *Controller (All Controller Projects)

An example of using wild cards with an application:

```
<ojdeploy-build basedir= "/home/jdoe" >
  <deploy>
    <parameter name= "workspace" value= "Application1.jws,Application2.jws" />
    <!-- above pattern gets /home/jdoe/Application1.jws and
/home/jdoe/Application2.jws -->
    . . .
  </deploy>
/ojdeploy-build>
```

9.8.2.2 How to Use Built-in Macros

The following built-in macros can be used in build files:

Table 9–5 Macros Available to Use With Build Files

Macro name	Description
`${workspace.name}`	Name of application, excluding the vT file extension.
`${workspace.dir}`	Directory containing the application (`.jws`) file.
`${project.name}`	Name of project, excluding the `.jpr` file extension.
`${project.dir}`	Directory containing the project (`.jpr`) file.
`${profile.name}`	Defined name of the profile.
`${deploy.dir}`	Default deploy directory, usually `${project.dir}/deploy` for project-level profiles or `${workspace.dir}/deploy` for workspace-level profiles.
`${base.dir}`	Value of the `-basedir` parameter, or the current directory.

9.8.2.3 How to Create a Log File for Batch Deployment

You can use the parameter `-statuslogfile c` to provide the absolute path to a log file. The path should not contain macros.

The log file contains a list of the deployment tasks processed and the status from each task in XML format. The status will be either `SUCCESS` or `FAILED` and includes an exitcode attribute. Possible values for exitcode are:

- 0 - Success
- 1 - Fatal error (NPE, OutOfMemory, etc.)
- 2 - JDeveloper configuration error (missing extensions, etc.)
- 4 - Deployment Error (compilation, deployment exception, etc.) All exit codes are bitwise OR-ed.

A combined status is available in a summary section at the end of each log.

Example 9–3 Example of Batch Deployment Log Output

```
<?xml version="1.0"?>
 <ojdeploy-log>
  <deploy-task>
   <target>
    <profile>webapp1</profile>
    <workspace>/scratch/jdoe/jdev/mywork/Application3/Application3.jws</workspace>
    <project>Project1.jpr</project>
   </target>
   <exception msg="**** One or more compilation errors prevented deployment from
continuing.">
    oracle.jdeveloper.deploy.DeployException: **** One or more compilation errors
prevented deployment from continuing.
    at
oracle.jdevimpl.deploy.common.ModulePackagerImpl.compileDependents(ModulePackagerI
mpl.java:143)
    at
oracle.jdeveloper.deploy.common.ModulePackager.compile(ModulePackager.java:65)
    at
oracle.jdeveloper.deploy.common.ModulePackager.prepareImpl(ModulePackager.java:52)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.WrappedDeployer.prepareImpl(WrappedDeployer.java:32)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.WrappedDeployer.prepareImpl(WrappedDeployer.java:32)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.DeploymentManagerImpl.deploy(DeploymentManagerImpl.java
:411)
    at
oracle.jdevimpl.deploy.fwk.DeploymentManagerImpl$1.run(DeploymentManagerImpl.java:
281)

   </exception>
   <status exitcode="4">FAILED</status>
  </deploy-task>
  <deploy-task>
  <target>
```

```
   <profile>archive1</profile>
   <workspace>/scratch/jdoe/jdev/mywork/Application3/Application3.jws</workspace>
   <project>Project1.jpr</project>
  </target>
  <exception msg="**** One or more compilation errors prevented deployment from
continuing.">
    oracle.jdeveloper.deploy.DeployException: **** One or more compilation errors
prevented deployment from continuing.
    at
oracle.jdevimpl.deploy.common.ModulePackagerImpl.compileDependents(ModulePackagerI
mpl.java:143)
    at
oracle.jdeveloper.deploy.common.ModulePackager.compile(ModulePackager.java:65)
    at
oracle.jdeveloper.deploy.common.ModulePackager.prepareImpl(ModulePackager.java:52)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.WrappedDeployer.prepareImpl(WrappedDeployer.java:32)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.WrappedDeployer.prepareImpl(WrappedDeployer.java:32)
    at
oracle.jdeveloper.deploy.common.AbstractDeployer.prepare(AbstractDeployer.java:69)
    at
oracle.jdevimpl.deploy.fwk.DeploymentManagerImpl.deploy(DeploymentManagerImpl.java
:411)
   at
oracle.jdevimpl.deploy.fwk.DeploymentManagerImpl$1.run(DeploymentManagerImpl.java:
281)

  </exception>
  <status exitcode="4">FAILED</status>
 </deploy-task>
 <deploy-task>
  <target>
   <profile>ejb1</profile>
   <workspace>/scratch/jdoe/jdev/mywork/Application3/Application3.jws</workspace>
   <project>Project3.jpr</project>
  </target>
 <status exitcode="0">SUCCESS</status>
 </deploy-task>
 <summary>
 <start-time>2007-12-19 12:10:42 PST</start-time>
 <end-time>2007-12-19 12:10:45 PST</end-time>
 <total-tasks>3</total-tasks>
 <failures>2</failures>
 <status exitcode="4">FAILED</status>
 </summary>
 </ojdeploy-log>
```

9.8.3 How to Deploy from the Command Line Using Ant

JDeveloper deployment is built around deployment profiles. A common implementation is an ArchiveProfile that describes the structure of a JAR archive. Deployment profiles can be created as part of a project or workspace. A command-line tool, OJDeploy, is available to allow deployment of ArchiveProfile(s) without invoking the JDeveloper IDE.

Command line deployment requires a JDeveloper installation, but this installation is invoked in 'headless mode', not displaying the JDeveloper IDE, loading all extensions defined for headless mode. This form of deployment can read JDeveloper applications and projects and their meta-data. Ant scripts to invoke command line deployment need to be created manually. The resulting deployed archive depends on version of JDeveloper used, and which extensions are enabled when command line deployment is invoked

9.8.3.1 How to Generate an Ant Build Script

To make it easier to create an Ant build script for command line deployment, an Ant script can be generated from JDeveloper.

Example 9–4 Structure of the Ant Build Script

```
<project name="Project1" default="all" basedir=".">
 <property file="build1.properties"/>
 <target name="init">
 <tstamp/>
 <mkdir dir="${output.dir}"/>
 </target>
 <target name="all" description="Build the project"
depends="compile,copy,deploy"/>
 <target name="clean" description="Clean the project">
   . . .
 </target> <target name="compile" description="Compile Java source files"
depends="init">
   . . .
 </target>
 <target name="copy" description="Copy files to output directory" depends="init">
   . . .
 </target>
 <!--- This is the additional part generated for deployment ---->

 <target name="deploy" description="Deploy JDeveloper profiles"
   depends="init,compile">
 <taskdef name="ojdeploy"
   classname="oracle.jdeveloper.deploy.ant.OJDeployAntTask"
   uri="oraclelib:OJDeployAntTask"
   classpath="${oracle.jdeveloper.ant.library}"/>
 <ora:ojdeploy xmlns:ora="oraclelib:OJDeployAntTask"
   executable="${oracle.jdeveloper.ojdeploy.path}"
   ora:buildscript="${oracle.jdeveloper.deploy.dir}ojdeploy-build.xml"
   ora:statuslog="${oracle.jdeveloper.deploy.dir}ojdeploy-statuslog.xml">
 <ora:deploy>
 <ora:parameter name="workspace"
   value="${oracle.jdeveloper.workspace.path}"/>
 <ora:parameter name="project"
   value="${oracle.jdeveloper.project.name}"/>
 <ora:parameter name="profile"
   value="${oracle.jdeveloper.deploy.profile.name}"/>
 <ora:parameter name="nocompile" value="true"/>
 <ora:parameter name="outputfile"
   value="${oracle.jdeveloper.deploy.outputfile}"/>
 </ora:deploy>
 </ora:ojdeploy>
 </target>
 <!-------------- end of deployment ---->
 </project>
```

9.8.3.2 About The build.xml File

The `build.properties` file, which is generated along with `build.xml`, defines the additional variables needed for command line deployment:.

Example 9–5 Example of the build.xml File

```
#Fri Feb 15 10:45:22 PST 2008
   #Sun Feb 24 18:47:36 PST 2008
   javac.nowarn=off
   javac.debug=on
   build.compiler=oracle.ojc.ant.taskdefs.OjcAdapter
   output.dir=classes
   oracle.home=../../oracle/
   javac.deprecation=off
   oracle.jdeveloper.ant.library=/scratch/jdoe/oracle/jdev//lib/ant-jdeveloper.jar
   oracle.jdeveloper.deploy.dir=/scratch/jdoe/Application7/Project1/deploy/
   oracle.jdeveloper.ojdeploy.path=/scratch/jdoe/oracle/jdev//bin/ojdeploy
   oracle.jdeveloper.workspace.path=/scratch/jdoe/Application7/Application7.jws
   oracle.jdeveloper.project.name=Project1
   oracle.jdeveloper.deploy.profile.name=*
   oracle.jdeveloper.deploy.outputfile=/scratch/jdoe/Application
```

9.8.3.3 About The build.properties File

The Ant build script can be run outside of JDeveloper by simply changing to the directory containing `build.xml` and running Ant. It can also be run from within JDeveloper, by right-clicking on the `build.xml` node in the Application Navigator and selecting the "all" or the "deploy" targets.

Example 9–6 Example of the build;.properties File

```
Buildfile: /scratch/jdoe/Application7/Project1/build1.xml

init:

compile:
deploy:
   [ora:ojdeploy]
   [ora:ojdeploy] Oracle JDeveloper Deploy 11.1.1.0.0
   [ora:ojdeploy] Copyright (c) 2008, Oracle. All rights reserved.
   [ora:ojdeploy]
   [ora:ojdeploy] ----build file----
   [ora:ojdeploy] <?xml version = '1.0' standalone = 'yes'?>
   [ora:ojdeploy] <ojdeploy-build>
   [ora:ojdeploy] <deploy>
   [ora:ojdeploy] <parameter name="workspace"
value="/scratch/jdoe/Application7/Application7.jws"/>
   [ora:ojdeploy] <parameter name="project" value="Project1"/>
   [ora:ojdeploy] <parameter name="profile" value="*"/>
   [ora:ojdeploy] <parameter name="nocompile" value="true"/>
   [ora:ojdeploy] <parameter name="outputfile"
value="/scratch/jdoe/Application7/Project1/deploy/${profile.name}"/>
   [ora:ojdeploy] </deploy>
   [ora:ojdeploy] <defaults>
   [ora:ojdeploy] <parameter name="buildfile"
value="/scratch/jdoe/Application7/Project1/deploy/ojdeploy-build.xml"/>
   [ora:ojdeploy] <parameter name="statuslogfile"
value="/scratch/jdoe/Application7/Project1/deploy/ojdeploy-statuslog.xml"/>
   [ora:ojdeploy] </defaults>
   [ora:ojdeploy] </ojdeploy-build>
```

```
   [ora:ojdeploy] ------------------
   [ora:ojdeploy] ---- Deployment started. ---- Feb 24, 2008 6:49:51 PM
   [ora:ojdeploy] Target platform is (WebLogic 10.3).
   [ora:ojdeploy] Running dependency analysis...
   [ora:ojdeploy] Wrote JAR file to
/scratch/jdoe/Application7/Project1/deploy/archive1.jar
   [ora:ojdeploy] Elapsed time for deployment: less than one second
   [ora:ojdeploy] ---- Deployment finished. ---- Feb 24, 2008 6:49:51 PM
   [ora:ojdeploy] ---- Deployment started. ---- Feb 24, 2008 6:49:51 PM
   [ora:ojdeploy] Target platform is (Java Enterprise Edition 1.5).
   [ora:ojdeploy] Running dependency analysis...
   [ora:ojdeploy] Wrote WAR file to
/scratch/jdoe/Application7/Project1/deploy/WindowMobile.war
   [ora:ojdeploy] Elapsed time for deployment: less than one second
   [ora:ojdeploy] ---- Deployment finished. ---- Feb 24, 2008 6:49:52 PM
   [ora:ojdeploy] Status summary written to
/scratch/jdoe/Application7/Project1/deploy/ojdeploy-statuslog.xml

BUILD SUCCESSFUL
   Total time: 19 seconds
```

Note: By default, the command line deployment task has the nocompile option enabled as the task has dependency on the compile task. If this dependency is removed then the nocompile option can be removed.

It is a best practice to generate an `.ear` file from JDeveloper for the application. The `.ear` file will be generated with all the right class dependencies required to deploy it. Deploying with Ant by referring to an application directly without generating an `.ear` file may require that dependencies for the classes and jars files must be resolved manually.

9.9 Deploying Using Java Web Start

JDeveloper supports the creation of the XML-based JNLP (Java Network Launching Protocol) definition upon which the Java Web Start technology is based. Java Web Start allows you to deploy Java applications so that they can be launched from an internet browser. Java Web Start lets you maintain Java client applications and applets on the web server, which users download and run on their client machines.With the Create Java Web Start-Enabled wizard in JDeveloper, you can set up applications and applets to be maintained on the web server, but downloaded and run on client machines.

The process of developing a Java Web Start application can be summarized as:

1. Develop the Java application.
2. Simulate the user's experience of running the application with Java Web Start within the JDeveloper IDE.
3. Use the JDeveloper Java EE Web deployment process to move the production application to the web server.

Note: To launch applications and applets with Java Web Start in JDeveloper, you must download and install the Java Web Start software. Users of your application or applet will also be required to install the software on their machines.

For more information on Java Web Start and to download the Java Web Start software, see http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html.

9.9.1 Purpose of the Java Web Start Technology

Although Java Web Start and applets may appear to be similar technologies, there are several differences:

- Unlike the applet approach to deploying web-centric Java applications, Java Web Start does not rely on the web browser to perform the downloading of the application JAR files. Instead, Java Web Start downloads the application resources after the Java Web Start JNLP descriptor is downloaded through the web browser. The JNLP descriptor causes Java Web Start to launch and perform the actual downloading.
- While users of the application may experience the applet identically in Java Web Start, they are not tied to the web browser as they would be with applets. Once the application is running, the web browser can be closed, and the application continues to run in Java Web Start.

With the Java Web Start software installed once on the client machine, the application user can run applications and applets simply by clicking on a web page link. If the application is not present on their computer, Java Web Start automatically downloads all necessary files from the web server where the application libraries reside. It then caches the files on the client computer so the application is always ready to be relaunched anytime either from an icon on your desktop or from the browser link. The most current version of the application is always presented to the user since Java Web Start performs updates as needed.

9.9.1.1 Files Generated by the Create Java Web Start-Enabled Wizard

Application users can use Java Web Start to run applications and applets on client machines, while you maintain the application on the web server. To support Java Web Start and web server downloading, the Create Java Web Start-Enabled wizard generates these files:

- The Java Network Launching Protocol (JNLP) definition required by Java Web Start to download and launch the application. The `.jnlp` file describes the archive files and whether this instance includes an applet or an application.
- An HTML file that contains the URL to initiate the downloading from the web server to the client. Although HMTL file creation is optional, it is highly recommended unless you intend to create the file manually.

Users can use Java Web Start to run applications and applets on client machines, while you maintain the application on the web server. To support Java Web Start and web server downloading, the Create Java Web Start-Enabled wizard generates these files:

9.9.1.2 Role of the Web Server in JDeveloper

JDeveloper provides an Integrated WebLogic Server web server. You can use it to simulate the process of deploying the Web Application Archive and downloading for use with Java Web Start. JDeveloper follows the J2SE deployment profile conventions for archiving components that run on the client machine (simple archive) and components that are deployed to the web server (Web Application Archive).

How to complete the Java Web Start setup:

1. Create a simple Java Archive (.jar) file that contains the application source files to be downloaded and run on the client machine.
2. Launch the Create Java Web Start-Enabled wizard in JDeveloper to create the HTML and JNLP files that will enable the application or applet to be downloaded and run on the client machine.
3. .Create a Web Application Archive (`.war`) file which you deploy to the web server. It will contain the contents of the `public_html` directory in your JDeveloper mywork folder, including the JAR, HTML and JNLP files.

> **Note:** You will not be required to deploy the application to use the JDeveloper-embedded web server. JDeveloper provides a default web.xml definition to locate the contents of the `public_html` directory in your JDeveloper `mywork` folder.

Once you have set up the web server, you can launch the Java Web Start software in JDeveloper using the generated `.html` file. Java Web Start relies on your web browser to download the components identified by the `.jnlp` file. Another definition in the `.jnlp` file determines whether it will run as an application or a secure applet. Once you have launched Java Web Start and the downloading is complete, you can close your web browser and continue to run the application or applet.

9.9.2 How to Create a Java Web Start File

A Java Network Launching Protocol definition file, `application-name.jnlp`, is automatically created when you use the Create Java Web Start-Enabled wizard to create Java clients to download and run Java applications and applets on client machines. However, if you want to control the contents of the `application-name.jnlp`, you can manually create your own file to use.

> **Note:** If this item appears grayed out, this indicates that there is an `application-name.jnlp` file already created in the project. You can have only one of each deployment descriptor type per project.

To manually create a Java Web Start (.jnlp) file:

1. In the Categories tree, expand General and select Deployment Descriptors. In the Items list, double-click Java Web Start (JNLP) Files.
2. If the category or item is not found, make sure the correct project is selected, and choose All Technologies in the Filter By dropdown list.
3. Click OK.
4. The newly created file opens in the Code Editor. Edit this file to add the configuration settings as appropriate.

For more information on Java Web Start, see http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html.

9.9.3 How to Create an ADF Swing Web Archive for Java Web Start

You can use the JDeveloper Java EE web deployment process to set up the web server before downloading and running the application using Java Web Start.

Once the application resides on the web server, it becomes very easy to maintain. Java Web Start takes care of identifying and downloading application updates each time the user runs the application.

How to create the ADF Swing web application archive for Java Web Start:

1. Before creating your archive files to ensure that you archive the latest source files, create both the following:
   - The Business Component project.
   - The ADF Swing project.
2. Run the ADF Swing Java Web Start dialog. In the Application Navigator, select the application or project in which you are working.

   Choose **File > New** to open the New Gallery.

   In the Categories tree, expand **Client Tier** and select **ADF Swing**. In the Items list, select **Java Web Start (JNLP) Files for ADF Swing**, and click OK.

   The dialog generates JNLP files for use with Java Web Start, an ANT build file ctbuild.xml, and a deployment profile in the project properties.
3. If for security reasons the password used for the Java keystore defined on the machine differs from the password used to protect the key and sign the code, then you must modify the ctbuild.xml ANT build file to reference the specified password:
   - Open the ctbuild.xml file and add the property for the key password below the other signing properties:

     ```
     <!--properties related to signing-->
     <property name="alias" value="ADFADF SwingTrust"/>
     <property name="storepass" value="welcome"/>
     <property name="keypass" value="myPassword"/>
     ```
   - Change the sign target from the following:

     ```
     <target name="sign" depends="jar">
        <signjar jar="${mt.jar.name}" alias="${alias}"
     storepass="${storepass}"/>
        <signjar jar="${ct.jar.name}" alias="${alias}"
     storepass="${storepass}"/>
     </target>
     ```

     to:

     ```
     <target name="sign" depends="jar">
        <signjar jar="${mt.jar.name}" alias="${alias}" keypass="${keypass}"
     storepass="${storepass}"/>
        <signjar jar="${ct.jar.name}" alias="${alias}" keypass="${keypass}"
     storepass="${storepass}"/>
     </target>
     ```
4. To create the client side archive files, right-click `ctbuild.xml` in the ADF Swing project and choose:

   Build Target > sign to require authentication of the archive, this will sign the contained JAR files and is a required step.

The build file should generate two signed archive files in your project's `public_html` directory: `client.jar` and `mymt.zip`. These archives are referenced by the generated deployment profile in the project's properties.

5. (Optional) If you want to edit the web.xml deployment descriptor, right-click the `web.xml` file in the Application Navigator and choose **Open**.
6. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Application Navigator and choose Project Properties, then select the name of the profile in the Deployment section of the Project Properties dialog and click Edit.

When you are ready to deploy the resulting WAR and EAR files to the target application server, make sure to create an application server connection.

9.9.4 How to Create a Java Client Web Archive for Java Web Start

You can use the JDeveloper Java EE web deployment process to set up the server before downloading and running the application using Java Web Start.

Once the application resides on the web server, it becomes very easy to maintain. Java Web Start takes care of identifying and downloading application updates each time the user runs the application.

To create Java client applications for deployment to the web server:

1. Create a simple JAR archive of your Java client application.
2. Create a Web Start JNLP Definition for Java Clients to generate the JNLP file and HTML file for use with Java Web Start.
3. In the Application Navigator, select the project in which you want to create the WAR deployment profile.
4. Choose **File > New** to open the New Gallery.
5. In the Categories tree, expand **General** and select **Deployment Profiles**. In the Items list, double-click **WAR File**.
6. If the category or item is not found, make sure the correct project is selected, and choose **All Technologies** in the **Filter By** dropdown list. Enter the name of the new deployment profile then click **OK**.
7. The WAR Deployment Profile Properties panel displays. Configure the settings for each page as appropriate. Click **OK** when you have finished defining the properties.

   The newly created web.xml deployment descriptor appear in the Application Navigator below the specified project.
8. Deploy the Java Client Web Archive for Java Web Start.
9. (Optional) If you want to edit the web.xml deployment descriptor, right-click the `web.xml` file in the Application Navigator and choose **Open**.
10. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Application Navigator and choose Project Properties, then select the name of the profile in the **Deployment** section of the Project Properties dialog and click **Edit**.

When you are ready to deploy the resulting WAR or EAR to the target application server, make sure to create an application server connection.

Note: The web module is deployed to the target deployment directory.

Make sure that the web application deployment descriptor is located inside the Web Application Archive (WAR) file `WEB-INF/web.xml`.

9.9.5 How to Create a Java Web Start JNLP Definition for Java Clients

You use the Create Java Web Start-Enabled wizard to create the XML-based JNLP (Java Network Launching Protocol) definition file that the Java Web Start software uses to download and run Java applications and applets on client machines.

Note: You must download and install the Java Web Start software to launch applications and applets with Java Web Start in JDeveloper. Users of your application or applet will also be required to install the software on their machines. See

`http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html`

The application or applet must be delivered in a set of JAR files and all application resources, such as images, configuration files and native libraries, must be included in the JAR files. The resources must be looked up using the ClassLoader `getResource` or another method. Java Web Start only transfers JAR files from the web server to the client. for additional information, see `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html`

The wizard adds a JNLP file and (optionally) an HTML file to your project. Java Web Start will use these generated files to determine what application source to download from the web server:

- The Java Network Launching Protocol (JNLP) definition is required by Java Web Start to download and launch the application. The `.jnlp` file describes the archive files and whether this instance includes an applet or an application.
- An HTML file. Although HMTL file creation is optional, it is highly recommended unless you intend to create the file manually. The HTML file contains the URL to initiate the downloading from the web server to the client.

Before you launch the Create Java Web Start-Enabled wizard to create the JNLP and HTML files, you must create a simple archive (JAR) file for it. You must also know in which class the main function can be found, as you will be asked to specify this.

To create the JNLP definition for your application or applet:

1. In the Navigator, select the project in which you want to generate a JNLP definition. Choose **File > New** to open the New Gallery.
2. In the **Categories** tree, expand **Client Tier** and select **Swing/AWT**. In the **Items** list, double-click **Java Web Start (JNLP) Files** to open the Create Java Web Start-Enabled wizard.

   Click **Next** in the Welcome page.
3. In the Application Information page, enter the file name, the name and location of the JAR file that you created, and the class that you want to use to run your application.

4. For detailed help in using the Create Java Web Start-Enabled wizard, press F1 or click **Help** from within the wizard.
5. Check **Create Homepage** to create the optional HTML file. Click **Next** after specifying the desired options.
6. In the Web Start page, specify information to document the JNLP file. Complete the wizard and click **Finish**.

You can also use a JSP file or servlet with Java Web Start; however, you will have to manually configure the file and change the content type. Here is an example JNLP with `contentType = application/x-java-jnlp-file`, specified in the first line:

Example 9–7 JNLP File

```
<%@ page contentType="application/x-java-jnlp-file" %>
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://192.168.1.102:8888" href="jnlpfile.jnlp">
<information>
<title>Test</title>
<vendor>Oracle</vendor>
<homepage href="Test.html"/>
<description>Encryption Tool</description>
<icon href="images/frontpage.gif"/>
<offline-allowed/>
</information>
<security><all-permissions/></security>
<resources>
<j2se version="1.3"/>
<jar href="/apps/archive1.jar" main="true" download="eager" />
</resources>
<application-desc main-class="oracle.Ide">
</application-desc>
</jnlp>
```

9.9.6 How to Deploy an ADF Swing Web Application Archive for Java Web Start

You can deploy the ADF Swing Web Archive to a server. Install the Java Web Start software on your machine. For more information on Java Web Start, see http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html.

To deploy the ADF Swing web application archive for Java Web Start:

1. If not already done, create a signed ADF Swing Web Archive for Java Web Start.
2. If not already done, create an application server connection to the target application server.
3. Right-click the ADF Swing project and choose Deploy to automatically generate the WAR file and deploy the application components. You must choose a connection to the desired web server.
4. (Optional) If you want to edit the web.xml deployment descriptor, right-click the `web.xml` file in the Application Navigator and choose **Open**.
5. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Application Navigator and choose **Project Properties**, then select the name of the profile in the **Deployment** section of the Project Properties dialog and click **Edit**.

9.9.7 How to Deploy a Java Client Web Application Archive for Java Web Start

You can use the JDeveloper simple Java EE web deployment process to set up the web server before downloading and running the application using Java Web Start.

Once the application resides on the web server, it becomes very easy to maintain. Java Web Start takes care of identifying and downloading application updates each time the user runs the application.

To deploy Java client applications to the web server:

1. If not already done, create a Java Client Web Archive for Java Web Start.
2. If not already done, create an application server connection.
3. Create a simple JAR archive of your Java client application.
4. Create a Web Start JNLP Definition for Java Clients to generate the JNLP file and HTML file for use with Java Web Start.
5. Select and right-click project in the Application Navigator. The context menu displays these deployment options:
   - **Deploy > deployment profile > to most-recent** to deploy the project to the application server or archive file you previously chose.
   - **Deploy > deployment profile > to application server connection** creates the archive type specified in the deployment profile, and deploys it to the selected application server connection.
   - **Deploy > deployment profile > to EAR file** to deploy the project and any of its dependencies (specified in the deployment profile) to an EAR. JDeveloper puts the EAR file in the default directory specified in the deployment profile.
   - **Deploy > deployment profile to > WAR file** the web module is packaged as a WAR file and saved to the local directory you specified earlier in the deployment profile settings.
6. (Optional) If you want to edit the web.xml deployment descriptor, right-click the web.xml file in the Application Navigator and choose **Open**.
7. (Optional) To reopen a project deployment profile later to make changes, right-click the project in the Application Navigator and choose **Project Properties**, then select the name of the profile in the Deployment section of the Project Properties dialog and click **Edit**.

Note: Make sure that the web application deployment descriptor is located inside the Web Application Archive (WAR) file WEB-INF/web.xml.

9.10 Deploying Using Weblogic SCA Spring

The Oracle JDeveloper Weblogic SCA Spring Extension provides integrated support for WebLogic SCA and for the open-source Spring framework.

The extension allows you to create:

- WebLogic SCA enabled projects that can be deployed as a JAR file which can then be included in an EAR file for deployment, or as a WAR file.
- Spring framework projects.

9.10.1 About WebLogic SCA

The extension provides support for creating WebLogic SCA applications in JDeveloper and deploying them in Oracle WebLogic Server. WebLogic SCA is based on a subset of the OASIS Service Component Architecture Spring Component Implementation Specification. For more information, see http://www.oasis-open.org.

Service Component Architecture (SCA) provides a model for building enterprise applications and systems as modular business services that can be integrated and reused. WebLogic SCA provides support for developing and deploying SCA applications using POJOs (Plain Old Java Objects). In SCA, the implementation of a component and its communication are separate. In WebLogic SCA, you can write Java applications using POJOs and, through the different protocols available, expose components as SCA services and access them via references. You do this using SCA semantics configured in a Spring application context. In SCA terms, a WebLogic Spring SCA application is a collection of POJOs plus a Spring SCA context file that declares SCA services and references with the appropriate bindings. WebLogic Spring SCA applications can be used without modification as components in Oracle SOA composites.

In WebLogic Server, WebLogic Spring SCA applications run in the WebLogic SCA Runtime. The runtime must be deployed to WebLogic Server as a shared Web application library before applications can be deployed to it. For more, see Section 9.10.4.3, "How to Deploy WebLogic SCA Applications to Integrated WebLogic Server."

For more information about Oracle WebLogic SCA, see *Oracle Fusion Middleware Developing WebLogic SCA Applications for Oracle WebLogic Server*.

9.10.2 About Spring

Spring is an open-source framework that simplifies development of enterprise Java applications. The Spring framework includes models for various layers and functionality areas of Java applications. It focuses on using POJOs, leverages inversion of control concepts and dependency injection, and implements aspect oriented programming.

The Weblogic SCA Spring Extension provides integrated support for creating open source Spring projects in JDeveloper that can be used in Java EE applications. It adds the Spring JAR files as a library to JDeveloper, and it adds a wizard and editing features for creating Spring Bean configuration files. The extension creates: Adds the Spring JAR files as the Spring 2.5 library to JDeveloper. Adds a wizard for creating Spring Bean configuration files Registers the relevant XSDs and DTDs with the IDE to provide a productive editing experience for Spring definitions

For more information about Spring, see *Oracle Fusion Middleware Spring Support in Oracle WebLogic Server*.

9.10.3 Installing the Weblogic SCA Spring Extension

In order to use the Oracle JDeveloper Weblogic SCA Spring Extension, you must download it and install it...

The extension adds the following to JDeveloper:

- The Spring category to the Business Tier in the New Gallery. The options for creating the Spring Bean Configuration file and the WebLogic SCA Configurations are available here.

- The Spring 2.5 library is added to JDeveloper, along with the JAR files of the Spring framework and support for WebLogic SCA.

9.10.4 Using Oracle WebLogic SCA

You can use the Weblogic SCA Spring Extension to create WebLogic SCA enabled projects that can be deployed as a JAR file which can then be included in an EAR file for deployment, or as a WAR file.

9.10.4.1 How to Create WebLogic SCA Projects

You begin developing a WebLogic SCA project by creating the WebLogic SCA Configuration file which acts as the control file for the application. As part of this process, JDeveloper configures either the JAR or WAR deployment descriptor for WebLogic SCA so that the necessary libraries are deployed to the server.

To create a WebLogic SCA application:

1. Create a Java application and project.
2. Choose **File > New > New Gallery > Business Tier > Spring**.
3. Choose either:
   - **WebLogic SCA Configuration for JAR** deployment to create a project that includes a JAR file that can be included in an EAR file for deployment.
   - **WebLogic SCA Configuration for WAR deployment** to create a project that includes a WAR file.

What the WebLogic SCA Wizard Does

When you run the WebLogic SCA Configuration wizard, the following happens:

- An SCA definition file called spring-context.xml is created in `META-INF/jsca` and opened in the JDeveloper XML source editor. You can use the advanced XML editing framework to assist you as you edit it.
- If the project does not already contain a `web.xml` file one is created.
- Depending on the option you choose in the New Gallery:
  - A JAR deployment descriptor is added to the project, and a dependency on the weblogic-sca shared library is added at application level.
  - A WAR deployment descriptor is added to the project, and a dependency on the weblogic-sca shared library is added at web application level.

Next Steps

Once you have created an SCA project, you can:

- Deploy the application to Oracle WebLogic Server.
- Test the application with the JDeveloper Integrated WebLogic Server.

9.10.4.2 How to Edit Oracle WebLogic SCA Definition Files

The SCA definition file created when you create a WebLogic SCA project is called spring-context.xml, and it is created in `META-INF/jsca` and opened in the XML source editor.

The outline `spring-context.xml` file is

Example 9–8 Outline spring-context.xml File

```
<?xml version="1.0" encoding="windows-1252" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
http://www.springframework.org/schema/tool
http://www.springframework.org/schema/tool/spring-tool-2.5.xsd
http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
  <!--Spring Bean definitions go here-->

</beans>
```

The comment shows where you enter the bean definitions.

Use the XML source editor features, Structure Window features, the Component Palette, and the Property Inspector to navigate the hierarchy of the XML file and edit it.

Source Editor Features

The source editor has a number of features which help you to edit an XML file.

- XML Code Insight, the XML-specific implementation of completion insight. Type < and wait for a second, and JDeveloper will pop-up the possible entries appropriate for that location. If the tag you chose has mandatory attributes, JDeveloper will automatically add them.
- The XML source editor provides many features to help, for example, errors are underlined with a curly red line.
- You can choose options from the context menu such as Find Usages, which will display all the usages of the element in the Usages log window. You can also use Find Usages from the Structure Window.

Structure Window Features

The Structure Window allows you to quickly navigate the hierarchy of the XML file, and it also offers editing features.

- Right click on nodes in the Structure Window to add more components.
- Error messages are displayed in the Structure Window.

Component Palette Features

You can select tags from the Component Palette and drag and drop them directly into the source editor or the Structure Window to build spring-context.xml files.

> **Note:** You can only drop tags in places that are correct in terms of syntax.

By default, the Component Palette displays all the available tags. Click All Pages and choose just the type of tags you want to reduce the number of tags displayed. For example, to use WebLogic SCA Bindings, choose that option at the top of the component palette and the components listed are EJB Binding and Webservice Binding.

Property Inspector Features

The Property Inspector allows you to edit the properties of tags.

- Changes in the property inspector are synchronized with the source editor view.
- Lists of values are shown when they are relevant for a specific property.

9.10.4.3 How to Deploy WebLogic SCA Applications to Integrated WebLogic Server

Once you have created a WebLogic SCA project you can test the application by quickly deploying it to the Integrated WebLogic Server.

To deploy to Integrated WebLogic Server:

- In the Application Navigator right-click spring-context.xml under the project node in the Application Navigator, and choose Run, or Debug, or one of the Profiler options.

What Happens When You Run the Application in Integrated WebLogic Server

If Integrated WebLogic Server has not yet been started, the default domain is automatically created with default settings and the server is started.

The application is deployed to Integrated WebLogic Server and the Log Window displays messages that show the progress of the deployment.

In the Application Server Navigator, you can see the services deployed under the Web Services and EJB nodes under IntegratedWebLogicServer.

9.10.4.4 How to Deploy WebLogic SCA Applications to Oracle WebLogic Server

Once you have created a WebLogic SCA project you can deploy it to Oracle WebLogic Server.

The process is slightly different depending on whether you chose to create a JAR or a WAR file.

> **Note:** Before you deploy a WebLogic SCA application to Oracle WebLogic Server, you must install WebLogic SCA on the server. For more information, see the chapter about deploying WebLogic SCA Runtime to WebLogic Server in the *Oracle Fusion Middleware Spring Support in Oracle WebLogic Server*.

To deploy an Application Containing a WebLogic SCA WAR File to WebLogic Server:

- Deploy the application as usual.

To deploy an Application Containing a WebLogic SCA JAR File to WebLogic Server:

1. Set the location of the JAR file to be either lib or APP-INF/lib and deploy it into an EAR file.

> **Note:** The EAR file must contain at least one other Java EE artifact, for example a WAR file or EJB-JAR or the deployment will fail.

2. Deploy the application as usual.

What Happens When You Deploy the Application to WebLogic Server

If necessary, an EAR file is created.

The application is deployed to the WebLogic Server connection and the Log Window displays messages that show the progress of the deployment.

In the Application Server Navigator, you can see the services deployed under the Web Services and EJB nodes under the connection node for the application server.

9.10.5 Using Spring

The Weblogic SCA Spring Extension provides integrated support for the open-source Spring framework. The extension adds a number of libraries to JDeveloper, and adds support for creating Spring framework projects

9.10.5.1 How to Create Spring Bean Applications

The Weblogic SCA Spring Extension adds libraries to JDeveloper containing the JAR files of the Spring framework. You begin developing a Spring framework application by creating the Spring Bean Configuration file, which acts as the control file for the application.

To create a Spring Bean Configuration file:

1. Create a Java application and project.
2. Choose **File > New > New Gallery > Business Tier > Spring > Spring**. Enter the file name and directory for the Spring Bean definition file and click OK.

9.10.5.2 What Happens When You Create a Spring Bean Configuration File

When you create a Spring Bean Configuration from the Spring category in the Business Tier of the New Gallery, the Spring 2.5 and Commons Logging 1.0.4 libraries are automatically added to the project. You can access the library definitions by choosing Project Properties from the context menu of the project in the Application Navigator, and then choosing Libraries and Classpath.

The Spring Bean Configuration file, beans.xml, is created in `META-INF/jsca` and opened in the XML source editor. You can use the advanced XML editing framework to assist you as you edit it.

9.11 Troubleshooting Deployment

There a number of common problems that you may come across when deploying applications. This topic describes them and their solutions. It is divided into issues that may arise when deploying to both Integrated WebLogic Server and Oracle WebLogic Server, and issues that are specific to one or other type of deployment

9.11.1 Common Deployment Issues

This section contains information about issues that may arise when deploying to both Integrated WebLogic Server and Oracle WebLogic Server.

9.11.1.1 [Deployer: 149164] The domain edit lock is owned by another session in exclusive mode - hence this deployment operation cannot proceed

Oracle WebLogic Server instances use the domain edit lock to make sure that only one user can deploy applications and change configurations at one time, and this message is displayed when another deployment is going on at the same time (only one deployment at a time is allowed), or some change has been made in the WebLogic Server Administration Console that has not been activated. Rarely, this message may also appear when you are running an application on Integrated WebLogic Server.

To activate a change in the WebLogic Server Administration Console:

1. Log in to the Administration Console.
2. In the Change Center, at the upper left of the console, click View changes and restarts.
3. In the Changes and Restarts section, ensure that the Change List tab is selected, and activate any pending changes.
4. Select the Restart Checklist tab, and select the server to restart, and click Start.

To enable or disable the domain configuration locking feature, see the section about enabling and disabling the domain configuration lock in the Administration Console Online Help, which is available from the WebLogic Server online documentation in your JDeveloper installation, or from the Administration Console.

If the error has appeared when you are deploying to Integrated WebLogic Server, you can check the Administration Console to determine what the problem is.

9.11.2 How to Troubleshoot Deployment to Integrated Application Servers

This section contains information about issues that are specific to running on integrated application servers.

9.11.2.1 Stopping Integrated Application Server

If you need to stop integrated application server, for example, to clear out an orphaned WebLogic Server instance that was created and left running from an earlier JDeveloper session, and you are unable to do so from within JDeveloper, go to `jdeveloper-user-home/DefaultDomain/bin`, and run `stopWebLogic.cmd` (on Windows) or `stopWebLogic.sh` (on Linux). This gracefully shuts down the integrated application server so that it will not conflict with subsequent attempts to launch the integrated application server from JDeveloper.

You can force shutdown of an instance that is still actively under the JDeveloper control (i.e., not orphaned) by pressing the **Terminate button** twice.

9.11.2.2 Running Out of Memory

If you run multiple applications on Integrated WebLogic Server, you may run out of memory and see the `java.lang.OutOfMemoryError: PermGen` space exception. To avoid this, increase the `MEM_MAX_PERM_SIZE` from the default of 128m to 256m, 512m, or higher. This is set in `setDomainEnv.cmd` (Windows) or `setDomainEnv.sh` (Linux), which is located at `jdeveloper-user-home/DefaultDomain/bin`.

You first need to stop Integrated WebLogic Server using one of the methods described above.

9.11.2.3 Reinstalling JDeveloper in a Different Location

If you reinstall JDeveloper into a new location, you may find that you have problems because the integrated application server uses some hard-coded references to JDeveloper. You must do one of:

- Set JDEV_USER_DIR to use a new system directory. This is described in "Setting the User Home Directory" in the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.
- Delete the old system directory, so that JDeveloper regenerates a new system directory.
- In the Application Server Navigator, right-click on IntegratedWebLogicServer and select **Delete Default Domain**.

9.11.3 How to Troubleshoot Deployment to Oracle WebLogic Server

This section contains information about issues that are specific to deploying to Oracle WebLogic Server.

9.11.3.1 ORA-01005: null password given; logon denied

This is usually caused by a blank password in the <encrypted-password> entry of the application-name-jdbc.xml file or no <encrypted-password> entry at all.

9.11.3.2 ORA-01017: invalid username/password; logon denied

This is usually caused by the wrong password in the <encrypted-password> entry of the application-name-jdbc.xml file.

9.11.3.3 [Oracle JDBC Driver] Kerberos Authentication was requested, but is not supported by this Oracle Server

This will cause logon to be denied, and it is due to using the Oracle WebLogic Server database driver, weblogic.jdbcx.oracle.OracleDataSource. This driver is not certified by Oracle and should not be used.

9.11.3.4 Application Does Not Work After Creating a Global Data Source from the Oracle WebLogic Server Administration Console

Make sure there is a target domain selected for the data source. If you clicked **Finish** before the last panel of the wizard, then this was not done.

Also, make sure that the Java naming lookup call is correct if you are using a lookup in Java code. For example, if the connection name is connection1, the naming lookup should be java:comp/env/jdbc/connection1DS.

9.11.3.5 Redeploying an Application to a Server that is Down

You can only deploy an application once to a server that is down.

If you attempt to redeploy the same application to the same down server a second time, deployment fails with the following log message:

```
[03:29:47 PM] ---- Deployment started. ----
[03:29:47 PM] Target platform is (Weblogic 10.3).
[03:29:47 PM] Retrieving existing application information
[03:29:47 PM] Running dependency analysis...
[03:29:47 PM] Building...
[03:29:50 PM] Deploying 2 profiles...
[03:29:50 PM] Wrote Web Application Module to
/path/oracle/jdeveloper/jdev/mywork/Application1/Project1/deploy/webapp1.war
[03:29:50 PM] Wrote Enterprise Application Module to
/path/oracle/jdeveloper/jdev/mywork/Application1/application1.ear
[03:29:50 PM] Redeploying Application...
[03:29:50 PM] [Deployer:149034]An exception occurred for task
[Deployer:149026]deploy application application1 on Server-1.:
[DeploymentService:290049]Deploy failed for id '1,244,759,390,503' since no
targets are reachable..
[03:29:50 PM] Weblogic Server Exception: java.lang.Exception:
[DeploymentService:290049]Deploy failed for id '1,244,759,390,503' since no
targets are reachable.
[03:29:50 PM] See server logs or server console for more details.
[03:29:50 PM] java.lang.Exception: [DeploymentService:290049]Deploy failed for id
'1,244,759,390,503' since no targets are reachable.
[03:29:50 PM] #### Deployment incomplete. ####
[03:29:50 PM] Remote deployment failed
```

9.11.3.6 Attempting to Deploy to a Server that No Longer Exists

When you have successfully deployed an application to a Managed Server, the deployment wizard saves this deployment action in its history so that you can perform the same action later. However, if the Managed Server is removed from your Oracle WebLogic Server domain and you subsequently deploy using the deployment history action, deployment fails with the following log message:

```
[02:38:40 PM] ---- Deployment started. ----
[02:38:40 PM] Target platform is (Weblogic 10.3).
[02:38:40 PM] Retrieving existing application information
[02:38:40 PM] #### Deployment incomplete. ####
[02:38:40 PM] [J2EE Deployment SPI:260013]Target array passed to DeploymentManager
was null or empty.
```

9.11.3.7 Deploying to a remove server fails with HTTP Error Code 502

If you are deploying to a server running on a machine that is not known to the network DNS server, and you have set a proxy for JDeveloper, deployment will fail with a 502 HTTP error code. This is because the proxy does not know where to forward the request. This will also happen if you are deploying to a server on the localhost that is referred to by its machine name, which typically happens with SOA development.

To avoid this happening either add the machine to the Exceptions list in the proxy settings in the Web Browser and Proxy page of the Preferences dialog, or choose not to use a HTTP Proxy Server for any connections.

9.11.3.8 No Credential Mapper Entry Found

If you see the following message, it usually means that an EAR using password indirection did not have the passwords injected via mbeans before deployment.

```
weblogic.common.ResourceException: No credential mapper entry found for password
indirection user=scott for data source Connection1
```

This usually happens when trying to deploy an EAR manually from the console or from an ant script.

9.11.4 How to Troubleshoot Deployment to IBM WebSphere

This section contains information about issues that may arise when deploying to both Integrated WebLogic Server and Oracle WebLogic Server.

9.11.4.1 Deployment Fails When EAR Contains Spaces

WebSphere deployment on Windows does not work when the directory containing the EAR generated by JDeveloper contains spaces.

9.11.4.2 Application Displays Administrative Console User Name

When you deploy your application to IBM WebSphere application servers and use the same machine to log into the WebSphere administrative console, your application may display the name of the user logged into the administrative console, instead of the name of the user who logs into the application.

Part III

Developing Java EE Applications

This part describes how to develop Java EE applications with Oracle JDeveloper including all of the frontend and business technologies you need to get your enterprise application up and running start to finish. This part of the book contains the following chapters:

- Chapter 10, "Getting Started with Developing Java EE Applications"

  This chapter introduces the technologies supported to build your Java EE Application.

- Chapter 11, "Developing Applications Using Web Page Tools"

  This chapter introduces the features, and covers tasks related to building web pages and related business components with HTML, JSF/facelets, JSP, servlets and scripting. It also provides a detailed look at the related tools and technologies available for web page development.

- Chapter 12, "Developing with EJB and JPA Components"

  This chapter introduces the features, and covers building your business components with EJB and JPA, including session bean and persistence technologies.

- Chapter 13, "Developing TopLink Mappings"

  This chapter covers TopLink technology and related steps for building persistence functionality on object-oriented programs based on relational data structures.

- Chapter 14, "Developing Secure Applications"

  This chapter introduces the features, and covers the steps of developing, deploying and administering secure applications.

- Chapter 15, "Developing Applications Using XML"

  This chapter introduces XML mappings and configuration files technologies, as well as creating and exiting the XML files.

- Chapter 16, "Developing Applications Using Web Services"

  This chapter introduces the features, and covers the discovery and implementation of existing system web services, and steps for deploying new web services.

10

Getting Started with Developing Java EE Applications

This chapter overviews the Java EE features available for your application development, and related components, tools and technologies provided in Oracle JDeveloper.

This chapter includes the following sections:

10.1 About Developing Java EE Applications

JDeveloper comes with a complete package of tools and features to create and edit your Java EE 6 application components. Use the wizards, built in source and visual editors, Component Palette and property inspector, and other features to create, assemble, and reuse your web tier and business components. You can build, test, and deploy powerful interactive, multitiered applications that perform well on a variety of different platforms, and are easy to maintain.

For more information on Java EE see the Oracle Technology Network (OTN) Java EE documentation at:
`http://www.oracle.com/technetwork/java/javaee/overview/index.html`

10.1.1 Java EE and Oracle Application Developer Framework

For the web-tier part of your Java EE application, take advantage of the ADF Faces rich client framework (RCF), which offers a rich library of AJAX-enabled UI components for web applications built with JavaServer Faces (JSF).

The ADF layer enables a unified approach to bind any user interface to any business service, without need to write code. When you build a Java EE application, and/or an EJB project, you can assign ADF data controls on your individual session beans. This adds a data control file with the same name as the bean.

The data control contains all the functionality of the application module. You can then use the representation of the data control displayed in JDeveloper Data Controls panel to create UI components that are automatically bound to the application module.

Using the ADF data control business-tier layer to perform business service access for your EJB projects ensures that the view and the business service stay in sync. For example, you could bypass the model layer and call a method on an application module by class casting the data control reference to the application module instance and then calling the method directly, but this renders the business services unaware of any changes.

For more information, see the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

10.2 About Web Page Tools

JDeveloper provides you with a wide range of tools to develop the web tier, or frontend of your Java EE applications. You can use wizards to walk you through creating all your HTML, JSP and JavaServer Faces (JSF) /Facelet pages and related files.

In addition, JDeveloper provides web page tools and step-by-step instructions for many of the tasks you will use to develop your application web pages. You can build web-tier components using all of the supported Java EE web application technologies such as JSF / Facelets, JavaServer Pages (JSP), Java Servlet, HyperText Markup Language (HTML), and Cascading Style Sheets (CSS). Web components in a Java EE application contain presentation logic and run on the integrated server.

For more information, see Chapter 11, "Developing Applications Using Web Page Tools."

10.3 About Enterprise JavaBeans and Java Persistence Components

You can create EJB projects, entities, Java persistence units, session beans, and message-driven beans using wizards in the New Gallery. You can build entities from online or offline database table definitions and from application server data source connections.

To quickly get started with your EJB application:

- Start by using the wizard **(File > New > General > Applications)** to create the framework for your Java EE application.
- Use wizards to create entities that correspond to database tables **(File > New >Business Tier > EJB)**.
- Use a wizard to create session beans and facades and to build a persistence unit. **(File > New > Business Tier > EJB)**. Oracle ADF provides components to enable data controls **(File > New > Business Tier > ADF Business Components)**.
- Use the JDeveloper integrated server capabilities to test your application. For more information on running and testing, see Chapter 7.3, "Running Applications."

For more information on EJBs, see Chapter 12, "Developing with EJB and JPA Components."

10.4 About Oracle TopLink

Oracle TopLink is an object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality

Use TopLink to configure TopLink descriptors and map Java classes, EJBs, and JPA entities to different data sources, including relational databases, enterprise information systems (EIS), and XML schemas. With the TopLink Editor, you can create this information without writing Java code. The TopLink Editor supports multiple standards, including JPA, JAXB, and Java EE.

For more information, see Chapter 13, "Developing TopLink Mappings."

10.5 About Secure Applications

You can secure Java EE applications using only container-managed security or, for Fusion web applications, Oracle ADF Security. Fusion web applications are Java EE applications that you develop using the Oracle Application Development Framework (Oracle ADF).

The Oracle ADF Security framework is the preferred technology to provide authentication and authorization services to the Fusion web application. The Oracle ADF Security is built on top of the Oracle Platform Security Services (OPSS) architecture, which provides a critical security framework and is itself well-integrated with Oracle WebLogic Server.

For more information, see Chapter 14, "Developing Secure Applications."

10.6 About Applications That Use XML

JDeveloper provides you with the tools you need to work with the XML files in your application. There is an XML source editor, an XML validator, and tools for working with XML schemas. You can also use JDeveloper to create and edit your XSQL files.

You can create your schema documents from scratch, generate schemas from XML documents or vice-versa in JDeveloper. Once your schema is created, manage your elements using the XSD Visual Editor and the Component Palette.

For more information, see Chapter 15, "Developing Applications Using XML."

10.7 About Applications That Use Web Services

Web services in JDeveloper provides a set of messaging protocols and programming standards that expose business functions over the internet using open standards. A web service is a discrete, reusable software component that is accessed programmatically over the Internet to return a response. JDeveloper provides tools that help you discover and use existing web services, and develop and deploy new web services.

JDeveloper also supports a set of standard Java-to-XML type mappings. You can also create custom serializers for types of objects that are not automatically supported. For more information, see Section 16.2, "Using JDeveloper to Create and Use Web Services."

You can create web services from Java classes, the remote interface of EJBs, and an ADF Business Components service session bean wrapped as an EJB. The Web service creation wizards create the deployment files for you, so once you have created your

web service the final step is to deploy it to application servers. For more information, see Section 16.5, "Creating SOAP Web Services (Bottom-Up)."

Alternatively, you can create a web service starting with a WSDL, as a top-down web service. For more information, see Section 16.6, "Creating SOAP Web Services from WSDL (Top Down)."

Finally, you can develop web services that are based on Representational State Transfer (REST). A RESTful web service is a simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. Section 16.7, "Creating RESTful Web Services."

11

Developing Applications Using Web Page Tools

This chapter describes how to build Java EE applications user interfaces and business services with HTML, JSP, and JSF/facelets using the latest tools and technologies included in Oracle JDeveloper.

This chapter includes the following sections:

11.1 About Developing Applications Using Web Page Tools

Oracle JDeveloper provides you with a wide range of tools to develop the frontend or view layer of your Java EE applications. There are handy wizards to walk you through creating all your HTML, JSP and JSF/facelet pages and related files. When you create web pages using the wizards your configuration files, bean mappings, tag libraries, and jar files are automatically set up and editable.

At the forefront of the web tools there are source editors, visual editors, and integrated component and property tools to add and edit the pages, elements and related properties in your pages, including your business service and localization components. You will be able to create and modify your style sheets and tag libraries, and use the Code Insight code and tag completion tools to efficiently code your HTML, JSP and JSF/facelet or Java source files.

This chapter walks you through the web page tools and step-by-step instructions for many of the tasks you will use to develop your application web pages.

11.1.1 Getting to Know the Source Editor Features

The source editor is your basic code editor for most of your web pages. You will use the source editor to add non-visual components, and custom coding, in conjunction with the visual editor which allows you to drop in components and visually modify

your pages. When you are using the source editor, there are many features to make coding tasks faster and easier. Figure 11–3 displays a source editor for a JSF page.

Figure 11–1 Source Editor with Typical JSF Code

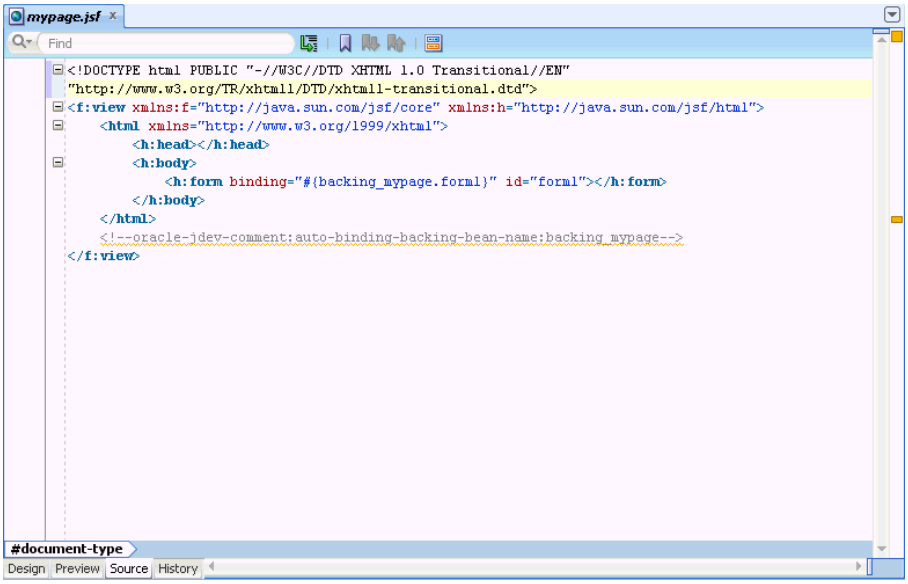


Table 11–1 lists the primary source editor features.

Table 11–1 Primary Source Editor Features

Features	Description
Quick Doc for Tags	View your tag definitions while you're coding. Put your curser on the tag and press Ctrl + d. A small window appears at the top of your editor with that tag definition detail. Click back in the editor and the window closes. You can also right-click and choose **Quick TagDoc**.
Code Templates	Save time by inserting pre-written code into source files instead of having to type it in manually. Templates can intelligently modify the inserted code to suit surrounding code. Use shortcuts to speed up the selection of the required template. See Section 3.8.3, "How to Customize Code Templates for the Source Editor" for more information on templates.
Code Insight	View and filter element and parameter options, and get code completion. The source editor provides Code Insight for tags, attribute names & values, and embedded CSS & JavaScript code.
Jump to Managed Bean	Quickly jump to your managed bean code from your web page source. Right-click in the source editor or Structure window and choose **Go to**, then select your choice from the list of all beans referenced from that page.
Editor Splitting	Toggle between code and visual views using the splitter. To split the file horizontally, grab the splitter just above the vertical scroll bar on the upper right-hand side of the window and drag it downward. To split the file vertically, grab the splitter just to the right of the horizontal scroll bar on the lower right-hand side of the window and drag it left.

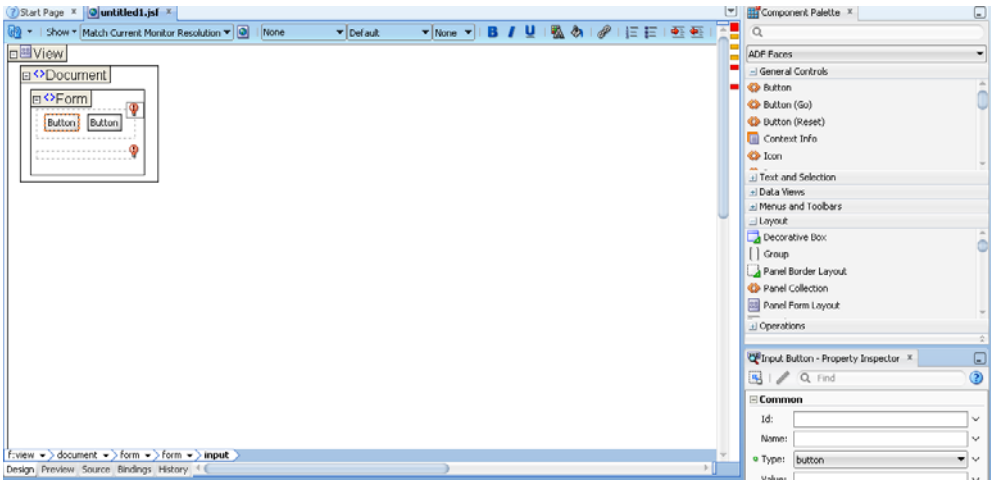
11.1.2 How to Work in the Visual Editing Environment

The JSP/HTML Visual Editor can be used for WYSIWYG editing of all your web pages including JSP, JSF, facelets, and HTML pages. The visual editor opens up with the Component Palette available to drop and drag components to the page, as shown in Figure 11–2.

Use the visual editor for the following tasks:

- Insert visual and non-visual page elements.
- Apply cascading style sheets and modify text style.
- Move and resize page elements.
- Design tables and forms.
- Modify web page element attributes.
- Select the reference device for mobile-enabled JSP documents.

Figure 11–2 Visual Editor Showing Typical JSF Page



With the exception of dynamic content such as JavaScript elements, all your web page elements are visually displayed or structurally represented in the visual editor. JSP tags including JSTL, and BC4J tags are visually displayed in the visual editor using icon and tag names, while HTML template content is rendered based on the browser look and feel. You can toggle back and forth or split the screen to see the source editor during design-time.

The visual editor is integrated with the Java Source Editor, Structure window, Component Palette, Property Inspector, and Data Binding Palette to support the assembly of databound web pages using simple drag and drop operations. You can move from one tool to another and see or edit your changes reflected across the board immediately.

Key visual editor features include the following:

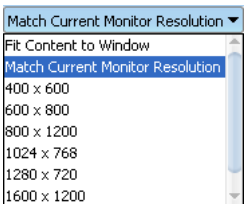
- When you open a file in the visual editor, the page is rendered in HTML and associated web formats, much like a web browser. You immediately see the results of your edits.
- View and select your nested components in chronological order using the breadcrumb that appears at the bottom of the visual editor window. Selecting an

element on a page opens the element attributes for editing in the Property Inspector.

- View the structure of data in the Structure window. You can view the data in the document currently selected in the active window of those windows that participate in providing structure: the diagrams, the navigators, the editors and viewers, and the Property Inspector. For more information, see Section 3.11.6, "Structure Window."
- Component Palette tag library pages are context-sensitive, displaying eligible components for insertion into the page.
- Right-click anywhere within the visual editor to bring up a context-sensitive menu of commands.

The visual editor comes with a toolbar at the top of the window that includes the usual toolbar icons to format font and paragraphs on your web pages. In addition you can set the window of the visual editor to your preferred resolution using the **Match Current Monitor Resolution** dropdown tool, as shown in Figure 11–3.

Figure 11–3 Match Current Monitor Resolution Tool



Use the **Match Current Monitor Resolution** drop-down list to choose a larger or smaller page picture in your visual editor window. Changing the monitor resolution does not impact the actual page or browser size, but only the way it is viewed in this editing window. By default the visual editor window is set to preview in the same resolution as the monitor you are using. You can also set it to fit the current size of the visual editor.

There are also some additional editing tools and features, as detailed below in Table 11–2.

Table 11–2 Toolbar Icon Features on the Visual Editor

Icon	Name	Description
	Refresh	There are two types of refresh for you to choose from. Use the dropdown menu on the refresh button. **Refresh Page** rebuilds and re-renders the internal data structures of a page. Use this tool if you have an included page (like a page template) that has been changed, and you want to see the affects in the including page. **Full Refresh** is used to first fully restart the internal design time for a page project (which includes rebuilding the servlet context from web.xml and tag libraries, and (for Faces projects) the Faces context from the faces-config.xml. With Full Refresh the internal data structures of the active page are rebuilt and it is re-rendered.

Table 11–2 (Cont.) Toolbar Icon Features on the Visual Editor

Icon	Name	Description
	Show component status	■ Use the dropdown list to choose options for component action and status. ■ Select the default Component Actions to display an arrow for a component when it has focus. ■ Click the arrow to display a list of possible actions relevant for that component, such as hide, rebound, or go to page definition. ■ Use to select if the component will be marked for a Warning, Incomplete, or Error status. You can also select to show your text resource and expression builder values in various forms, or not at all on your components. ■ For your EL Expression rendering choose Show User-Friendly Values (Resolve Bindings) to automatically resolve your expressions for viewing. This is selected by default. Values that are not resolved are rendered according to your preference for Fallback Data Display Style. ■ You can select to abbreviate your expressions, show no expressions, or populate the page with dummy data for display purposes only. You can also choose to display the full expression.
	Preview in Browser	Lets you see your how your web page will appear in your default browser. Click the icon or Shift+F10,V.

Table 11–3 lists the features that are available with simple keystroke commands while you are editing your web pages.

Table 11–3 Primary Visual Editor Command Features

Features	Description
Toggle Line Comments	Adds or removes comment markers from the beginning of each line in a selected block. Select a single line to comment or uncomment that line only.
Breadcrumbs	View and select your nested components in chronological order using the breadcrumb that appears at the bottom of the visual editor window.
Component Selection	Hovering your curser over a component on the page highlights that component with an orange outline.
Editing Containers	In the Structure window or visual editor window select a container. Right-click that container and choose **Design This Container**. That container is selected in the editing window. This feature allows you to more easily view and edit the individual components in that selected container.
Visual EL Expression View Preferences	Select whether to resolve expressions for viewing, and how to view those that are unresolved. Choose the **Show** toolbar feature to select your preference for EL rendering, or you can also go to **Tools >Preferences > JSP and HTML Visual Editor**.

Table 11–3 (Cont.) Primary Visual Editor Command Features

Features	Description
Expression Builder and Text Popup	Select your component. Slow double-click or F1 to open a popup window with a value field for editing your expressions or text.
Corresponding Element Display	Page elements are displayed hierarchically in the Structure window. Double-clicking a node in the Structure window shifts the focus to the Property Inspector.
Visual and Code Editor Splitting	Edit your file simultaneously with the visual and source editors by opening the page in one of the editors and using the splitter to open a second page view in the alternate editor. To split the file horizontally, grab the splitter just above the vertical scroll bar (on the upper right-hand side of the window) and drag it downward. To split the file vertically, grab the splitter just to the right of the horizontal scroll bar (on the lower right-hand side of the window) and drag it left.
Easy Edit Focus	By default new JSP or HTML pages are opened with the visual editor in focus. Double-clicking a node in the Application Navigator opens or brings the default editor to the foreground. To locate the node in the Application Navigator that corresponds to the file you are currently working on, right-click and choose **Select in Navigator**. Or use the keyboard shortcut (default keymap, Alt+Home).
Tag Display	The scope of tags with embedded body content is structurally represented with the tag icon and name bounded by, or contained within, a shaded rectangle. These tag containers are nested or structurally displayed to represent, for example, an iterated row in a table. Click the tag to select a cursor for entering content in the tag container.
Extracting CSS code from HTML/JSP to a CSS files	Extract a CSS block from a HTML/JSP file to a new CSS file and all the references are updated automatically. This option is available to use from the Code editor and the Structure window.
Style sheet Linking to HTML files	Link a style sheet to your HTML files simply by dropping a `<style>` or `<link>` tag from the Component Palette common tab into your HTML page.
Mobile Device Display	For mobile-enabled JSP documents, the design view emulates the viewport of the selected device category. The device category icon is displayed on the toolbar along with the reference device dropdown list.

11.1.2.1 How to Expand and Collapse Container Elements

Another feature that can be used while working in the visual editor or Structure window is the ability to expand or collapse JSP and HTML page elements containing or nesting other elements. To do this use the - (minus) and + (plus) sign to expand and collapse the parent container element.

For example, in the visual editor, JSP container tags are displayed as nested rectangles. An expanded JSP `<c:choose>` tag containing a `<c:when>` and `<c:otherwise>` tag displays as shown in Figure 11–4.

In the Structure window, the example of a collapsed HTML table with multiple rows displays as shown in Figure 11–5.

Figure 11–4 Container Tags in Nested Rectangles

Figure 11–5 Collapsed HTML Table

To collapse the container element:

Do one of the following:

- Click the + (plus) sign of the container element.
- Right-click the container element and choose **Expand Tag** from the context menu.

11.1.2.2 How to Customize the Visual Editor Environment

You can customize the following default visual editor environment settings such as:

- Text foreground and background color, element and tag outline color, and caret color.
- Synchronization between the visual editor and the Structure Window or the source editor.
- Display of errors and warnings.
- Display of tag names.

To change the general environment settings for the visual editor:

1. From the main menu, choose **Tools** > **Preferences.**
2. In the Preferences dialog, select the **JSP and HTML Visual Editor** node.
3. Select the options and set the fields as appropriate.
4. Click **OK**.

11.1.2.3 How to Display Invisible Elements

You can customize the display of invisible elements such as:

- HTML named anchors, script tags, and line breaks.
- JSP tag library directives and bean tags.

To change the display of invisible elements in the visual editor:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **JSP and HTML Visual Editor** node and select the **Invisible Elements** page.
3. Select your options and click **OK.** After setting preferences you can toggle the display of invisible elements on and off when you are working in the visual editor by going to the main menu and choosing **Design** > **Show** and select **Invisible HTML Elements** or **Invisible JSP Elements.**

11.1.2.4 How to Execute JSP Tags in the JSP Visual Editor

To get a close approximation of a runtime visualization, run the tag library in a simulated JSP/Servlet container available in the design time page context.

To set a JSP Tag Library to execute at design time:

1. From the main menu, choose **Tools** > **Manage Libraries**.
2. In the Manage Libraries dialog, select **JSP Tag Libraries**.
3. Choose a tag and select **Execute Tags in JSP Visual Editor**.
4. Click **OK**.

To set this option for a particular project:

1. In the Application Navigator, double-click the project.
2. Choose **JSP Tag Libraries**.
3. Select **Execute Tags in JSP Visual Editor**.

11.1.2.5 How to Display JSP Tags by Name Only

Display JSP tags by name only, by omitting embedded EL syntax. For example, `<c:out value="${Row.Deptno}"></c:out>` would display simply as `out` vs. `${Row.Deptno}` if this select this option.

To display JSP tags by name only:

1. From the main menu, choose **Tools** > **Preferences**.
2. Choose **JSP and HTML Visual Editor**.
3. Select the **Show JSP Tag Name Only** checkbox. This checkbox is deselected by default.

11.1.2.6 How to Change Keyboard Preferences

Use the JDeveloper keyboard or mouse for navigating any of your development tasks. You can also customize the default keymap, and within each keymap specify any of the accelerator assignments.

To customize keymap accelerators:

1. From the main menu, choose **Tools** > **Preferences.**
2. In the Preferences dialog, select **Accelerators**.
3. Select a preset keymap, and make changes to the accelerator.
4. Click **OK.**

11.1.2.7 How to Select Web Page Elements

You can select a single element to manage, or select a container element along with included elements such as a table, or multiple elements. A dotted line encloses the selection. In the Structure window, a selected element is highlighted.

To select an element:

1. In the visual editor or Structure window, position your pointer cursor on the element.

2. Click the element. If the selected element contains included elements, selecting the element also selects all its contained elements. If you copy, move, or delete the container, all its contained elements are also copied, moved, or deleted.

   OR

   Right-click the element. When you select an element by right-clicking, a context menu of options is displayed. For example, to highlight the element code in the Source view of the page, right-click the element and select **Go to Source.**

   Note: Double-clicking an element brings up an editor dialog for the tag.

To select text:

In the visual editor in an open web page, do one of the following:

- Double-click a single word.
- Triple-click a word within a text string you wish to select.
- Select and drag your cursor across the text.
- Click at the start of the selection, scroll to the end of the selection, then hold down **Shift** and click.

To select multiple element:

1. In the visual editor or Structure window, position your pointer cursor on the element in an open web page.
2. Click the first element.
3. Press and hold down the Ctrl key.
4. Click any additional element. If you want to deselect one without losing the other selections, continue to hold down the Ctrl key and click the element again. Selecting multiple, non-adjacent elements for any reason other than deleting them might lead to unexpected results. For example, if the elements exist at different levels in the web page hierarchy, they can lose their relative hierarchical positions if you move or copy them to another position in the page.

To select a range of adjacent elements:

1. In the visual editor or Structure window, position your pointer cursor on the first element.
2. Click the element.
3. Scroll to the end of the selection, then hold down **Shift** and click.

Tip: For JSP tag libraries, when you pass the mouse pointer over an element, a tooltip with the tag name is displayed, making it easier to know where to click to select a element.

Press `Ctr +Shift+Up` to select the container element for an element contained inside another element. Do the same to move through nested containers until you reach your target. For example, select when you add a link to text you will need to press `Ctr +Shift+Up` twice to move to the link target.

When you select an element in the visual editor, it is also selected in the Structure window, and vice-versa. You can look at the selection in both tools to see what is selected and where the insertion point is.

11.1.2.8 How to Select Insertion Points in the Design Tools

While inserting, copying, or moving page elements, you select an insertion point on the page in the visual editor or in the structure window in relation to a target page element. JDeveloper provides visual cues to locate the insertion point before, after, or contained inside a target element.

To select an insertion point in the visual editor:

- When dragging a JSP or HTML page element to an insertion point, drag it until you see a vertical line | in the desired location, then release the mouse button.
- When selecting an insertion point by clicking, do one of the following:
  - Select the desired location on the page, indicated by a blinking cursor.
  - Select the element to contain the inserted element, indicated by a dotted line.

Note: Copying an element from the clipboard into a selected element will replace the selected element.

To select an insertion point in the Structure window:

When dragging a web page element to an insertion point, do one of the following:

- To insert an element before a target element, drag it towards the top of the element until you see a horizontal line with an embedded up arrow, then release the mouse button.
- To insert an element after a target element, drag it towards the bottom of the element until you see a horizontal line with an embedded down arrow, then release the mouse button.
- To insert or contain an element inside a target element, drag it over the element until it is surrounded by a box outline, then release the mouse button. If the element is not available to contain the inserted element, the element will be inserted after the target element.

When selecting a target position by clicking, highlight the target element.

Note: A disallowed insertion point is indicated when the drag cursor changes to a slashed circle.

11.1.2.9 How to Insert Web Page Elements

Use the Component Palette to add UI and data elements to your web pages. You can insert page elements in the visual editor or the Structure window. When you select an insertion point, the selection is reflected in both, enabling you to verify the insertion position visually as well as hierarchically.

For more information, see Section 11.1.4, "How to Use the Component Palette."

To insert a page element:

1. With a page open, select the Component Palette package, or page from the drop down list. The Component Palette is context sensitive and displays only those options that are relevant to the active file.
2. Do one of the following:
   - Select the insertion point where you want the element to appear on the page in the visual editor or in the Structure window, then click the element in the Palette.
   - Drag the element from the palette to the desired insertion point on the page.
3. Depending on the element, a corresponding insertion dialog appears, prompting you to select or insert a file, or supply tag attributes.

When you insert a page element, JDeveloper generates the source code for the element. When you delete an element, the associated lines from the code are also deleted.

11.1.2.10 How to Set and Modify Web Page Element Properties

The Property Inspector displays the properties of web page elements selected in the visual editor or the Structure window. Use the Property Inspector to set or modify the property values for any element in your web pages. Set property values are marked with a green square. To undo changes, from the main menu select **Edit** > **Undo action**. Use the **Set to Default** button to reset a property with a default value to its original value.

For more information, see Section 11.1.3, "How to Use the Property Inspector."

To set element properties:

1. With a web page open, select an element in the visual editor or the Structure window. The Property Inspector displays the property values for the selected element. If the Property Inspector is not in view choose **View Property Inspector** or use the shortcut **Ctrl+Shift+I.**
2. Scroll until the property you want is visible, then select it with the mouse or the arrow keys. A brief description of the property is displayed at the bottom of the Property Inspector.

   Note: You can also use the Find box at the top of Property Inspector to search for the property.

3. Enter the property value in the right column in one of the following ways:
   - In a text field, type the string value for that property, for example a text value or a number value, then press **Enter.**

- In a value field with a down arrow, click the down arrow and choose a value from the list, then press **Enter.**
- In a value field with an ellipsis (...), click it to display an editor for that property, for example, a color or font selector. Set the values in the property editor, then press **OK**.

To display an editor to set or modify an element's properties:

- Double-click the element.

11.1.2.11 How to Set a Data Source for a Property

As an alternative to working with the Data Control Palette to create databound UI components, you can set ADF bindings for UI components that you display in the visual editor.

Use the Property Inspector to set or remove a data source for an element property. From a Value Binding dialog you can select available data sources defined by the objects or the application ADF binding context that you specify for an EL expression. Note that before you can specify an ADF binding as a data source you must first create the binding.

To databind an element property:

1. With the JSP page open, select an element in the visual editor or Structure window.
2. Scroll until the property for which you wish to specify a data source is visible, then select it with the mouse or the arrow keys.
3. Click the Bind to Data button. An EL expression is displayed in the property value field and an ellipsis button becomes available.
4. Click the ellipsis (...) button to display a Value Binding dialog, and then select the data source.
5. Click **OK**.

> **Tip:** To remove a data source from a JSP element property, toggle the Bind to Data button off.

11.1.2.12 How to Set Properties for Multiple Elements

If you have multiple elements selected, by default the Property Inspector displays all the properties of the selected elements. Click **Union** in the Property Inspector toolbar to toggle between displaying all the properties of the selected elements (union) and displaying only the properties that the selected elements have in common (intersection). Values represented in italic font indicate common properties that have differing value.

To set properties for multiple elements:

Do one of the following:

- Hold down the Ctrl key and select each of the elements.
- To change the list of properties displayed by the Property Inspector, click the Union button in the Property Inspector toolbar:
- S elect and edit the desired property in the Property Inspector. If the value is shown in italic font, the selected elements have differing values. Editing the value of a shared property will cause all selected elements to have the same value.

11.1.2.13 How to Use Basic Commands to Manage Your Elements

Cut, copy, and paste web page elements in the visual editor or Structure window. You can perform these operations between files of the same project or different projects.

To cut one or more elements:

1. Select the page element you wish to cut in the visual editor or the Structure window.
2. Press **Ctrl + X.** Right-click and select **Cut**. You can also choose **Edit** > **Cut** from the main menu.

   The element is removed from the editor and placed into a local clipboard only accessible by JDeveloper not to the system clipboard. If you quit JDeveloper without pasting the element, the cut version of the element will be lost.

   The cut command is the first step in a cut and paste action. You can also delete an element. Deleting an element removes it without changing the contents of your clipboard.

To delete web page elements:

1. Select one or more page elements you wish to delete in the visual editor or the Structure window.
2. Press **Delete** or **Backspace**. You can also right-click and select **Delete**, or choose Edit > **Delete** from the main menu. If the element selected for deletion contains included elements, deleting the element also deletes all its contained elements.

To copy one or more elements

1. Select the page element to copy in the visual editor or the Structure window.
2. Press Ctrl + C. You can also right click and select **Copy**, or choose **Edit** > **Copy** from the main menu.

   In the visual editor you can also:

   - Right-click drag an element to an insertion point, release the mouse, and then choose **Copy Here** from the context menu.
   - Hold down Ctrl and drag a copy of the selected element to an insertion point on the page.

To paste an element:

1. Open the file to paste an element in the visual editor or Structure window.
2. Select the insertion point where you want to paste the component.
3. Press Ctrl + V. You can also Right-click and select **Paste** or choose **Edit** > **Paste.**

To move web page elements:

1. Drag the element(s) from the original position to an insertion point in the Visual Editor or Structure window.
2. Right-click drag the element(s) from the original position to an insertion point in the visual editor or Structure window, and then choose **Move Here** from the context menu.

To Resize HTML Page Elements

1. Go to the Property Inspector under Style Size and select your size preference or return to default which is 100 percent width of the page.
2. Double-click the element, set size properties in the editor dialog, and then click **OK**. You can also Right-click the element, choose Edit Tag, set size properties in the editor dialog, and then click **OK**, or Select the element, and then set size properties in the Property Inspector.

11.1.2.14 How to Work with Data Tables

JSF applications use the dataTable tag to display a data table. You use the Create Data Table Wizard to insert that tag on a JSF page. This wizard also provides rudimentary formatting. Once created, you can further edit the table by setting or changing attribute values. You can also add or delete columns, and add components or objects to columns.

To create and edit a data table:

1. Open a JSF page in the visual editor.
2. In the Component Palette, select JSF from the dropdown menu.
3. Double-click or drag Data Table from the palette. The Create Data Table Wizard opens.
4. Follow the steps in the wizard.
5. To change or set values for attributes not accessed using the wizard:
   - Select the `h:dataTable` component in the Structure window.
   - In the Property Inspector, click in the field next to the attribute to set the value. Use the right-click context sensitive Help for information about the different attributes.

To work with columns in a data table:

- To add a single column, right-click an existing column next to where you want to add the new column, and select either **Insert before h:column > Column** or **Insert after h:column > Column**. A column is added either before or after the selected column. The new column is now selected.

  Note: You can also select the data table in the visual editor or structure window. In the visual editor dropdown menu, select **Insert inside Data Table** > **Column**.

- To add multiple columns.
  - Right-click an existing column next to which you want to add the new columns, and **select DataTable > Insert Columns.**
  - Complete the dialog.
- To reorder the columns, drag and drop the columns in the Structure window or in the visual editor.
- To add a component or other object to a column (for example to display data), right-click the column and select **Insert Inside Column.** Use the menus to select components or other objects to place inside the column. You can then use the Property Inspector to set attribute values for the added components.

> **Note:** You can also select the column in the visual editor or structure window. In the visual editor dropdown menu, select **Insert inside Column** > **Output Text**.

- To delete a column, right-click the column and select **Delete**.

11.1.2.15 How to Work with Panel Grids

JSF applications use the `panelGrid` tag to display an HTML table. You then place other components inside the panel grid. JDeveloper provides the Create PanelGrid Wizard to help you create the grid. Once created, you can further edit the grid by adding, moving, and deleting components in the grid.

To create and edit a panel grid:

1. Open a JSF page in the visual editor.
2. In the Component Palette, select JSF from the drop-down menu.
3. Select Panel Grid. The Create PanelGrid Wizard opens.
4. Complete the wizard.
5. To change attribute values set in the wizard, double-click on the `h:panelGrid` component in the Structure Pane. The properties editor opens. Change any values as needed.
6. To insert a component into the grid, in the Structure Pane, right-click an existing component and elect to place the component either before or after the existing component. If you need to nest components in a cell, you must first place a `panelGroup` tag in the cell. You can then elect to place other components inside the `panelGroup` tag. Note that you cannot add rows to a panel grid. You can only add columns using the Columns attribute. Components are then placed in columns along a row in the order they appear in the Structure window.
7. To reorder the components, drag and drop the columns in the Structure window or in the visual editor.
8. Add a header or footer to the grid.
9. To delete a grid or a component in a grid, right-click the component and select **Delete.**

11.1.2.16 How to Paste Markup Code in JSP and HTML Pages

You can copy and paste source code between files in the same project or different projects. Paste source code without interpretation, for example as sample code, by selecting **No** in the Confirm Markup Insert dialog.

To paste markup code

1. Copy your source code on the local system clipboard.
2. Choose **Edit** > **Paste Special**.

11.1.2.17 How to View and Edit Web Page Head Content

HTML head content such as style definitions and the browser window title are invisible elements on web pages. In the visual editor you can view and edit head section elements.

To view elements in the head section of a page:

With a web page open in the visual editor choose **Design** > **Show** > **Head Content**. For each element of the head section, an icon appears in a bar at the top of the page.

When you select an element in the head section bar, the source code for the element is highlighted in the code editor.

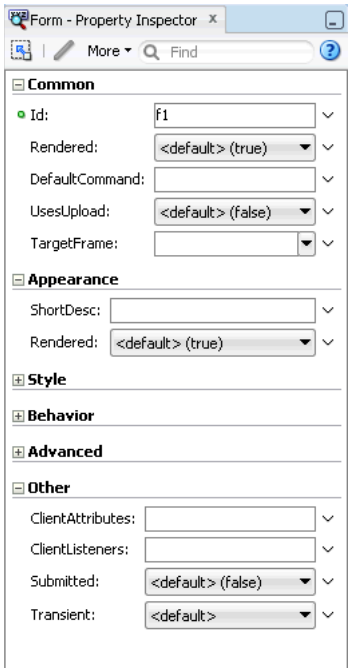
To edit an element in the head section of a page:

1. In an open web page display the head section elements by choosing **Design** > **Show** > **Head Content**.
2. In the visual editor do one of the following:
   - Click an element in the head section bar to select, and set or modify the element properties in the Property Inspector.
   - Right-click the element and choose **Edit Tag** from the context menu to open an editor dialog. To open a cascading style sheet for editing choose **Open** `css/filename.css` from the context menu.

11.1.3 How to Use the Property Inspector

Use the Property Inspector to view and edit the properties of a selected component.

Figure 11–6 Property Inspector



As shown in Figure 11–6, the title bar of the Property Inspector displays the name of the selected component, for example, form or body. The main area of the inspector displays the component properties and their values. If you have selected more than one component in the active tool, the word "Multiple" appears in the title bar, and only the properties shared among the selected components display.

The main area of the Property Inspector displays groups of properties in named sections that you can expand or collapse.

The Property Inspector displays component properties as fields, dropdown lists, or combo boxes. For boolean properties, a checkbox before the property name to indicates its current value. boolean properties that accept EL expressions use either a field or dropdown list, depending on whether the current value is an expression or an explicit value of true or false.

In all properties, an asterisk (*) appears next to a property name indicating that a value is required for that property. To see a description of a property, right-click a property name, field, dropdown list, or combo box to display a popup window. A description of the property appears in a scrollable box under Property Help at the bottom of the popup window. Click + (plus) and - (minus) to toggle the help box display within the popup window. Resize the popup window by dragging the bottom right corner of the window. When you resize the popup window, the new size is used for all subsequent property popup windows that you open until you change the size again.

Table 11–4 Toolbar Icon Features on the Property Inspector

Icon	Name	Description
	Enable/Disable Auto-Extend	Use to toggle on and off the automatic expansion of the Property Inspector to display the full contents when the cursor is over the Inspector. When focus moves to another part of the user interface, the Inspector returns to the default position.
	Bind to ADF Control	When available, click to bind or rebind a property to an ADF data control of your choice.

11.1.3.1 Editing Properties

To edit a property value, enter a new value in a field or select a value from a fixed set of values using a dropdown list. When you edit a property value, a green dot appears next to the property name to indicate that it has been changed from its default setting. Other ways to edit a property value include the following:

- For some properties, click ... at the end of the field or box to use a property editor or browser tool to select and enter a value for the property.
- For some properties, click at the end of the field or box to display a popup window and then choose a command, or choose a property editor or builder tool to select and enter a value for the property.
- For boolean properties with checkboxes, select or deselect the checkbox to change the value.
- For boolean properties that can accept EL expressions, enter an expression in the field or click the down arrow at the end of the field to use a builder tool to enter a value.

11.1.3.2 Writing Custom Property Editors

When you write your own property editors, you can control what the Property Inspector displays. If your property editor supports tags, the Inspector displays those tags in a dropdown list as the fixed set of values. If the property editor does not support tags, the Inspector will query your editor to see whether it supports a custom property editor. If neither are supported, a text area will be displayed for the user to type directly into.

11.1.3.3 Additional Features for Customization Developers

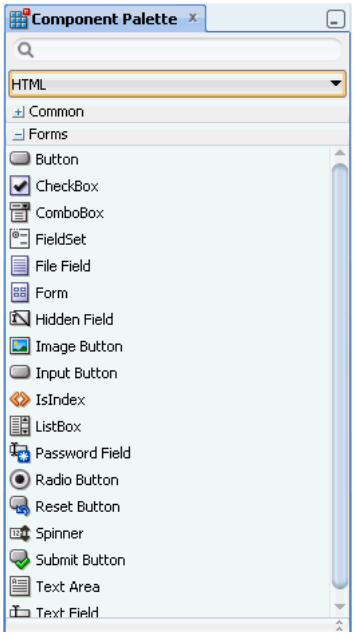
The following additional features are available in Customization Developer role:

- When you edit a property value, an orange dot appears next to the property name. (Property values that were modified in Default role have green dots next to the properties.)
- From the property menu next to a text-only property, choose **Remove Customization** to remove existing customization that was previously applied in the same customization layer context.

11.1.4 How to Use the Component Palette

The Component Palette displays the elements of your component libraries, and lets you assemble a user interface by using simple drag and drop operations. The components available in the palette vary depending on the type of file in the active editor window. For example, if you are editing an HTML file, the palette displays a list of common components, as shown in Figure 11–7.

Figure 11–7 HTML Components in Component Palette



If you are editing a .java file, a completely different set of components display.

11.1.4.1 Using the Component Palette Features

Your file or page components are organized in pages in the Component Palette. Select the palette page you want from the dropdown list at the top of the palette.

To insert a component into a file, open in the active editor, drag the component from the palette to an insertion point in the editor. In some file types you click a component in the palette and then click in the editor to insert the component.

11.1.4.2 Overview of the Component Palette Features

The Component Palette provides the following features:

- To search for a component by name, enter the name or part of the name in the binocular icon field and click the green go arrow. The components matching the

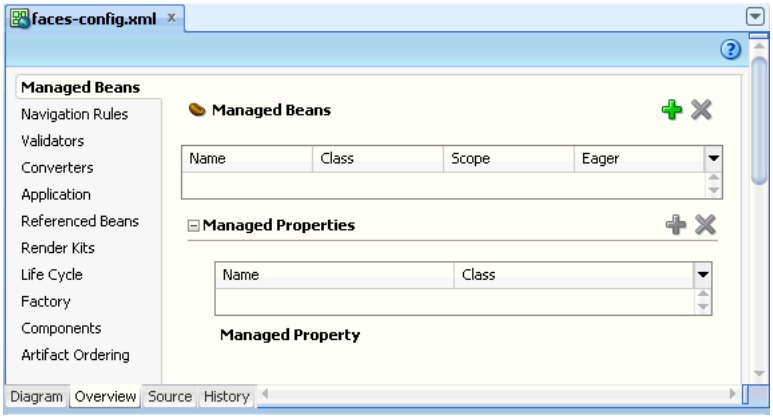
name or part of the name in the palette pages for the active editor will display in the Search Results panel.

- By default components are displayed in a list view (icon plus name). You can change the display to an icon only view. To toggle between views, right-click a component in the palette and choose **Icon View** or **List View**.
- For component libraries with available component help, right-click a component in the palette and choose **Help**.
- To add frequently-used components to a palette page for easy access, right-click a component in the palette and choose **Add to Favorites**. The selected component is added to the Favorites panel of the My Components palette page, which you can select from the palette dropdown list.
- For projects with JSP tag libraries: To change the list of JSP tag libraries available for selection in the palette dropdown list, right-click a component in the palette and choose **Edit Tag Libraries,** then use the dialog to add or remove tag libraries as needed.

11.1.5 How to Use the Overview Editor for JSF Configuration Files

Use the overview editor for JSF configuration files to visually design and edit your JSF application configuration data stored in `faces-config.xml`. Figure 11–8 displays the overview editor.

Figure 11–8 Overview Editor for JSF Configuration File



When you open `faces-config.xml` its contents are displayed in an editor group. When you select the **Overview** tab at the bottom of this group, the overview editor appears.

When the overview editor is open, the Property Inspector displays the metadata child elements for the currently selected element. Use the Property Inspector to manage these. For instance, you use the Property Inspector to set the `<description>` and `<display-name>` child elements.

The overview editor has three sections:

- The left-hand column displays the main JSF configuration elements.
- The top area of the main panel shows child elements for the element selected in the element list on the left.

- The bottom area of the main panel shows child elements for the element selected at the top area.

You can add, delete, or edit your JSF element and child elements using the Overview Editor.

To work with a main JSF configuration element and its immediate child elements:

1. In Application Navigator, open the workspace that contains your JSF application.
2. In the workspace, open the project that contains your JSF pages.
3. In the project, open the WEB-INF node.
4. Under the WEB-INF node, double-click the `faces-config.xml` file to open.
5. At the bottom of the editor, select the Overview tab.
6. Select an element from the element list on the left. The main panel displays corresponding configurable child elements in a table at the top of the main panel.

To add, delete, or edit JSF configuration elements:

- **To add a new child element**. Click **New**. A dialog box opens to create the element. If no new button displays, the child element must be an existing class. You can select the class by clicking **Browse...** . If no browse button appears, or if the entry is not a class name, you can enter a value directly.
- **To delete an existing child element**. Select the element from the table and click **Delete**. The element is removed from the table. If no delete button displays, the entry can be deleted manually.
- **To edit an existing child element**. Select the element from the table and click **Edit**. The Properties panel for the element opens to change the value.

To view, add, delete, or edit child configuration element child element:

- **To view child elements.** Select an element from the element list on the left. The main panel displays. Select an existing child element from a table at the top of the main panel. Allowed child elements display in a table at the bottom of the main panel. If a child element allows child elements, but no children are currently defined, the list area for those children might be hidden. To display the list area and add children, click the show arrow to the left of the area title. To hide the list area, click the hide arrow.
- **To add a new child element.** Click **New**. If no new button displays and the child element must be an existing class, you can select the class by clicking **Browse...** to open the **Class Editor** dialog box. If no browse button appears, or if the entry is not a class name, you can enter a value directly.
- **To edit an existing child element.** Select it from the table and click **Edit.** The Properties panel for the element opens to change the value. If no edit button displays, you can either select a new class (if applicable), or edit the entry To delete an existing child element, select it from the table and click **Delete**.
- T**o delete an existing child element**. Select it from the table and click **Delete**. The element is removed from the table. If no delete button displays, you can delete the entry manually.

11.1.6 How to Plan Your Page Flow With JSF Navigation Diagrams

The JSF Navigation Diagrammer has features to diagram the JSF pages, and the navigation between the pages.

The pages are represented by icons, and the navigation between pages as lines. The navigation is mirrored in navigation cases in the `faces-config.xml` file for the application.

When a JSF navigation diagram is displayed, the Component Palette is also displayed. The JSF Diagram Objects page of the Component Palette shows entries for the elements that can be included on a JSF navigation diagram. To add JSF diagram elements to a JSF navigation diagram, select them from the Component Palette.

11.1.6.1 How to Work with Navigation Diagrams

When you first view the navigation diagram, JDeveloper creates a diagram file to hold diagram details and it maintains this diagram file, and the corresponding JSF configuration file that holds all the settings needed by your application. If you are using versioning or source control, the diagram file is included as well as the configuration file it represents.

To view the navigation diagram for an JSF application:

1. In the Application Navigator, expand your JSF application.
2. Expand the project that contains your application. If you created the application using a template that included JSF, the project name is ViewController.
3. In the project, expand the **WEB-INF** node and double-click to open the JSF configuration file. The default configuration file name is `faces-config.xml`.
4. If the navigation diagram for the application is not displayed, select the Diagram tab below the window.

When you view the Application Navigator using Group by Category (default), a single entry for the JSF configuration file represents both the configuration file and the associated diagram file. If you view all files using Group by Directory, you see separate nodes for the two separate files: the configuration file using the full file name, and the diagram file is shown as the name of the JSF configuration file with the .jsf_ diagram extension.

When you first open the JSF configuration file, the configuration file node displayed in the Application Navigator indicates that there have been changes, even though no changes have yet been made to the JSF configuration file. This is because the node displayed in the Application Navigator represents both the JSF configuration file and the navigation diagram file. So, although the JSF configuration file has not changed, a navigation diagram file has been created. Similarly, if you make changes to a navigation diagram that do not affect the JSF configuration file, such as changing the layout, the node in the Application Navigator indicates that changes have been made.

If you have a large or complex application, the file can be large and loading can take a long time. If you do not want JSF diagram files to be created for your JSF configuration files, choose not to use the diagram as the default editor so that no diagram file will be created unless you specifically request one.

11.1.6.2 How to Plan Page and the Navigation Flows

Use the JSF navigation diagram and the Component Palette to create a diagram representing the pages in your application and the navigation cases between them.

The navigation cases you add to the diagram are automatically added in the JSF configuration file.

To add an element to a JSF Navigation Diagram

1. View the JSF navigation diagram for your project.
2. In the Component Palette, JSF Diagram Objects, Components page, select JSF Page.
3. To add the page to the diagram, click on the diagram in the place where you want the page to appear, or drag JSF Page onto the diagram surface. An icon for the page is displayed on the diagram with a label for the page name. Initially, before you have defined the new JSF page, the icon indicates that the physical page has not been created, as show in Figure 11–9.

Figure 11–9 Icon Showing Page Not Created.

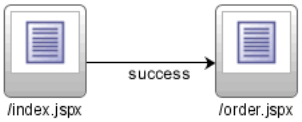
4. To specify the name of the page, click the icon label in the diagram and edit the label. The name requires an initial slash, so that the page can be run. If you remove the slash when you rename the page, it will be reinstated.
5. To define the new page, double-click the icon and use the Create JSF Page dialog. For help using the dialog, click Help. When you have created the page, the icon on the diagram changes to indicate that the physical page has been created as shown in Figure 11–10.
6. Save your changes.

Figure 11–10 Icon Showing Page Has Been Created

To add a JSF navigation case to a JSF navigation diagram:

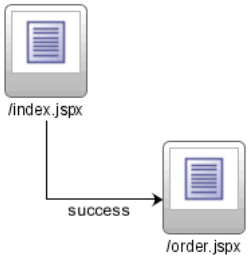
1. View the JSF navigation diagram for your project.
2. If they are not already defined, define the JSF pages that are to be the source <from-view-id> and the destination <to-view-id> for the navigation case you want to create.
3. In the Component Palette, JSF Diagram Objects, Components page, select JSF Navigation Case.
4. On the diagram, click on the icon for the source JSF page, then click on the icon for the destination JSF page to create one navigation case. To draw the navigation case as a straight line between the source and destination pages, click the source page then click the target page as shown in Figure 11–11.

Figure 11–11 Navigation Case Straight Line



To draw the navigation case as a line with angles in it, select either Polyline or Orthogonal in the editor toolbar as shown in Figure 11–12.

Figure 11–12 Navigation Case Angled Lines



5. The navigation case is shown as a solid line on the diagram, and a default <from-outcome> value is shown as the label for the navigation case. To edit the <from-outcome> value, click on the label and enter the new value.
6. A navigation rule is added to the JSF configuration file if there is not one already for the source page, and a navigation case is added for the rule.
7. Save the changes to your JSF navigation diagram and save the changes to the JSF configuration file.

To add a note to a JSF navigation diagram

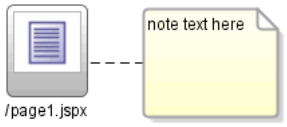
1. View the JSF navigation diagram for your project.
2. In the Component Palette, JSF Diagram Objects, Diagram Annotations page, select **Note.**
3. Click on the diagram surface in the place where you want to add the note. A note is displayed on the diagram with the cursor in place ready for you to enter text.
4. Enter the text and then click outside the note.
5. To select text in the note for editing, click anywhere in the note. To select the note itself, click on the upper right corner. To edit the text, click in the middle of the note.
6. Save the changes to your JSF navigation diagram. Notes appear only on the JSF navigation diagram, not in the JSF application configuration file

To attach a note to an element in a JSF navigation diagram:

1. View the JSF navigation diagram for your project.
2. If the note is not already on the diagram, add the note.
3. In the Component Palette, JSF Diagram Objects, Diagram Annotations page, select **Note Attachment.**

4. Click on the note in the diagram, then click on the element to which you want to attach the note. A dotted line appears, representing the note attachment for the selected page as shown in Figure 11–13.

Figure 11–13 Note Attachment Diagram



5. Save the changes to your JSF navigation diagram. Note attachments appear only on the JSF navigation diagram, not in the JSF configuration file.

To lay out the elements on a JSF navigation diagram automatically

1. Choose the layout style you want from the editor toolbar options. See Table 11–5 for the list of layout styles.
2. Save your changes.

Table 11–5 Navigation Diagram Layout Styles

Icon	Icon Description
	Draws straight lines for navigation cases between page icons.
	Draws lines with angles for navigation cases between page icons.
	Draws lines with right angles for navigation cases between page icons.
	Arranges the icons in a horizontal layout.
	Arranges the icons in a vertical layout. The elements on the diagram are laid out according to the pattern you chose.

To refresh the JSF navigation diagram to reflect changes to the JSF configuration file:

1. Select **Refresh Diagram**. The refresh speed for a diagram scales with the number of nodes in the diagram and the number of connections between the nodes.
2. Save your changes.

11.1.6.3 How to Use the JSF Navigation Diagrammer to Manipulate JSF Pages

You can use the navigation diagrammer to add, edit, rename, and delete JSF pages.

Effect of deleting

The associated web page is no longer visible in the JSF navigation diagram. If you created the file, it is still available from the Web Content folder in the ViewController project in the Application Navigator.

Effect of editing

When you edit web pages manually, JDeveloper does not automatically update either the JSF navigation diagram or the JSF configuration file.

Effect of renaming

If you rename a JSF page on a navigation diagram, this is like removing a page with the original name from the diagram and adding a new one with the new name. If you have created the underlying page, that page remains with its original name in the file system; on the diagram, the page icon changes to the icon that indicates the page does not yet exist.

If you have already created a JSF page and it is displayed on the diagram, if you rename it in the navigator, this is equivalent to removing the original file and creating a new file. The diagram retains the original name, and now displays the page icon that indicates the page does not exist.

Renaming a page on a JSF navigation diagram affects the navigation rules and cases in the JSF configuration file.

To view a web page from a JSF navigation diagram:

Double-click the icon for the web page you want to view. The page opens in the appropriate editor.

11.1.6.4 How to Use the JSF Navigation Diagrammer for JSF Navigation Case

Use navigation diagrammer to delete navigation cases between the pages.

Effect of deleting

The associated `<navigation-case>` is removed from the JSF configuration file.

The associated web page is still visible in the diagram and, if it has been created, is still available from the Web Content folder in the ViewController project in the Application Navigation.

Effect of editing

When you edit the label for the navigation case on the diagram, the associated <navigation-case> is updated in the JSF configuration file.

Once you have created a navigation case in the JSF navigation diagram, you cannot change the destination of the navigation case in the diagram. To change the destination for an existing navigation case in the diagram, delete the existing navigation case and create a new one to the correct destination

If your JSF diagram file is large and would take a long time to open in the JSF navigation diagrammer, you may be asked if you would like to open the JSF configuration file in another editor instead.

To view properties of a navigation case on a JSF navigation diagram:

The navigation cases are displayed on the diagram as solid lines, with the `<from-outcome>` element value displayed as the label.

1. If the Property Inspector is not displayed, open it from the **View** menu.
2. Select the navigation case with properties you want to view. The properties of the navigation case are shown in the Property Inspector.

11.1.6.5 How to Publish a Diagram as a Graphic

Diagrams can be saved as .jpg, .png, .svg or .svgz files for use in documents, or on web pages. Images saved in .jpg format will tend to create the largest files, followed by .svg, .png and .svgz.

To publish a diagram as a graphic:

1. Right-click on the surface of the diagram that you want to publish as a graphic, then choose **Publish Diagram**.

   Or

   Click on the surface of the diagram that you want to publish as a graphic, then choose **Diagram** > **Publish Diagram.**

2. Using the Location drop-down list, select the destination folder for the graphic file.
3. For file name, enter a name for the graphic file, including the appropriate file extension.
4. From the file type drop-down list, select the file type for the graphic file.
5. Click **Save**.

11.1.7 How to Use Code Insight For Faster Web Page Coding

Use Code Insight to speed up your coding tasks by providing available options for you to select while coding to quickly complete or insert elements.

Code Insight provides completion insight and parameter insight. To invoke completion insight, pause after typing the period separator or, in the default keymap, press Ctrl+Space. To invoke parameter insight, pause after typing an opening (the left) parenthesis or, in the default keymap, press Ctrl+Shift+Space. To exit either type of insight at any time, press Esc.

To use Code Insight in a web page in the source editor:

1. Click the Source tab to open the file in the source editor, and place your cursor at the location where you want to add a tag.
2. Enter the < (open angle bracket) and then either pause or press Ctrl + Space (using the default keymapping) to invoke Code Insight. A list of valid elements based on the file is displayed. Narrow the list by typing the first letter of the tag or enter a tag library prefix followed by a colon (i.e., <jsp:).
3. From the list of valid tags, double-click the tag, or highlight the tag and press **Enter**. JDeveloper inserts the selected tag in the file, e.g., `<jsp:include`. There should be no space between the prefix and the tag name.
4. To add an attribute to the tag you inserted, enter a space after the tag name, then either pause or press Ctrl+Space to open a list of valid attributes. Select the tag by double-clicking or highlighting and pressing Enter. For example: `<jsp:include page`.
5. Enter the attribute value. For example: `<jsp:include page="filename.jsp"`.
6. Add other attribute and values as necessary. Use a space between an attribute value and the next attribute. For example: `<select size="4" name="ListBox"></select>`.
7. When finished adding attributes and values, enter the > (close angle bracket). The correct end tag (e.g., </select>) is automatically inserted for you if the **End Tag Completion** feature is enabled. Whether **End Tag Completion** is enabled or

disabled, the correct end tag is always automatically inserted for you when you enter </ (open angle bracket and forward slash characters) to close the tag.

Right-click any tag name in the editor and choose **Select in Structure** to highlight that tag in the Structure window. The Structure window also displays any syntax errors found as you edit. You can double-click an error, element, or attribute to edit it in the source editor.

To enable the **End Tag Completion** feature, choose **Tools** > **Preferences** > **Code Editor** > **JSP/XML/HTML** to open the panel and select the option.

Code Insight is also available in JavaScript and CSS files.

To use Code Insight in a JavaScript file:

- Place the cursor inside any <SCRIPT> tag, then type the open angle-bracket and press Ctrl + Space.

JDeveloper will display a list of possible completions. You can filter the available completions by typing the first character; for example, if you type the letter d, JDeveloper will display completions beginning with D (Date, decodeURI, etc.)

Code Insight will also prompt for completion inside JavaScript-specific XML attributes.

To use Code Insight in a CSS file:

- Place the cursor inside any <STYLE> tag, then type the open angle-bracket and press Ctrl + Space.

JDeveloper will display a list of possible completions at this point in the CSS file. You can filter the available completions by typing the first character of the element for which you are using Code Insight.

11.2 Developing Applications with JavaServer Faces

This section covers JDeveloper support and tools for your user interface development using JavaServer Faces (JSF) technology within the Java EE platform.

JDeveloper provides full support for developing user interfaces with JSF and facelets technology in accordance with the JSF 2.0 specification found at http://jcp.org/aboutJava/communityprocess/final/jsr314/index.html. The JSF content in this section assumes you are using facelets technology for your JSF development.

11.2.1 How to Build Your JSF Application

You can build your application from the ground up using the features provided in JDeveloper. The first thing to do is build a framework or application template for your web pages. Get started quickly using the application templates. Choose from a combination of technologies to include in your application as you build your application with the New Gallery Wizard. The application you choose determines the project folders created and the libraries added to the folders as shown in Table 11–6.

Table 11–6 Web Application Templates

Application	Description
Fusion Web Application (ADF)	Creates a databound ADF web application. This application contains one project for the view and controller components (ADF Faces and ADF Task Flows), and another project for the data model (ADF Business Components).
Java EE Application	Creates a databound web application. This application contains one project for the view and controller components (JSF), and another project for the data model (EJB and JPA entities)
Generic Application	Creates an application with a single project. The project is not preconfigured with JDeveloper technologies and can be customized to include any technologies.

11.2.1.1 How to Build Your Application Framework

Use the wizards to build a customized application framework.

To create a web application and project for a JSF application:

1. From the main menu select **File** > **Menu** > **New** > **General** > **Applications**.
2. Select an application to create.
3. Complete the steps. The project folders, Model and ViewController, are created and listed in the Application Navigator under the new application node. If you chose Generic Application, you only see a Project folder.
4. Double-click the ViewController project to open the Project Properties dialog, and select Dependencies. Make sure the Model project is selected under Project Dependencies.

11.2.1.2 How to Create Your JSF Pages and Related Business Services

Once you have created the framework of your application, get your pages up and running fast with the page building, editing, and modeling tools.

To quickstart your JSF application end to end:

1. Build a web application with the easy wizards. See Section 11.2.1.1, "How to Build Your Application Framework".
2. Create your JSF pages using the New Gallery JSF wizard. See "To create your JSF pages:" on page 11-28.
3. Choose a Business Service. See "Choosing a Business Service" on page 11-29.
4. Create the backing beans for your business services. See Section 11.2.2.2, "How to Work with Managed Beans".
5. Bind the interface components to data. See Section 11.2.2.4, "How to Bind Components to JSF Pages".
6. Add application resources and managed beans to `faces-config.xml`. See Section 11.2.2.12, "How to Configure JSF Applications".
7. Run your JSF pages. See Section 11.2.3, "How to Run and Test JSF Applications".

To create your JSF pages:

1. In the Application Navigator, select your project for the new JSF 2.0 page or document. Note that you can also create you JSF pages from the Navigation Modeler.

2. Choose **File** > **New** to open the New Gallery.
3. In the Categories tree, expand Web Tier and select JSF. From this wizard create a JSF/facelet page or a JSP XML page. Choose ADF Faces page templates or quick start layouts. ADF Faces page templates (.jsf file) define an entire page layout in a page template definition file that allows for reuse and parametization. The quick start layouts are a a pre-defined page layout that automatically inserts and configures the ADF Faces components required to implement the layout look and behavior.

Choosing a Business Service

With JDeveloper you can work with data sources based on Enterprise JavaBeans (EJB) or JavaBeans. You can also use Oracle TopLink to map your Java classes and EJBs to database tables. Web services is available if you don't need to create the backend business service and want to expose existing business services, including EJB components, stored procedures in the database, or other services writing Java and other languages.

None of the application model and backend classes should refer to JSF classes so that the same classes can be used with any type of user interface.

When you want to work with a specific business service, you can open the New Gallery and use the provided wizards and dialogs to create or, in the case of web services, expose the entities in your Model project, as shown in Table 11–7.

Table 11–7 Business Service New Gallery Options

If you want to use...	Then choose this New Gallery option...
Enterprise JavaBeans in the Model project	EJB in the Business Tier category
Oracle TopLink in the Model project	TopLink in the Business Tier category
JavaBeans in the Model project	JavaBeans in the General category
Web services that were created based on legacy code, software components (such as EJB components), or even PL/SQL in the database and make it accessible through HTTP quickly and easily.	Web Services in the Business Tier category

To create a business service:

1. Create a web application and project. See web application options in Table 11–6.
2. In the Application Navigator, under your application node, select the Model project and choose **File** > **New** to open the New Gallery.
3. In the Categories list, expand a node and you will see categories related to your chosen technology scope. Under the Business Tier node, you will see business service options such as ADF Business Components, EJB, Toplink, and Web Services. Choose your business service.

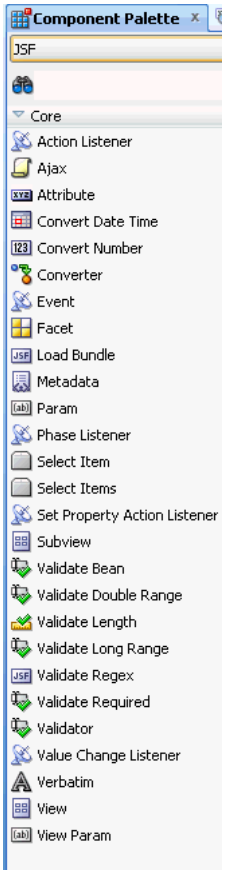
11.2.2 How to Build your JSF Business Component Framework

JDeveloper comes with a Component Palette stocked with standard JSF components that you can easily drag and drop onto your JSF pages. When you create a JSF page the backing beans are created and automatically binded to all of the components you put on the page and to corresponding properties. In addition, there is a Property Inspector and Expression Language feature to assist you.

For localization, resource bundles are automatically added when you add content components to your page. You can manage your resource bundles, or create new resource bundles in the Project Properties feature of your application.

Among the many standard component options provided with JDeveloper, there are validating and converting components that are configurable through the Property Inspector, as well as a Message component to help you set up the error message output for your JSF pages, as shown in Figure 11–14.

Figure 11–14 Core JSF Components Available in Component Palette



JSF Common Components

For complete information about JSF tag libraries, see the tag libraries documentation at: http://download.oracle.com/docs/cd/E17410_01/javaee/6/javaserverfaces/2.0/docs/pdldocs/facelets/index.html.

Table 11–8 Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:actionListener	<f:actionListener [type] [binding] [for] />	Registers an action listener on the UIComponent associated with the closest parent component.
f:ajax	<f:ajax [disabled] [event] [execute] [immediate] [listener] [oneevent] [oneerror] [render] />	Registers an AjaxBehavior instance on one or more UIComponents implementing the ClientBehaviorHolder interface. This tag may be nested within a single component (enabling Ajax for a single component), or it may be "wrapped" around multiple components (enabling Ajax for many components).
f:attribute	<f:attribute [name] [value] />	Adds an attribute to the UIComponent associated with the closest parent UIComponent custom action.
f:convertDateTime	<f:convertDateTime [dateStyle] [locale] [pattern] [timeStyle] [timeZone] [type] [binding] [for]	Registers a DateTimeConverter instance on the UIComponent associated with the closest parent UIComponent custom action.
f:converter	<f:converter [converterID] [binding] [for] />	Registers a named Converter instance on the UIComponent associated with the closest parent UIComponent custom action.

Table 11–8 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:convertNumber	<f:convertNumber [currencyCode] [currencySymbol] [groupingUsed] [integerOnly] [locale] [maxFractionDigits] [minIntegerDigits] [pattern] [type] [binding] [for] />	Register a NumberConverter instance on the UIComponent associated with the closest parent UIComponent custom action.
f:event	<f:event [name] [listener] />	Allows you to install ComponentSystemEventListener instances on a component in a page.
f:facet	<f:facet/>	Registers a named facet on the UIComponent associated with the closest parent UIComponent custom action.
f:loadBundle	<f:loadBundle [basename] [var] />	Loads a resource bundle localized for the Locale of the current view, and expose it as a java.util.Map in the request attributes of the current request under the key specified by the value of the "var" attribute of this tag. The Map must behave such that if a get() call is made for a key that does not exist in the Map, the literal string "KEY" is returned from the Map, where KEY is the key being looked up in the Map, instead of a Missing Resource Exception being thrown. If the Resource Bundle does not exist, a JspException must be thrown.

Table 11–8 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:metadata	<f:metadata/>	Declares the metadata facet for this view. This must be a child of the <f:view>. This tag must reside within the top level XHTML file for the given viewId, not in a template. The implementation must insure that the direct child of the facet is a UIPanel, even if there is only one child of the facet. The implementation must set the id of the UIPanel to be the value of the UIViewRoot.METADATA_FACET_NAME symbolic constant.
f:param	<f:param [binding] [id] [name] [value] [disable] />	Adds a child UIParameter component to the UIComponent associated with the closest parent UIComponent custom action.
f:phaseListener	<f:phaseListener [type] [binding] />	Registers a PhaseListener instance on the UIViewRoot in which this tag is nested.
f:selectItem	<f:selectItem [binding] [id] [itemDescription] [itemDisabled] [itemLabel] [escape] [itemValue] [value] [noSelectionOption] />	Add a child UISelectItem component to the UIComponent associated with the closest parent UIComponent custom action.

Table 11–8 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:selectItems	<f:selectItems [binding] [id] [value] [var] [itemValue] [itemLabel] [itemDescription] [itemDisabled] [itemLabelEscaped] />	Adds a child UISelectItems component to the UIComponent associated with the closed parent UIComponent custom action. When iterating over the select items, toString() must be called on the string rendered attribute values. Version 2 of the specification introduces several new attributes, described below. These are: var, itemValue, itemLabel, itemDescription, itemDisabled, and itemLabelEscaped.
f:setPropertyActionListener	<f:setPropertyActionListener [value] [target] [for] />	Registers an ActionListener instance on the UIComponent associated with the closest parent UIComponent custom action. This actionListener will cause the value given by the "value" attribute to be set into the ValueExpression given by the "target" attribute.
f:subview	<f:subview [binding] [id] [rendered]	This handles the Container action for all JavaServer Faces core and custom component actions used on a nested page via "jsp:include" or any custom action that dynamically includes another page from the same web application, such as JSTL's "c:import"
f:validateBean	<f:validateBean [validationGroups} [disabled] [binding] [for]	This is a validator that delegates the validation of the local value to the Bean Validation API. The validationGroups attribute serves as a filter that instructs the Bean Validation API which contraints to enforce. If there are any constraint violations reported by Bean Validation, the value is considered invalid
f:validateDoubleRange	<f:validateDoubleRange [disabled] [maximum] [minimum] [binding] [for] />	Registers a DoubleRangeValidator instance on the UIComponent associated with the closest parent UIComponent custom action.

Table 11–8 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:validateLength	<f:validateLength [disabled] [maximum] [minimum] [binding] [for] />	registers a LengthValidator instance on the UIComponent associated with the closest parent UIComponent custom action.
:validateRegex	:<validateRegex [disabled] [pattern] [binding] [for] />	This is a validator that uses the pattern attribute to validate the wrapping component. The entire pattern is matched against the String value of the component. If it matches, it's valid.
f:validateRequired	<f:validateRequired [disabled] [binding] [for] />	This is a validator that enforces the presence of a value. It has the same affect as setting the required attribute on a UIInput to true.
f:validator	<f:validator [disabled] [validatorId] [binding] [for] />	Registers a named Validator instance on the UIComponent associated with the closest parent UIComponent custom action.
:valueChangeListener	<:valueChangeListener [type] [binding] />	Registers an ValueChangeListener instance on the UIComponent associated with the closest parent UIComponent custom action.
f:verbatim	<f:verbatim [escape] [rendered] />	Creates and register a child UIOutput component associated with the closest parent UIComponent custom action, which renders nested body content.
f:view	<f:view [locale] [renderKitId] [beforePhase] [afterPhase] />	Container for all JavaServer Faces core and custom component actions used on a page.

Table 11–8 (Cont.) Standard JSF Core Tag Library Supported Elements

Component Tag	Syntax	Description
f:viewParam	<f:viewParam [converter] [converterMessage] [id] [required] [requiredMessage] [validator] [validatorMessage] [value] [valueChangeListener] [maxLength] [for] />	Used inside of the metada facet of a view, this tag causes a UIViewParameter to be attached as metadata for the current view. Because UIViewParameter extends UIInput all of the attributes and nested child content for any UIInput tags are valid on this tag as well.

11.2.2.1 Support for Standard JSF Component Tag Attributes

View and set your component tag attributes in the Property Inspector. When you select an attribute, a brief description of the attribute appears in the text area below the attribute list. Most of the standard JSF component tag attributes accept value binding expressions, #{expression}.

When you add a component to the JSF page, the Property Inspector displays the supported attributes for the component tag grouped in these categories:

- **Common**. Used commonly, such as `id` and `title`. For localization there are language translation attributes such as `lang` and `dir`.
- **Appearance**. Defines how things appear on the page such as links and text.
- **Style**. Used for HTML presentation attributes such as background and font.
- **JavaScript**. Used for JavaScript attributes for associating client-side scripts with events, such as `onclick`, `onkeypress`, and `onmouseover`.

General and Core Attributes

Most standard JSF component tags support the following general and core attributes:

- **binding**. The JSF EL expression that binds a component instance to a property in a bean.
- **id.** The unique identifier of a component. This must be a valid XML name, that is, you cannot use leading numeric values or spaces in the id name.
- **rendered.** A Boolean value that specifies whether a component should be rendered. Default is "true".

HTML Event and Style Attributes

The JSP actions in the tag library support most of the attributes that the HTML 4.01 specification declares for corresponding HTML elements. These attributes are optional and can be set to static values or using any type of JSF EL expression.

Client-side JavaScript event handling attributes supported could include:

```
onblur, onchange, onclick, ondblclick, onfocus, onkeydown,
onkeypress, onkeyup, onload, onmousedown, onmousemove,
onmouseout, onmouseover, onmouseup, onselect, onunload.
```

Style and presentation attributes supported could include background, border, cellpadding, cellspacing, font, margin, style, outline.

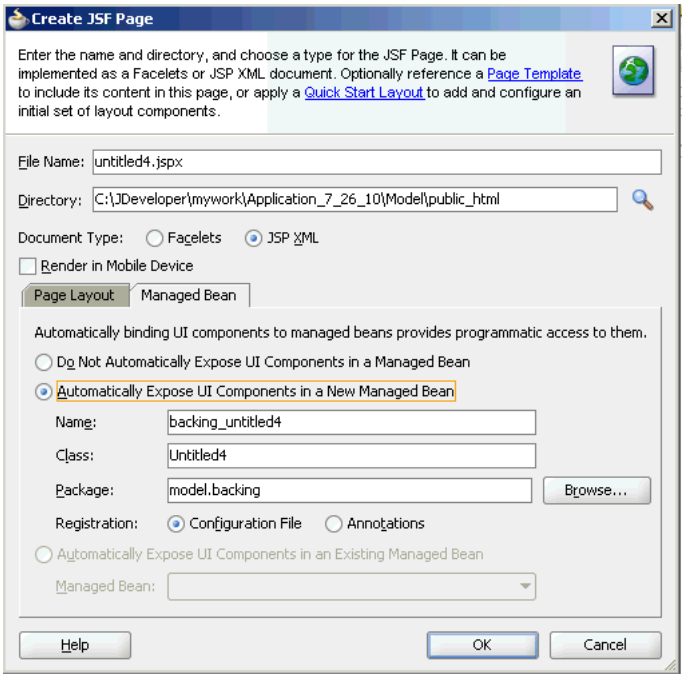
JDeveloper supports the standard JSF tag library contains JSF component tags for all UIComponent + HTML RenderKit Renderer combinations defined in the JavaServer Faces Specification.

All supported attributes are shown in square brackets ([ ]). Default values are provided, where applicable (e.g., [disabled="false"]). An asterisk (*) placed after an attribute means a value is required. Where applicable, supported child and facet components are also shown.

11.2.2.2 How to Work with Managed Beans

Backing beans are managed beans that contain logic and properties for UI components on a JSF page. JDeveloper provides an option to automatically bind components on the Managed Bean tab of the Create JSF page dialog. When this option is selected, a default backing bean is created (or uses a managed bean of your choice) for the page you are creating, and then automatically binds all components you place on the page to a corresponding property in that bean. It also creates the associated accessor methods.

Figure 11–15 Create JSF Dialog - Create Managed Bean Tab



When you create a JSF page using the Create JSF page dialog, you choose between two options for automatic component binding: Automatically Expose UI Components in a New Managed Bean Automatically, or Expose UI Components in an Existing Managed Bean.

When either option on, JDeveloper automatically uses a managed bean for the page you are creating, and automatically binds any component that you drag and drop onto the page to a corresponding property in the bean.

To create managed beans:

1. Create a JSF or JSPX page.
2. Select the Managed Bean tab.
3. Select **Automatically Expose UI Components in a New Managed Bean**. JDeveloper creates a new backing managed bean named the same as the JSF page and places it in the model.backing directory.
4. Add or delete component tags as needed to the JSF page. JDeveloper automatically adds or deletes the properties and corresponding accessor methods in the backing bean. For component tags with attributes that require method binding, use the Property Inspector to enter method binding expressions and select from existing methods in the page backing bean (see procedure below for adding methods to backing beans). You can also enter new method names. JDeveloper creates the new skeleton method in the page backing bean. Add the logic to the method.

To add methods to a managed bean:

1. Open your backing bean in the source editor.
2. From the method binding toolbar on the top of the editor select a component from the Components dropdown menu.
3. From the Events dropdown menu, select the type of method to create. A skeleton method for the component is added.
4. Replace the *// Add event code here...* comment with appropriate business logic.

To create managed beans with the JSF configuration editor

1. In the Application Navigator, double-click on the `faces-config.xml` file. This file is located in the `Web Content/WEB_INF` directory.
2. At the bottom of the window, select the Overview tab. The JSF Configuration Editor window displays.
3. In the element list on the left, select **Managed Beans**.
4. Click **New** to open the Create Managed Bean dialog.
5. Enter the name and fully qualified class path for the bean.
6. Select a scope, check the Generate Java File check box, and click **OK**.
7. This creates a Java file for the managed bean that contains a public constructor method. Manually add all properties and additional methods. The file is named and placed using the fully qualified class name set as the value of "Class". The new file appears within the project Application Sources node in the Application Navigator and within the defined package in the System Navigator.

11.2.2.3 How to Work with Automatic Component Binding

If you create a page and elect to automatically bind components, JDeveloper does the following automatically:

- If you elect to create a backing bean, a JavaBean using the same name as the JSF or JSPX is created, and placed in a the view.backing package. A managed bean entry

is also created in the `faces-config.xml` file for the backing bean. By default, the managed bean name is backing_<page_name> and the bean uses the request scope.

- On the newly created or selected bean, a property and accessor method is added for each component tag you place on the page.
- The component tag is binded to the property using an EL expression as the value for its binding attribute. Because JDeveloper automatically places a form component on a JSF or JSPX page on creation, properties and accessor methods for the form component are automatically created.
- Properties and methods are deleted when you delete components from the page.

11.2.2.4 How to Bind Components to JSF Pages

You can choose to let JDeveloper create and configure a default backing bean or use an existing managed bean for the page you are creating. When automatic component binding is turned on, JDeveloper controls the binding attribute of components that you drop onto the page. Choose from the component binding options when you are creating your new page in the New Gallery as shown in Table 11–9.

Table 11–9 Component Binding Options

If you want to...	Then choose...
Use a default managed bean for a JSF page	**Automatically Expose UI Components in a New Managed Bean**. Accept the default names, or enter names of your choice.
Use an existing managed bean of your choice for a JSF page	**Automatically Expose UI Components in an Existing Managed Bean**. Then select a managed bean from the dropdown list.

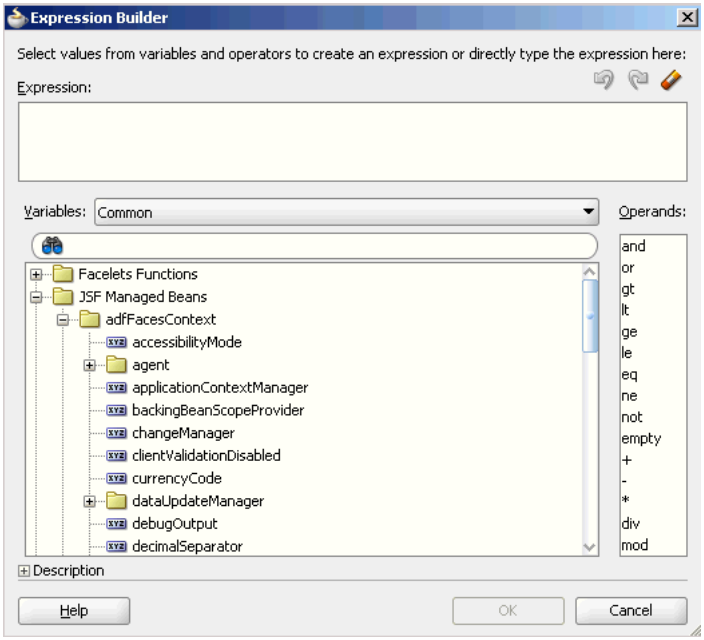
11.2.2.5 How to Bind Components with EL Expressions

JavaServer Faces provides an expression language (JSF EL) that can be used in JSF pages to access the JavaBeans components in your page bean and in other beans in your web application, including the session and the application beans.

To bind any property of a component, add the component to a page and then select the component and create the bindings from the Property Inspector.

You can use the Expression Builder dialog box to choose which JavaBeans property the component property is to be bound to and write your EL Expressions using the tools, as shown in Figure 11–16.

Figure 11–16 Expression Builder Dialog



The JSF expression language syntax uses the delimiters #{}. An expression can be a value-binding expression for binding UI components, or their values to external data sources, or a method-binding expression for referencing backing bean methods.

The syntax supported for a JSF value binding expression is for the most part the same as the syntax defined in the JavaServer Pages Specification (v 2.0), with the following exceptions:

- The expression delimiters for a value binding expression are #{ and } instead of ${and }.
- Value binding expressions do not support JSP expression language functions.

Examples of valid value binding expressions include:

- `#{Page1.name}`
- `#{Foo.bar}`
- `#{Foo[bar]}`
- `#{Foo["bar"]}`
- `#{Foo[3]}`
- `#{Foo[3].bar}`
- `#{Foo.bar[3]}`
- `#{Customer.status == 'VIP'}`
- `#{(Page1.City.farenheitTemp - 32) * 5 / 9}`
- `Reporting Period: #{Report.fromDate} to #{Report.toDate}`

Method binding expressions must use one of the following patterns:

- `#{expression.value}`
- `#{expression[value]}`

Expression language provides the following operators, in addition to the . and []operators:

- Arithmetic: +, - (binary), *, / and div, % and mod, - (unary)
- Logical: and, &&, or, ||, not, !
- Relational: ==, eq, !=, ne, <, lt, >, gt, , ge, >=, le. Comparisons can be made against other values, or against boolean, string, integer, or floating point literals.

  Empty: The empty operator is a prefix operation that can be used to determine whether a value is null or empty.
- Conditional: A ? B : C. Evaluate B or C, depending on the result of the evaluation of A.

To construct an EL expression that uses JSF technology:

1. Open a JSP page in the visual editor.
2. Select the component attribute to bind.
3. In the Property Inspector, select the attribute name.
4. In the attribute action dialog select Expression Builder.
5. Build your expressions and click **OK**. You can edit your EL expressions from that component field in the Property Inspector. Click inside the field to see long expressions. Click Ctrl+Space to invoke Code Insight from the Property Inspector field. You can also add or edit EL expressions by slow-clicking the component and clicking Expression Builder.

11.2.2.6 How to Use Automatic Component Binding for Components that Allow Method Binding

Automatic component binding in a page affects how you enter method binding expressions for the attributes of command and input components such as

- `action`
- `actionListener`
- `launchListener`
- `returnListener`
- `valueChangeListener`
- `validator`

Use the Expression Builder dialog box shown in Figure 11–16 to choose the component property that will be bound.

When Automatic Component Binding is Off

When automatic component binding is turned off, you have to select an existing managed bean or create a new backing bean as you enter method binding expressions for component attributes. If you create a new backing bean, a managed bean is configured in application `faces-config.xml`.

When Automatic Component Binding is On

When automatic component binding is turned on, you do not have to select a managed bean. As you enter method binding expressions for component attributes, you can select from existing methods in the bean, or if you enter new method names,

JDeveloper automatically creates the new skeleton methods. You then add the logic to the method.

In addition, when you edit a Java file that is a backing bean, a method binding toolbar appears in the source editor for you to bind appropriate methods to selected components in the page.

Suppose you created a JSF page with the file name `myfile.jsp`. If you let JDeveloper automatically create a default managed bean, then JDeveloper creates the backing bean as `view.backing.Myfile.java`, and places it in the `\src` directory of the ViewController project. The backing bean is configured as a managed bean in the application resources file (`faces-config.xml`), and the default managed bean name is `backing_myfile`.

When automatic component binding is turned on, any component that you insert in the page is automatically bound (via its binding attribute) to a property in the backing bean, as shown in Example 11–1 and Example 11–2.

Example 11–1 JSF Page (myfile.jsf) Using Default Managed Bean

```
...
<h:form binding="#{backing_myfile.form1}">
   <h:inputText binding="#{backing_myfile.inputText1}"/>
   <h:commandButton value="button0"
      binding="#{backing_myfile.commandButton1}"
      action="#{backing_myfile.commandButton_action}"/>
...
</h:form>
...
```

Example 11–2 Default backing bean Java file: Myfile.java

```
package view.backing;
import javax.faces.component.html.HtmlForm
import javax.faces.component.html.HtmlCommandButton
import javax.faces.component.html.HtmlInputText;


public class Myfile
{
 private HtmlForm form1;
 public void setForm1(HtmlForm form1)
{
this.form1 = form1;
}
 public HtmlForm getForm1()
}
return form1;
}
private HtmlInputText inputText1;
public void setInputText1(HtmlInputText inputText1)
{
public HtmlInputText getInputText1()
{
return inputText1;
}
private HtmlCommandButton commandButton1;
public void setCommandButton1(HtmlCommandButton commandButton1)
{
this.commandButton1 = commandButton1;
}
```

```
return commandButton1;
}
public String commandButton_action()
{
// Add event code here...
return null;
}
}
```

Application resources file: `faces-config.xml`

```
...
<managed-bean>
 <managed-bean-name>backing_myfile</managed-bean-name>
<managed-bean-class>view.backing.Myfile</managed-bean-class>
 <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
...
```

When editing a JSF page in the visual editor, you can turn off or turn on the automatic bind option or change the managed bean selection

- If automatic bind is on and you change the managed bean selection, all existing and new component bindings are switched to the new bean.
- If you turn automatic bind off, nothing changes in the existing component bindings in the page.
- If you turn automatic bind on, all new and existing component bindings are bound to the chosen managed bean.

To turn off or on automatic component binding

1. Open the JSF page in the visual editor.
2. Choose **Design > Page Properties**.
3. Click **Component Binding**.
4. Uncheck or check the **Auto Bind** option.

To select a managed bean for automatic component binding in a JSF page

1. Open the JSF page in the visual editor.
2. Choose **Design > Page Properties**.
3. Make sure the **Auto Bind** option is checked.
4. Click the drop-down arrow and select an existing managed bean, or click **New...** to define a new managed bean. All existing bound components and any new components that you insert are bound to the selected managed bean.

To value bind a component to a property:

1. In the visual editor, select the component.
2. In the Property Inspector, click the dropdown menu in an appropriate field. and choose Expression Builder.
3. Enter an EL Expression that binds to a property on a bean or a value in a resource bundle.

To manually bind component instances to properties

1. In the visual editor, select the component.

2. In the Property Inspector, click the down arrow next to the Binding attribute. The Binding dialog displays.
3. Select a managed bean or click **New...** to create a new one.
4. Select an existing property using the dropdown menu, or click **New...** next to Property to add a new property name.
5. When you are finished click **OK.** If you created a new property, it is inserted as accessor method code in the bean of your choice.

To bind to an existing method with auto component binding on:

1. In the visual editor, select the component.To bind to an existing method using auto component binding, the method must already exist on the backing bean associated with the JSF page.
2. In the Property Inspector, click the column next to the attribute that accepts method binding.
3. Click the dropdown menu and select a method name. Only methods on the backing bean with the proper signature are available for selection.

To bind to a new default method with auto component binding on:

1. Open the associated backing bean.
2. In the source editor, use the method binding toolbar to select the component from the Component dropdown menu.
3. From the Events dropdown menu, select the appropriate attribute. A default method at the bottom of the page is inserted. The cursor is placed at the new method. The binding expression in the JSF page is also created.
4. In the source editor, enter the code for the method.

To bind to a new method with auto component binding on:

1. In the visual editor, select the component.
2. In the Property Inspector, click the column next to the attribute that accepts method binding.
3. Enter the method name, for example: `myMethod`. Note that because the Action attribute can take either a string or a method, you must include the brackets for an action method, for example: `myAction()`. For other methods, the brackets should be omitted. A skeleton method in the associated backing bean, and the binding code in the JSF page is created.
4. In the source editor, enter the code for the method.

To bind to a method with auto component binding off:

1. In the visual editor, select the component. In the Property Inspector, click the dropdown menu next to the attribute that accepts method binding.
2. Select a managed bean or click **New...** to create a new managed bean.
3. Select an existing method using the dropdown menu or click **New...** next to **Method** to add a new method name.
4. Click **OK**. The binding code in the JSF page is created. If you created a new method, a default method code is automatically inserted into your backing bean.
5. Open the bean in the source editor and enter the code for the method.

11.2.2.7 How to Use Localized Resource Bundles in JSF

All of the content you build in your JSF application components is stored in resource Bundles. You can add or remove resource bundles easily from your application in the Default Project Properties dialog.

During development right-click your component to select text resources. The resource bundles available for the project are displayed. Select the bundles to make available for the project you are working on. New text is stored in the resource bundle you select.

You can also assign a key value string to uniquely identify the text object in the resource bundle. By default the name, or a part of the name you enter for display value is used. This value is used by translators to correlate your base content with its localized partner. Existing content strings you have previously added to resource bundles are available and displayed when you are adding new content. Reusing existing content strings optimizes localization efforts, ensuring you don't add new content strings with unique identifiers when a duplicate string with a different identifier already exists. Recycling content strings across your project and application using resource bundles reduces translation efforts and costs.

To add resource bundles in your JSF application:

Add a Resource bundle to your project by going to **Application > Project Properties > Resource Bundle > Bundle Search** . Find your project resource bundle then click to add it to your project.

In your JSF page, you can reference a resource bundle string from any component tag attribute that accepts value binding expressions, e.g., `#{bundle.key}`.

To use localized resource bundles in JSF:

1. Create resource bundles containing the key-value pairs for your localized message and data stings. Place the localized bundles in the application's classpath.
2. In the Application Navigator, double-click `faces-config.xml` to open it in the JSF Configuration Editor. Switch to the Overview, if necessary.
3. Click Application, then click the forward arrow to expand **Locale Config.**
4. Under **Locale Config,** enter a value for **Default Locale**. In **Supported Locale**, click **New** to add an ISO locale identifier for a supported locale. You can add more than one supported locale.
5. Open your JSF page in the visual editor.
6. In the Component Palette, select **JSF Core** from the dropdown list, then drag and drop **LoadBundle** to the page. A dialog appears for you to enter the base name of the resource bundle, and any name for the map variable that will be used in request scope.

Example 11–3 Resource Bundle Code Sample

In the `faces-config.xml`:

```
<faces-config>
  <application>
    <locale-config>
     <default-locale>en</default-locale>
     <supported-locale>en-us</supported-locale>
     <supported-locale>fr</supported-locale>
      <supported-locale>es</supported-locale>
    </locale-config>
  </application>
```

```
...
</faces-config>
```

In the JSF page:

...

```
<f:loadBundle basename="model.login.ApplicationMessages" var="loginBundle"/>
<f:view>
 <html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"/>
    <title>Sample Application</title>
    <link href="css/mycompany.css" rel="stylesheet" media="screen"/>
  </head>
 <body>
 <H2><h:outputText value="#{loginBundle.someHeadLabel}" /></H2>
   <h:form id="loginForm">
   <h:outputText value="#{loginBundle.useridLabel}" />
    <h:inputText id="userid" value="#{login.userid}"
                     required="true" size="15">
   <f:validateLength minimum="4" maximum="7"/>
  </h:inputText>
  <h:commandButton value="#{loginBundle.loginLabel}
 action="someBean.someMethod"} />
...
   </h:form>
  </body>
 </html>
</f:view>
```

11.2.2.8 How to Work with Facets

Many components use facets, and when you use wizards to create complex components (such as a table or panel), output tags are often automatically created and inserted into the facets. You can manually edit these components or add other components to facets. You can also add or delete facets using a context menu in the Structure window.

To work with facets:

1. In the Structure window, expand the parent tag (such as h:dataTable) by clicking the plus sign to the left of the tag. A facet folder displays at the bottom of the tree.
2. Expand the facet folder by clicking the + icon. All facet folders pertaining to that parent display.
3. To edit a component within a facet folder:
   - Expand the folder and select the component.
   - Use the Property Inspector to edit attribute values.
4. To add a component to a facet:
   - Right-click the folder.
   - Select **Insert inside <facet-name>**.
   - Use the resulting menus to select the appropriate object.
   - Use the Property Inspector to set attribute values.

11.2.2.9 How to Build JSF Views with Facelets

Facelets removes the need to write custom tags for JSF components because the technology uses JSF custom components natively. You need very little special coding to bridge JSF and facelets. You can use JSF components directly within the facelets templating language. Facelets allows you to define component assemblies that can be included directly into a page or can easily be added to a facelet tag library. Facelets also allows you to define site templates (and smaller templates). You can also use facelets inside of a custom JSF component because the facelets API provides an interface that is easily integrated with.

Facelets technology offers the following features:

- Reduces UI development and deployment time.
- Faster compiliation time.
- Compile time validation.
- High performance rendering.
- Functional extensibility of components and server-side technologies through customization.
- Support for code reuse through templating and composite components.

Facelets Tag Libraries

JSF uses various tags to express UI components in a web page. Facelets uses the XML namespace declarations to support the JSF tag library mechanism. All of these libraries are included in JDeveloper.

Table 11–10 Facelets Tag Libraries Included with JDeveloper

Tag Library	URI	prefix	Example	Contains
JSF UI Tag Library	http://java.sun.com/javaee/javaserverfaces/reference/api/	ui:	ui:component ui:insert	This tag Library is used for templating
JSF HTML Tag Library	http://java.sun.com/javaee/javaserverfaces/reference/api/	h:	h.head h.body h.outputText h.inputText	This tag library contains JavaServer Faces component tags for all UIComponent + HTML RenderKit Renderer combinations defined in the JavaServer Faces 2.0 Specification.
JSF Core Tag Library	http://java.sun.com/products/jsp/jstl/reference/docs/index.html	f:	f:actionListener f:attribute	This tag library contains tags for JavaServer Faces custom actions that are independent of any particular RenderKit.
JSTL Functions Library	http://java.sun.com/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.1 Functions Tag Library

Facelets support EL (expression language) based on the unified EL syntax defined by JSP 2.1. EL expressions are used to bind UI component objects or values or

managed-bean methods or managed-bean properties. Note that for Unified EL in Facelets there is no difference between ${} and #{}.

To create a facelet:

1. Choose **File > New > New Gallery > Web Tier > JSF/Facelets > Page**.
2. Enter the file name and path for your facelet and click O**K**.

What The Facelet Wizard Does

When you create a facelet the necessary classpath and deployment files are modified to use facelet technology in the following ways:

Example 11–4 Facelet Code Added to Your faces-config.xml

```
<application>
<view-handler> .com.sun.Facelets.FaceletViewHandler</view-handler>
&lt/application>
```

This is added to ensure you view your facelets correctly, and not with the default mappings.

Example 11–5 Facelet Code Added to Your Web.xml

```
<context-param>
<param-name> Facelets.VIEW_MAPPINGS </param-name>
<param-value> *.xhtml</param-value>
</context-param>
```

Example 11–6 Facelet Code Added or ADF

```
<context-param>
<param-name&g;torg.apache.myfaces.trinidad.FACELETS_VIEW_MAPPINGS</param-name>
<param-value>*.xhtml</param-value>
 </context-param>
```

This is added to ensure you view your Facelets correctly, and not with the default JSP mappings. The facelets JAR, `jsf-Facelets.jar` is added to your classpath via the facelets runtime library.

11.2.2.10 How to Convert and Validate JSF Input Data

JDeveloper provides a variety of tools and components to make converting and validating your JSF input data easier. There is a converter component to register a named converter instance, a convert number, and convert date and time all at your fingertips in the Component Palette.

You can configure your converter and validator properties in the Overview editor for your `faces-config.xml` file.

To register a JSF standard converter on a component using a supplied tag:

1. In the visual editor, select the component to register a standard converter.
2. In the Component Palette, select **JSF Core** from the dropdown list, then click a standard converter. (e.g., **convertDateTime**).
3. In the Property Inspector, set the attributes for the converter.

Example 11–7 Registered Standard Converter Code Sample

```
<h:inputText id="hiredate" value="#{employee.hireDate}"
  <f:convertDateTime dateStyle="full"/>
  <f:convertDateTime dateStyle="full"/>
 </h:inputText>
```

To register a JSF standard converter that does not have its own tag:

1. In the visual editor, select the component on which you wish to register a standard converter.
2. In the Component Palette, select **JSF Core** from the dropdown list, then click Converter. A dialog appears for you to enter the converter registered ID.
3. Select a converter ID from the dropdown list (e.g., j**avax.faces.Integer**). When done, click **OK**. This inserts the `f:converter` tag in the page. Instead of using the `f:converter` tag, you can use the Property Inspector to enter the converter ID on the component's converter attribute.

Example 11–8 Registered Standard Converter Using a f:converter Tag within a Component Code Sample

```
<h:inputText id="age" ...>
 <f:converter converterId="javax.faces.Integer" />
</h:inputText>
```

Example 11–9 Registered Standard Converter Using a f:converter Attribute within a Component Code Sample

```
<h:inputText id="age" converter="javax.faces.Integer" />
```

To register a JSF standard validator on a component using a standard tag:

1. In the visual editor, select the input component to register a standard validator.
2. In the Component Palette, select JSF Core from the dropdown list, then click the standard validator of your choice (e.g., **ValidateLength**).
3. In the Property Inspector, set the attributes for the validator. You can register more than one validator on a component. JSF calls the validators in the order they are added to a component.

Example 11–10 Registered Standard Validator Using a Supplied Tag Code Sample

```
<h:inputText id="zip" value="#{employee.zipCode}">
  <f:validateLength minimum="5" maximum="9"/>
</h:inputText>

<h:inputText id="bonus" value="#{employee.bonus}">
  <f:validateLongRange minimum="#{MyBean.miminum}"/>
</h:inputText>
```

To display a message next to the component that generated the conversion or validation error:

1. Open your page in the visual editor.
2. Use the Property Inspector to assign a unique ID to the component to show a message.
3. In the Component Palette, select **JSF** from the dropdown list, then drag and drop Message to the page and position it next to the component to show the message. A dialog appears to enter the unique ID.

4. Enter the ID and click **OK**.
5. In the Property Inspector, set the attributes for the message tag.

Example 11–11 Message Display Next To Component that Generated Conversion or Validation Error Code Sample

```
<h:form>
 <h:inputText id="zip" value="#{employee.zipCode}">
  <f:validateLength minimum="5" maximum="9"/>
  </h:inputText>
  <h:message for="zip"/>
  </h:panelGrid>
 <h:commandButton value="Submit" />
</h:form>
```

To register a custom converter or validator in the JSF application configuration file:

1. In the Application Navigator, double-click the application's `faces-config.xml` file to open it in the JSF Configuration Editor. In the editor, click the Overview tab.
2. In the Overview page of the configuration editor, click **Converters** or **Validators**, then click **New.** The Create Converter or Create Validator dialog appears to enter an identifier and a fully qualified class name. For a custom converter, you can register it under an identifier or a fully qualified class name for a specific data type.
3. Enter the required information. Click **OK**.
4. (Optional) To add attributes or properties, click **New** next to the Attributes or Properties panel. If you don't see **New**, expand the panel by clicking the forward arrow. The Create Attribute or Create Property dialog appears for you to specify generic attributes or JavaBeans properties that may be configured on the custom converter or validator.

To edit a custom converter or validator configuration in an application:

1. In the Application Navigator, double-click the application `faces-config.xml` file to open it in the JSF Configuration Editor. In the editor, click the **Overview** tab.
2. In the Overview page of the configuration editor, click **Converters** or **Validators**. Select a converter or validator from the displayed list, then click **Edit.** The converter or validator properties dialog appears.
3. Enter the necessary changes.

To delete a custom converter or validator in the JSF application configuration file:

1. In the Application Navigator, double-click the application `faces-config.xml` file to open it in the JSF Configuration Editor. In the editor, click the **Overview** tab.
2. In the Overview page of the configuration editor, click **Converters** or **Validators**. Select a converter or validator definition from the displayed list, then click **Delete**. The converter or validator definition is removed.

To register a custom converter on a component using a converter identifier:

1. In the visual editor, select the component to register a custom converter.
2. In the Component Palette, select the JSF Core page, then click **Converter**. A dialog appears to enter the custom converter ID as registered in the application.

3. Select a registered converter identifier from the dropdown list. Only implementations of the converter interface are available. Click **OK**. This inserts the `f:converter` tag. You can use the Property Inspector to enter the registered converter ID.

Example 11–12 Custom Converter with the f:converter tag, Code Sample

```
<h:inputText id="memberNumber" ... >
 <f:converter converterId="customConverter"/>
</h:inputText>
```

Example 11–13 Custom Converter using the Converter attribute, Code Sample

```
<h:inputText id="memberNumber" converter="customConverter"/>
```

To register a custom converter on a component using a value binding expression:

1. In the visual editor, select the component to register.
2. In the Property Inspector, select the converter property, then click the dropdown arrow and choose Expression Builder.
3. Use the Expression Builder to enter a EL expression. Instead of using the converter property, you can add the `f:converter` tag to the component. Use the Expression Builder to enter a value binding expression. The bean property must be an object of a class that implements the converter interface.

Example 11–14 Custom Converter Instance Using a Value Binding Expression, Code Sample.

```
<h:inputText id="age" converter="#{someBean.someProperty}" />
```

To register a custom validator instance on a component:

1. In the visual editor, select the input component to use.
2. In the Component Palette, select JSF Core or ADF Faces Core page from the dropdown list, and then click the Validator component.
3. In the Property Inspector, select a registered validator identifier from the dropdown list, or enter a binding expression. Click **OK**.

Example 11–15 Registered Custom Validator Instance, Code Sample

```
<h:inputText id="name"
    value="#{MyBean.name}"
    size="10" ... >
  <f:validator validatorId="customValidator" />
  <f:attribute name="someName" value="someValue" />
</h:inputText>
```

To bind a component to a new validator method:

1. In the visual editor, double-click the input component . The Bind Validator Property dialog displays.
2. From the **Managed Bean** dropdown list, select a managed bean or click **New**... to create a new one.
3. Enter a new method name in **Method** or accept the default name.

4. Click **OK**. The default validator method code is inserted in the backing bean, and the backing bean.java file opens in the source editor. The cursor is placed at the new method.
5. In the source editor, enter the code for the validator method.

Example 11–16 Component Binded to a New Validator Method, Code Sample

JSF page with automatic component binding off:

```
<h:selectOneMenu validator="#{nonauto.validatename1}">
  <f:selectItems value=""/
</h:selectOneMenu>
```

Default validator method code:

```
...
public void validatename1(FacesContext facesContext, UIComponent uiComponent,
Object object)
{
// Add event code here...
}
...
```

JSF page with automatic component binding on:

```
<h:selectOneMenu binding="#{backing_auto.selectOneMenu1}"
              validator="#{backing_auto.selectOneMenu_validator}">
 <f:selectItems value="" binding="#{backing_auto.selectItems2}"/>
</h:selectOneMenu>
```

Default validator method code:

```
...
public void selectOneMenu_validator(FacesContext facesContext, UIComponent
uiComponent, Object object)
{
// Add event code here...
}
...
```

JSF Standard Converters and Validator Tags and Syntax

All of the attributes supported by JDeveloper are shown in Table 11–11 and Table 11–12. Attributes in square brackets ([ ]) are not required. All accepted, predefined attribute values are separated with vertical bars (|); the default value is in boldface. For attributes that do not have a fixed set of accepted values, the values are shown in italics.

Table 11–11 JSF Standard Converter Tags

Tag	Syntax
f:convertDateTime	<f:convertDateTime [dateStyle="default | short | medium | long | full"] [timeStyle="default | short | medium | long | full"] [pattern="pattern"] [type="time | date | both"] [locale="locale"] [timezone="timezone"] />
f:convertNumber	<f:convertNumber [pattern="pattern"] [minIntegerDigits="min"] [maxIntegerDigits="max"] [minFractionDigits="min"] [maxFractionDigits="max"] [groupingUsed="true | false"] [integerOnly="true | false"] [type="number | currency | percent"] [currencyCode="currencyCode"] [currencySymbol="currencySymbol"] [locale="locale"]

Table 11–12 JSF Standard Validator Tags

Tag	Syntax
f:validateDoubleRange	<f:validateDoubleRange [maximum="max] [minimum="min"] />
f:validateLength	<f:validateLength [maximum="max"] [minimum="min"] />
f:validateLongRange	<f:validateLongRange [maximum='max"] [minimum="min"] />

11.2.2.11 How to Display Error Messages

Create and define error messages using the Message component and the Property Inspector to define the attributes.

To display one error message next to a component that generated an error:

1. Open your JSF page in the visual editor.
2. Assign a unique ID to the component to show a message. You can use the Property Inspector to do this.
3. In the Component Palette, select **JSF** from the dropdown list, then drag and drop **Message** to the page and position it next to the component for which the message is to be shown. A dialog appears for you to enter the ID of the component for which to display a message.
4. Click the column next to **For*** and type the component ID. Then click **OK**.
5. In the Property Inspector, set the attributes for the message tag.

Example 11–17 Error Message Next To A Component, Code Sample

```
<h:panelGrid columns="3">
  <h:outputLabel for="enum" value="Enter employee number: "/>
  <h:inputText id="enum" converter="javax.faces.Long" >
    <f:validateLength minimum="5" maximum="9"/>
  </h:inputText>
  <h:commandButton value="submit"/>
  <h:message for="enum"/>
</h:panelGrid>
```

> **Tip:** To enable a component detail message to appear as a tooltip during runtime, set the message tag tooltip attribute to true. The tag showSummary and showDetail attributes must also be set to true. If you are using ADF data controls to create JSF forms and tables, the h:messages tag is automatically added, which displays all error messages by default. You don't have to add individual h:message tags manually.

To display all error messages generated in a page:

1. Open your JSF page in the visual editor.
2. In the Component Palette, select **JSF** from the dropdown list, then drag and drop Messages to the page and position it at the top of the page.
3. In the Property Inspector, set the attributes for the **Messages** tag.

Example 11–18 Display All Error Message, Code Sample

```
<h:form>
 <h:messages globalOnly="true" layout="table"/>
  ...
</h:form>
```

> **Tip:** Set the globalOnly attribute to true if you want to display only global messages which are not associated with components. If you're using ADF data controls to create JSF forms and tables, JDeveloper automatically adds the h:messages tag for you. You don't have to add the tag manually.

To replace the standard message texts in JSF:

1. Create a property resource bundle containing the key-value pairs for the replacement texts, and place this bundle in the application' classpath.

2. In the Application Navigator, double-click `faces-config.xml` to open it in the JSF Configuration Editor. Go to the **Overview** mode.
3. Click **Application.**
4. In **Message Bundle,** add the fully qualified path to the message resource bundle, e.g., `model.login.Resources`.
5. In your JSF page, use the `h:message` tag to display one error message, or `h:messages` tag to display all error messages. JSF first looks for messages in any registered resource bundle before looking into the JSF standard bundle. This lets you can override any JSF standard message by using the appropriate key in your resource bundle. For a list of messages see the JSF API `javax/faces/Messages.properties`.

To add information about a form field to which a message refers:

1. Create a `PhaseListener` implementation that retrieves and adds a generic attribute to a message.
2. In the Application Navigator, double-click `faces-config.xml` to open it in the JSF Configuration Editor. Switch to the **Overview** mode, if necessary.
3. Click **Life Cycle**, then click **New** to add a custom phase listener.
4. In **Create Phase Listener,** enter the fully qualified path to the phase listener implementation or click **Browse...** to select one.
5. Open your JSF page and locate the input component of your choice.
6. In the Component Palette, select **JSF Core** from the dropdown list, then drag and drop **Attribute** to the input component. A dialog appears for you to enter the required generic attribute information.

To change the appearance of error messages in a JSF page:

1. Open the JSF page of your choice in the visual editor.
2. Link a CSS stylesheet to your page.
3. Select the h:message or h:messages component.
4. In the Property Inspector, set the CSS class that you want to apply to a particular type of message. For example, if you want messages with a severity level of "ERROR" to use a particular stylesheet, set the `ErrorClass` attribute to the name of a style class defined in your CSS file. To do this, in the Property Inspector click the column next to `ErrorClass`, then select a style class.

> **Note:** To use one or more inline styles, expand ErrorStyle in the Property Inspector; then enter or select a value next to the style you want to specify, e.g., background-color.

Example 11–19 Changing The Appearance of Error Messages, Code Sample

```
In CSS file: mystyles.css:

.error {
 font-style: italic;
   color:red;
}

.prompt {
```

```
 color:blue;
}

In the JSF file:

...
<f:view>
 <html>
  <head>
   <link media="screen" rel="stylesheet" href="css/mystyles.css"/>
  </head>
<body>
 <form>
  <h:inputText id="someid" value="{somebean.someproperty}"/
  <h:message for="id" errorClass="error"/>
<h:outputText value="{}" styleClass="prompt"/>
 ...
   </form>
   </body>
  </html>
 </fview>
...
```

11.2.2.12 How to Configure JSF Applications

You register JSF application resources such as managed beans, custom validators and converters, and define navigation rules in the application configuration file. Typically, this JSF configuration file is named `faces-config.xml`.

JSF allows more than one <application> element in a single faces-config.xml file. The overview editor for JSF configuration files only allows you to edit the first instance in the file. You'll need to edit the file directly using the XML source editor for any other <application> element.

You configure referenced beans in the `faces-config.xml` file. By declaring the bean in this file, design-time tools can understand beans that are not available at design time (such as data access) but will be available at runtime.

JDeveloper automatically creates a `WEB-INF/faces-config.xml` file when you create an application using one of the JSF web application templates. However, you can have more than one JSF configuration file. You might choose to do this if you need individual configuration files for separate areas of your application. Additionally, if you choose to have packaged libraries containing custom components and/or renderers, you need a separate `faces-config.xml` file for each library. For these, the configuration file is stored in the META-INF directory (as opposed to the WEB-INF directory).

To use the overview editor for configuration files to set the <application> element:

1. Open the overview editor for JSF configuration files.
2. In the left-hand column, select **Application.** The main area of the editor displays each of the child elements to configure. If you do not specify a value for an element, the default JSF implementation class is used.
3. In the main area, populate the text fields with class names that correspond to the child elements. For all elements that take a fully qualified class name as a value, you can use the **Browse...** button to launch the Class Browser to find the class. Once you exit a field, the value is populated to the XML file.

To add a bean to the configuration file using the JSF Configuration Editor:

1. In the Application Navigator, double-click on the `faces-config.xml file.` This file is located in the `Web Content/WEB_INF` directory.
2. At the bottom of the window, select the **Overview** tab. The JSF Configuration Editor window displays.
3. In the element list on the left, select **Referenced Beans.**
4. Use the **New**, **Edit,** and **Delete** buttons to configure the bean.

To manually add a bean to the configuration file:

1. In the Application Navigator, double-click on the `faces-config.xml` file. This file is located in the `Web Content/WEB_INF` directory.
2. At the bottom of the window, select the **Source** tab. The file opens in the XML Source editor.
3. Add the referenced bean element.

To create a new JSF configuration file:

1. In the Application Navigator, select the project to add your new configuration file. The project contains a WEB-INF node, which in turn contains the file web.xml.
2. Right-click the project node and choose **New** from the context menu.
3. In the New Gallery, go to the **Categories** tree, expand the **Web Tier** node, then select JSF/Facelet.
4. In the **Items** list, select **JSF Page Flow & Configuration.**
5. Click **OK**. The Create JSF Configuration File dialog appears.
6. Set the values according to the purpose of the configuration file. If you are adding a configuration file for your application:
    1. Enter a **File Name** for the new configuration file.
    2. Verify or change the **Directory.**
    3. Check the **Add Reference to web.xml** checkbox. When selected, JDeveloper adds the new file name to `web.xml`, so that JSF reads it as part of your application configuration.
    4. Click **OK.** This creates a new configuration file using the entered name.
7. If you are creating a configuration file for custom components or other JSF classes delivered in a library `.jar`:
    1. Set the file name to `faces-config.xml`.
    2. Change the **Directory Name** to `META-INF`.
    3. Clear the **Add Reference** to **web.xml** checkbox.
    4. Click **OK**. This creates a new configuration file using the entered name. You can then include this configuration file in the `.jar` file that you use to distribute your components or classes.

Editing a JSF Configuration File:

1. In the Application Navigator, locate the configuration file to edit. By default, this is the `faces-config.xml` file. It is located in the `WEB-INF` node of the JSF project.
2. Double-click the file to open it.

3. The JSF navigation diagrammer appears by default. To select an editor, click one of the tabs at the bottom of the editor window. To open:
   - JSF navigation diagrammer, click **Diagram.**
   - Overview editor for JSF configuration fules, click **Overview**.
   - XML source editor, click **Source**.
   - History tool, click **History.**

11.2.3 How to Run and Test JSF Applications

JDeveloper has an Integrated WebLogic Server that enables you to run and test web applications from the IDE. No special connection setup is required. Run either the entire application project or individual JSF pages.

To run and test individual pages:

1. In the navigator or the JSF navigation diagram (`faces-config.xml`), select the JSF page to run.
2. Right-click the JSF page and choose **Run** from the context menu. The JSF page is displayed in your default browser. If this is the first time you run or start your domain, and the server has not yet been created, you will be prompted to provide a new password in the Configure Default Domain dialog.

To run and test an entire project:

1. In the navigator, select the application project (for example, ViewController).
2. Right-click the project and choose **Run** from the context menu. The application is launched in your default browser.
3. The Configure Default Domain dialog appears if this is the first time you run or start the domain and the server has not yet been created. Enter your new password.

To run a project, you must first specify a default run target. If you have not already done so, JDeveloper prompts you to enter a default run target the first time you run a project. You can also specify the default run target by editing the project properties.

When you run a JSF application from the IDE, JDeveloper automatically:

- Compiles the application.
- Starts the Integrated WebLogic Server processes and launches the application in your default browser using the default address.

  For example:

  http://127.0.0.1:8988/myproject-ViewController-context-root/faces/home.jsp

  Where 127.0.0.1 is your your_machine_IP_address and 8988 is your http_port.

  Note that you can change the default application name and web context root in the project properties.

11.3 Developing Applications with HTML Pages

JDeveloper provides full support for developing with HTML technology in accordance with the HTML 4.01 W3C specification at `http://www.w3.org/TR/html401/`.

JDeveloper gives you a full set of integrated and synchronized design tools and components for creating and editing HTML pages. For information on the HTML Source Editor and Visual Editor see Section 11.1.1, "Getting to Know the Source Editor Features", and Section 11.1.2, "How to Work in the Visual Editing Environment".

11.3.1 How To Build Your HTML Pages

To get started with you HTML web pages you first need to create a web application. To see the available application types go to Table 11–6, " Web Application Templates".

Once you have created your web application framework, you re ready to start building your HTML pages.

HTML Core Components

When you are building your HTML page use the Component Palette to click or drop and drag most of the commonly used tags into your page. JDeveloper features a commonly used set of HTML element tags as well as a set of form tags to add user input attributes and behaviors.

Figure 11–17 HTML Common Component Palette

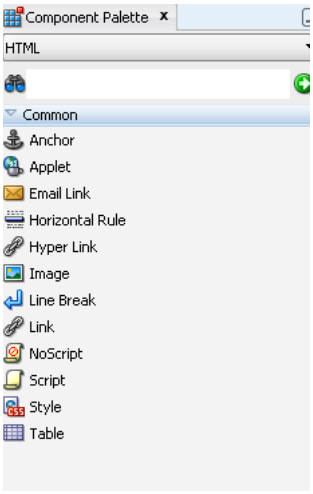


Table 11–13 HTML Common Components

Tag Name	Description
Anchor	Inserts a named anchor <A name> invisible element.
Applet	Embeds a Java applet in your page.
Email Link	Inserts an HTML <A> element in your page with the email address you provide.
Horizontal Rule	Inserts HTML <hr> element in your page at the current cursor location to display a horizontal line.
Hyper Link	Inserts a link to a HTML reference you define.

Table 11–13 (Cont.) HTML Common Components

Tag Name	Description
Image	Adds the HTML <img> element to insert an image into your page.
Line Break	Inserts a line break..
Link	Inserts a link to an external style sheet or any other external document.
Noscript	Provides alternate content when a script is not executed using an HTML <noscript> element.
Script	Embeds the <script> element and custom code into the page. Use code for any scripting language including VBScript, Tcl, and JavaScript.
Style	Embeds an internal style sheet in the document.
Table	Inserts a skeleton HTML <table> tag.

Figure 11–18 HTML Forms Component Palette

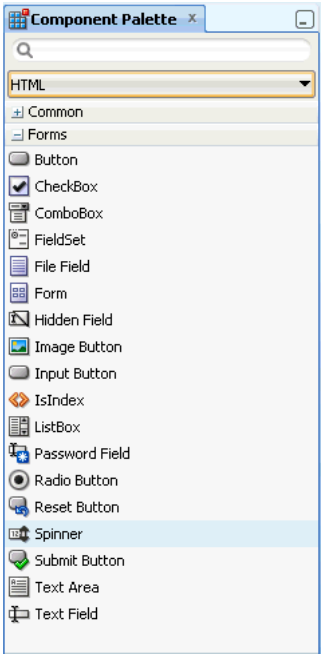


Table 11–14 HTML Forms Components

Tag Name	Description
Button	Inserts an HTML <button> element with the type attribute specified, to create a push button.

Table 11–14 (Cont.) HTML Forms Components

Tag Name	Description
Checkbox	Inserts the HTML <input> element with the type attribute specified, to create a checkbox control in a form.
Combo Box	Inserts a select element to define a form control for the selection of options.
Fieldset	Inserts a fieldset element that defines a form control group. By grouping related form controls, authors divide a form into smaller, more manageable parts, improving usability issues when confronting users with too many form controls.
File Field	Inserts an HTML <input> element with the type attribute specified, to create a file select control. The file select control creates a Browse button and field so a user can select files to be submitted with a form
Form	Inserts an HTML <form> tag to insert form processing information into your page.
Hidden Field	Inserts an HTML <input> element with the type attribute specified, to create a hidden control in a form.
Image Button	Inserts an HTML <input> element with the type attribute specified, to create a graphical submit button.
Input Button	Inserts an HTML <input> element with the type attribute specified, to create a push button. Push buttons have no default behavior.
IsIndex	Use to insert an HTML <isindex> tag to create a single-line text input control. This element has been deprecated. Authors should use the <input> element.
ListBox	Use the HTML <select> element to create a menu of choices represented by an <option> element
Password Field	Inserts an HTML <input> element with the type attribute specified, to create a password field.

Table 11–14 (Cont.) HTML Forms Components

Tag Name	Description
Radio Button	Inserts an HTML <input> element with the type attribute specified, to create a radio button control in a form.
Reset Button	Inserts an HTML <input> element with the type attribute specified, to create a reset button; this resets all controls to their initial values.
Spinner	Inserts an HTML <select> element to create a menu of choices represented by an <option> element. A <select> element must contain at least one <option> element.
Submit	Inserts an HTML <input> element with the type attribute specified, to create a button that submits a form.
Text Area	Inserts an HTML <textarea> element to create a multiline text input field.
Text Field	Inserts an HTML <input> element with the type attribute specified, to create a text field.

To create an HTML page:

The New Gallery wizard walks you through all of the necessary steps to build the web pages framework for your application.

1. In the Application Navigator, select the project in which you want to create the HTML page.
2. Choose **File** > **New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **HTML**, as shown in Figure 11–19.
4. Leave the **Directory** field unchanged to save your work in the directory where the system expects to find web application files, as shown in Figure 11–20. In the **File Name** field, enter the name of the file you want to generate then click **OK**. A simple HTML file is generated and appears in your active project. The deployment descriptor file web.xml is also added to your project. The deployment descriptor file is used by the Integrated WebLogic Server when you run the HTML.

Figure 11–19 New Gallery Create HTML Page Option

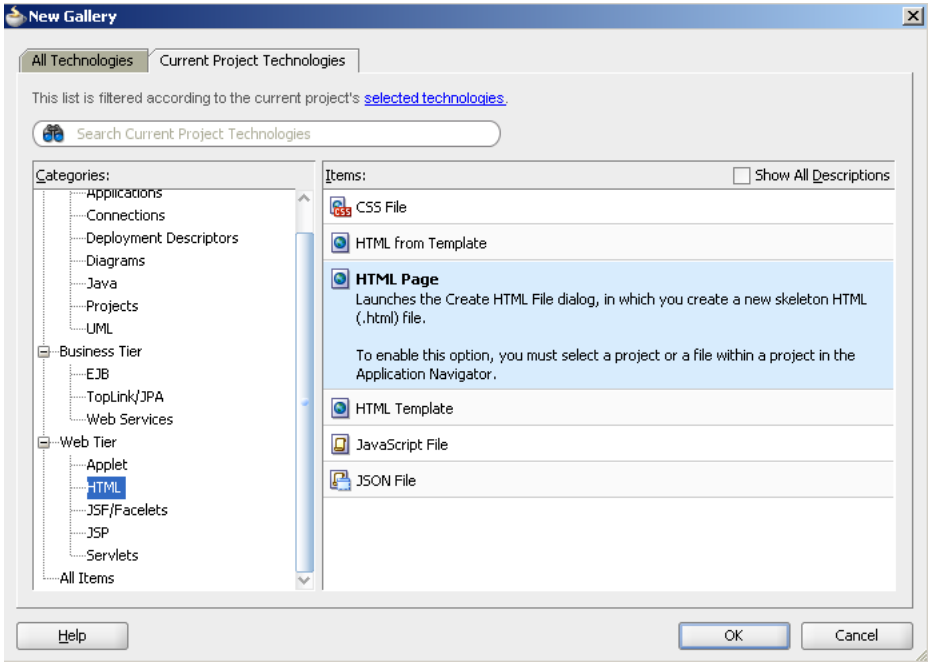
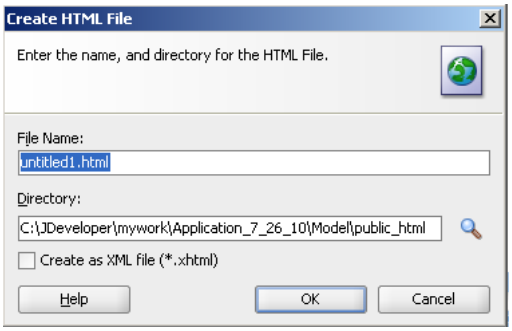


Figure 11–20 Create HTML Dialog



To Save JSP Files as HTML:

You can save your JSP pages as HTML pages by opening your JSP file in the source editor and choosing **File** > **Save as HTML**.

The Save as HTML option saves a copy of your file with the HTML extension leaving the original file unchanged. The HTML file preserves text formatting so when it is viewed in a browser or as a snippet of code on a blog it looks the same as when viewed. The saved HTML file can be reopened and viewed as HTML, but it won't be understood as code.

11.3.2 How to Work with Cascading Style Sheets

Use Cascading Style Sheets (CSS) to control the style and layout of multiple web pages. CSS styles can define the formatting attributes for HTML tags, ranges of text identified by a class attribute, or text that meets criteria conforming to the Cascading Style Sheets (CSS2) specification. For more information on CSS, see the W3C web page at `http://www.w3.org/TR/1998/REC-CSS2/`.

For CSS Development, JDeveloper provides the following tools:

- The Cascading Style Sheet Source Editor, which provides a full set of Java-aware Code Insight editing features.
- The ADF Skin Editor, which allows you to create and modify ADF skins. An ADF skin is a type of CSS file that defines the look and feel of an ADF application.
- Wizards to create new HTML pages.
- Source and visual editors to edit HTML page.
- Drag and drop linking a CSS file to an HTML or JSP page.
- Property Inspector to set or modify CSS selector properties and values.
- Code Insight to provide available options and complete code while editing.
- Structure window to sort and view CSS elements by groupings.

Table 11–15 lists the Cascading Style Sheet Source Editor features:

Table 11–15 CSS Source Editing Features

Feature	Description
Code insight for CSS	Displays a list of HTML selectors, properties, values, pseudo-classes and pseudo-elements, for the CSS file under the cursor, to select an appropriate completion. For example, if you place the cursor just after the opening brace in a style rule, it displays a list of all possible properties to enter at that point in the file.
Reformat for CSS	Correctly reformats your code on that CSS page. Right-click on your file in the CSS editor or from the Application Navigator and choose Reformat.
CSS Error handling	Highlights invalid CSS properties, values, and missing semicolon and braces.
Stylesheet linking to HTML files	In the Component Palette, select the HTML palette, Common page, and link a stylesheet to your HTML files simply by dropping a Link element into your HTML page. Another option is to choose **CSS** in the Component Palette. The list of available CSS files displays in the Component Palette. You can then drag and drop any CSS file from the Componenet Palette to the page.
Style preview	See what your styles look like while you're coding.
Code colors	Easily spot properties, values, and keywords.
CSS Refactoring	Refactors across the application when you rename CSS files, class and ID attributes, or move, copy, and safe delete files.
Brace Matching for CSS Code Editor	Highlights the matching braces, brackets, and parentheses in the code editor based upon the cursor position.
Toggle Line Comments	Adds or removes comment markers from the beginning of each line in a selected block. Select a single line to comment or uncomment that line only.
Quick docs	Open the description from the W3C standard.

11.3.2.1 How to Select and Group CSS Elements

When a CSS file is open for editing, CSS selectors in the file are displayed in the Structure window in the following type icons:

Element

The HTML element or tag defined by the CSS selector. Property and value are separated by a colon and surrounded by curly braces. For example: `body {color:black;.}`

Class

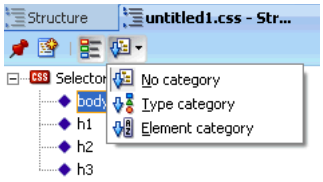
Different styles defined for the same type of HTML element. For example `p.right {text-align:right;}` to define right-aligned paragraph text, and `p.left {text-align:left;}` to define left aligned paragraph text. You can also omit the tag name in the selector to define a style that will be used by all HTML elements that have a certain class. For example `center {text-align:center;}` defines all HTML elements with `class="center"` to be center-align.

ID

Style unique to one HTML element. For example `p#para1 {color:green;}` defines the p element that has the id `value="para1" and *#ver905 {background-color:red;}` defines the first HTML element with id `value="ver905"`.

Tools For Grouping Elements

You can use the Categories dropdown list in the Structure window toolbar to show CSS selectors by categories.



No category

Displayed in order of appearance in the CSS file. Default setting.

Type Category

Arranged by CSS selector types: Element, Classes or ID.

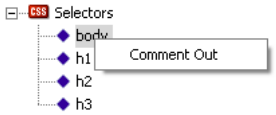
Element category

Arranged by HTML element or tag.

Select a CSS selector in the Structure window to highlight the selector in the CSS file and display associated properties and values in the Property Inspector for editing.

Select the Separate Grouped Selectors icon to separate or ungroup the selector categories in the Structure window.

Select an element group and right click and select **Comment Out** , to comment out the selected element in your CSS file.



11.3.2.2 How to Use the CSS Basic Tools

You can create your CSS stylesheet with a New Gallery wizard. Once created, drag and drop a stylesheet onto your web page to link the stylesheet. Use the soure editor with Code Insight to make changes directly in the CSS code, and edit your selector properties and values in the Property Inspector.

> **Note:** The **Preview** tab is located near the **Source** tab in the source editor. You can use it to see what the CSS formatting you have entered on the **Source** tab will look like.

To create a simple Cascading Style Sheet:

1. In the Application Navigator, select the project in which you want to create the new style sheet.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **HTML.**
4. In the **Items** list, double-click **CSS File** to open the CSS File dialog.
5. Leave the **Directory Name** field unchanged to save your work in the directory where JDeveloper expects to find web application files.
6. In the **File Name** field, enter the name of the file you want to generate then click **OK**. A simple CSS file is generated and appears in your active project in the CSS folder under Web Content.

To set or modify CSS selector properties and values:

1. In the Structure window of the CSS file, select the CSS selector element, class or id in which you want to set a property.
2. In the Property Inspector, scroll until the property you want is visible. To quickly locate a property in a long list, click the search button in the Property Inspector toolbar. In the Find text field, type the name of the property, then press Enter. Enter the property value in the right column in one of the following ways:
   - Type the string value for the property In a text field, then press Enter.
   - Click a button in a value field to choose a value from the displayed list.
   - Click in a value field to display the ellipsis button. Click the ellipsis to display an editor for that property. Set the values in the property editor, then press **OK**. The selector value is modified and pages linked to the CSS file reflect the style changes.
   - Type the string value for the property In a text field, then press **Enter.**

To edit a CSS File in the Source editor:

1. In the Application Navigator, double-click the CSS file to open it in the default Source editor window.
2. Enter the CSS selector (HTML element, class, or id) you wish to define.
3. Enter the { (open curly bracket)and press Ctrl+Space (using the default keymapping) to invoke Code Insight.
4. Double-click a property name from the list of valid properties. The selected property is inserted in the file, followed by a colon and a space. For example: `{background-color:`
5. To enter a value for the property you have inserted press Ctrl+Space to open a list of valid values and double-click a value to insert it. The selected value is inserted, followed by a semicolon. For example: `body {text: blue;`
6. Add other properties and values as necessary. Be sure to use a semicolon between a property value and the next property. For example: `p {text-align:center; color:red;`
7. When you've finished adding properties and values, enter the } (close curly bracket).

Note: The Structure window displays any CSS syntax errors found as you edit.

Double-click an error or element in the Structure window to edit it in the Source editor.

11.3.3 How to Work with HTML Tables

Using the visual editor, use tables to lay out data on your HTML pages. Once you create a table you can easily modify both the appearance and the structure of the table. You can edit tables to add text and images; add, delete, resize, reorder, split, and merge rows and columns; modify table, row, or cell properties for color and alignment; copy and paste cells, and nest tables in table cell.

To add text to a table cell:

1. Click in a cell to add text and when a blinking cursor appears, do one of the following:
   - Type text into the table. Table cells automatically expand as you type.
   - Paste text copied from another page.
2. Press Tab to move to the next cell or press Shift+Tab to move to the previous cell. Pressing Tab in the last cell of a table automatically adds another row to the table.

Using a table cell as the insertion point you can add and remove graphics or other UI and data elements to tables.

To remove content from one or more cells select cells:

- Click **Delete** or **Backspace**.

  Or

- From the main menu select **Edit > Delete.**

Note that only the contents of the cell, not the cell, will be removed from the table. If the entire row or column is selected, the table structure will be modified to remove the row or column along with the contents of the cell.

11.3.3.1 How to Format Tables and Cells

Set the properties of HTML tables, rows, columns, and cells using the design tools. Use the Property Inspector, the Edit Table dialog, or the visual editor toolbar to set table element properties.

When formatting tables with the design tools, you can define properties that apply to the entire table or to selected cells, rows, or columns in the table. When a property like background color or alignment is set with a value for the whole table and a different value for individual table cells, precedence in formatting is applied in the following order:

1. `table cell, <td> tag`
2. `table row, <tr> tag`
3. table, <table> tag

If you specify a background color of green for a single cell and then set the background color of the entire table to red, the green cell will not change to red, since the <td> tag takes precedence over the `<table>` tag.

To set table and cell properties using the Property Inspector:

1. Select the table, row, or cell in the visual editor, or the corresponding <table>, <tr>, <td> in the Structure window. The Property Inspector displays the property values for the selected element. If the Property Inspector is not in view, choose **View > Property Inspector** or use the shortcut Ctrl+Shift+I.

   Tip: To quickly locate a property in a long list, click the search button in the Property Inspector toolbar. In the Find text field, type the name of the property, then press Enter.

2. Enter the property value in the right column in one of the following ways:
   - Type the string value for the property in a text field, then press Enter.
   - Click In a value field to choose a value from the displayed list.
   - Click in a value field to display the ellipsis button. Click the ellipsis to display an editor for that property. Set the values in the property editor, then press **OK**.

To set table and cell properties using the visual editor Toolbar:

1. Select the table, row, or cell in the visual editor. You can also select the corresponding `<table>, <tr>, <td>` in the Structure window.
2. Use the standard toolbar editing icons to set properties such as: align and indent/outdent, and so forth.

To resize a table, do one of the following:

- Select the table in the visual editor and use the resize handles to drag the table height, width, or both to the desired size.
- Select the table in the visual editor or the corresponding `<table>` element in the Structure window, and then set the table width attribute in the Property Inspector.

- Double-click the table in the visual editor and in the Edit Table dialog reset the table width in pixels or percentage of page width.
- Right-click the table in the visual editor or the corresponding <table> element in the Structure window, and then choose Edit Tag from the context menu to display an **Edit Table** dialog.

To change the size of rows or columns:

1. In the visual editor, open the page with a table you want to resize the rows or columns.
2. In the visual editor, open the page with a table you want to resize the rows or columns. Place your cursor at the border of the row or column you wish to resize, and click when the horizontal border handle or vertical border handle appears.
3. Drag the row or column border to the desired size, then release the mouse.

To add rows or columns to a table:

1. Select the table cell in the visual editor or the corresponding `<td>` element in the Structure window.
2. Right-click the table cell or element and select **Table** in the context menu.
3. Choose one of the following:
   - Select Insert Row to add a row above the row where the table cell is selected.
   - Select Insert Column to add a column before the column where the table cell is selected.
   - Select Insert Rows Or Columns... for an Insert Rows or Columns dialog to add multiple rows or columns and to specify the location for adding the row(s) or column(s). Then click **OK**.

To remove rows or columns in a table:

1. Select the table cell in the visual editor or the corresponding `<td>` element in the Structure window.
2. Right-click the selected table cell or element and select **Table** in the context menu.
3. Choose one of the following:
   - Select **Delete Row** to remove the row where the table cell is selected.
   - Select **Delete Column** to remove the column where the table cell is selected.

   You can also select one or more rows or columns in the visual editor or the corresponding `<tr>` element in the Structure window and do one of the following:
   - Click **Delete** or **Backspace**.
   - From the main menu select **Edit > Delete**. Note that If you are deleting the last row in the table the entire table is removed.

To merge table cells:

1. Select the table cells in the visual editor, or the corresponding <td> elements in the Structure window. The selected cells must be contiguous and form a rectangular region.
2. Right-click the selected table cells or elements and select Table from the context menu, then click M**erge Cells.**

Or

From the main menu select **Design** and select **Table**, then click **Merge Cells**. The contents of the individual cells are placed in the resulting merged cell.

To split a table cell:

1. Select the table cell in the visual editor or the corresponding <td> element in the Structure window.
2. Right-click the selected table cell or element and select Table from the context menu, then click Split Cells.

   Or

   From the main menu select **Design** and select **Table,** then click **Split Cells**.
3. In the Split Cells dialog, choose whether to split the cell into rows or columns, and then enter the number of rows or columns.
4. Click **OK**.

To change the display order of rows, columns, or groups of table cells using the visual editor:

1. Select the row, column, or group of table cells you want to change the order of in the HTML table. The selected cells must be contiguous and form a rectangular region.
2. Drag the row, column, or group of table cells to a new position in the table with one of the following actions:
   - To insert a row or group of cells above a target row, drag it towards the top of the row until you see a horizontal line with an embedded up arrow, then release the mouse button.
   - To insert a row or group of cells below a target row, drag it towards the bottom of the row until you see a horizontal line with an embedded down arrow, then release the mouse button.
   - To insert a column or group of cells before a target column, or a column before a target column, drag it towards the left of the row or column until you see a vertical line with an embedded left arrow, then release the mouse button.
   - To insert a column or group of cells after a target column, drag it towards the right of the node until you see a vertical line with an embedded right arrow, then release the mouse button.

To change the display order of rows using the Structure window:

1. Select the `<tr>` element you wish to change the order of in the table. The selected cells must be contiguous and form a rectangular region.
2. Drag the row, column, or group of table cells to a new position in the table with one of the following actions:
   - To insert a row above a target row, drag it towards the top of the row until you see a horizontal line with an embedded up arrow, then release the mouse button.
   - To insert a row below a target row, drag it towards the bottom of the row until you see a horizontal line with an embedded down arrow, then release the mouse button.

To increase row or column span in a table:

1. Select the table cell in the visual editor or the corresponding `<td>` element in the Structure window.
2. element in the Structure window. Right-click the table cell or element and select **Table** in the context menu.
3. Choose one of the following:
   - Select **Increase Row Span** to expand the selected cell by one row.
   - Select **Increase Column Span** to expand the selected cell by one column.

To reduce row or column span in a table:

1. Select the table cell in the visual editor or the corresponding `<td>` element in the Structure window.
2. Right-click the selected table cell or element and select **Table** in the context menu.
3. Choose one of the following:
   - Select **Decrease Row Span** to reduce the span of the selected cell by one row.
   - Select **Decrease Column Span** to reduce the span of the selected cell by one column.

11.3.4 How to Work with HTML Forms, Text, and Images

Use the design tools to add and format text on JSP or HTML pages. Use the Component Palette to add your HTML forms, and the Property Inspector and Structure window to manage elements and the configure properties for your HTML forms.

For your HTML graphics, use the Component Palette to easily add graphics to your pages, or you can drag and drop them from your Windows Desktop or Explorer. You can insert an image into a page, table, or form, or use an image as a background. Modify images to set image size, add a border, and set alignment on a page or in a table cell. Create interactive graphics, such as rollover images or navigation bars, by adding a JavaScript event to your image.

11.3.4.1 How to Work with HTML Forms

Use HTML forms on your HTML pages to interact with or gather information from users of your web pages. Forms are composed of:

- Form tags, which include form processing information.
- Form fields, which may include text fields, menus, checkboxes, or radio buttons.
- Submit button, which sends the data to the form processing agent.

To create a new HTML form:

With a JSP or HTML file open, do one of the following:

- Select the insertion point in the visual editor or the Structure window where you want the form to appear, then click **Form** on the HTML page of the Component Palette.
- Drag the Form element from the HTML page of the Component Palette to the desired insertion point on the page or in the Structure window. The HTML code to create a skeleton form is inserted into your HTML or JSP file. Note that a form appears as a dotted outline in the visual editor.

After creating the skeleton form, add form fields and buttons and specify form processing information. By default, forms are created with a Get form processing attribute.

When form fields or buttons from the Component Palette are added to the HMTL or JSP page, a <form> element is automatically inserted as a parent element.

To delete a form element:

Select the form in the visual editor or the corresponding `<form>` element in the Structure window, and do one of the following:

- Click **delete** or **backspace**.
- From the main menu select **Edit > Delete.** The form, and any form fields and buttons within the form are removed. To remove the form element without deleting form fields or buttons, right-click the form and select **Form > Remove Form Tag**.

To insert a form field or button:

1. With a JSP or HTML file containing a form element open in the visual editor, do one of the following:
   - Select the insertion point in the visual editor or the Structure window where you want the field or button to appear on the form, then click the desired element on the HTML page of the Component Palette.
   - Drag the form field or button element from the HTML page of the Component Palette to the desired insertion point on the form or in the Structure window. Note that If you attempt to insert a form field or button without first creating the form, you'll get a message "Do you want to add a form element for this component?" Choose **Yes** to automatically create form tags for the field or button. Checking **Hide Add Form Element Confirmations** in this dialog will close the automatic display of the dialog. Reinstate the display by selecting **Tools > Preferences > JSP HTML visual editor** from the main menu and checking **Prompt to Add Form Element**.
2. For form fields or buttons with required attributes, set property values using the displayed editor dialog.

To delete a form field or button, do one of the following:

- Select the element and click **Delete** or **Backspace**.
- Select the element and from the main menu choose **Edit > Cut.**

To edit form processing information using an Edit Form dialog:

1. Right-click the form in the visual editor or the corresponding `<form>` element in the Structure window, and select **Edit > Tag**.
2. In the Edit Form dialog set the form processing attributes.
3. Click **OK** to add the form processing information to the form element. For example:
```
<form method="post"
action="http://www.oracle.com/orderEdit.html"
enctype="application/x-www-form-urlencoded"
name="form1"></form>.
```

To set form processing information using the Property Inspector:

1. Select the form in the visual editor, or the corresponding <form> element in the Structure window. The Property Inspector displays the property values for the selected element. If the Property Inspector is not in view choose **View > Property Inspector** or use the shortcut `Ctrl+Shift+I`.
2. Scroll until the property you want is visible, then select it with the mouse or the arrow keys. A brief description of the property is displayed at the bottom of the Property Inspector.
3. Enter the property value in the right column in one of the following ways:
   - Type the string value for the property In a text field, then press **Enter**.
   - Click In a value field to choose a value from the displayed list.
   - Click in a value field to display the ellipsis button. Click the ellipsis to display an editor for that property. Set the values in the property editor, then press **OK**.

     Tip: To quickly locate a property in a long list, click the search button in the Property Inspector toolbar. In the Find text field, type the name of the property, then press Enter.

To change the form method from the context menu:

1. Right-click the form in the visual editor or the corresponding <form> element in the Structure window, select **Form,** and then **Method**.
2. In the sub-menu select **Post** or **Get** to change the form method.

11.3.4.2 How to Work with HTML Text

Use the visual editor to add and format text to your HTML files.

Use the toolbar in the visual editor to set text properties in your pages. Attributes set using the toolbar are marked with a green square in the Property Inspector. To undo changes, from the main menu select **Edit > Undo** action. To reset text properties to default values, select and delete the value from the Property Inspector.

To add text, do one of the following:

- Click the position in the visual editor where you wish to insert text. Begin typing when the blinking cursor appears.
- Copy and paste text from files in the same project or different projects.

  You can format inserted text using the Toolbar in the visual editor. The Toolbar applies manual or inline formatting in the page. For example

```
<H5><EM><FONT color="#ff0000">This is a Heading 5 in italics
iUse the Toolbar to:n the color red</EM></FONT></H5>
```

HTML Toolbar Features

When editing your pages in the visual editor you can use the toolbar for formatting changes.

Use the tool bar to:

- Set the default formatting style (None, Paragraph, Preformatted, Heading 1, Heading 2, and so on) for a block of text.
- Change the font, color, and alignment of selected text.

- Apply formatting such as bold, italic, or underline.
- Create ordered (numbered) and unordered (bulleted) lists.

Formatting Text with CSS Features

You can also use Cascading Style Sheets (CSS) to automatically update text and page formatting within a page or across several web pages. CSS styles define the formatting for all text in a particular class or redefine the formatting for a particular tag such as h2 or I. Apply CSS styles with an external style sheet.

You can use CSS styles and manual or online HTML formatting within the same page. Manual HTML formatting overrides formatting applied by a CSS style. For complete information on CSS style sheets, see the W3C Cascading Style Sheets home page at, http://www.w3.org/Style/CSS.

To set text properties:

1. Select the text in which you wish to set a manual or online HTML style.
2. Use the tabular to set text properties.

11.3.4.3 How to Work with HTML Images

The JDeveloper design tools support the following graphic file formats:

- JPEG/JPG
- GIF
- PNG

To insert an image:

1. With a file open in the visual editor, do one of the following:
   - Select the insertion point in the visual editor or the Structure window where you want the image to appear on the page, then click Image on the page of the Component Palette.
   - Drag the Image element from the page of the Component Palette to the desired insertion point on the page or in the Structure window.
2. In the Insert Image dialog that displays, click **Browse** to choose a file, or type the path for the image file location. Browsing to the file location opens the Select Image Source dialog, which displays the directory based on current context. If the image file is located outside the HTML root of the current project you will be prompted with an option to add the file to the current context in the Application Navigator. Click **Yes** for a Save Image dialog to add the image to the document root.
3. Set additional image properties in the Insert Image dialog.
4. Click **OK**. The image appears on your page.

   You can also drag an image from your Windows Desktop or Explorer to the desired location on the page. You will be prompted with an option to add the file to the directory based on current context in the Application Navigator. Click Y**es** for a Save Image dialog to add the image to the document root. The image will appear on your page.

To delete an image, do one of the following:

- Select the image and click **Delete** or **Backspace**.

- Select the image and from the main menu choose **Edit** > **Cut.**

To resize an image, do one of the following:

- Right-click and select **Properties** , then adjust pixels for width and height.
- Select and use the resize handles at bottom and right sides of the image and in the bottom right corner to adjust the image width and height.
- Select and modify the image width and height attributes in the Property Inspector.

  Image properties set using the visual editor are marked in the Property Inspector with a green square. To return a resized element to its original dimensions delete the values in the width and height fields in the Property Inspector, or click the **Reset Size** button.

To move an image by dragging:

In the visual editor or Structure window do any of the following:

- Drag the image from the original position to an insertion point in the visual editor or Structure window.
- Right-click drag the image from the original position to an insertion point in the visual editor or Structure window, and then choose **Move Here** from the context menu.

In the visual editor or Structure window do any of the following:

- Cut the image. Then, paste into some other position in the Visual Editor or Structure Window.
- Cut the image. Then, paste into another file in the same project or a different project.

To use an image as a background:

1. Select the page `<body>` element in the Structure window. The Property Inspector displays the property values for the selected element. If the Property Inspector is not in view choose **View > Property** Inspector or Ctrl+Shift+I.
2. Scroll to the background property in the Property Inspector, and then select it with the mouse or the arrow keys.
3. Enter the property value in the right column in one of the following ways:
   - Click in a value field to choose an available background image from the displayed list.
   - Click in a value field to display the ellipsis button. Click the ellipsis to display a background dialog, and click **Browse** to choose a file, or type the path for the image file location. Browsing to the file location opens the Select Image Source dialog, which displays the directory based on current context. If the image file is located outside the HTML root of the current project, you will be prompted with an option to add the file to the current context in the Application Navigator. Then click **Yes** for a Save Image dialog to add the image to the document root. Click **OK**. The image will tile as the background image on your page.

11.4 Working with Java Server Pages

This section covers JDeveloper support and tools for your user interface development using JavaServer Faces (JSP) technology within the Java EE platform.

JDeveloper provides a complete user interface development environment for Java Server pages (JSP) development in accordance with the JSP 2.1 specification defined at http://jcp.org/aboutJava/communityprocess/final/jsr245/index.html.

11.4.1 How to Build Your JSP Application

You can build your application from the ground up using the features provided in JDeveloper. The first thing you'll want to do is build a framework, or application template for your web pages. Get started quickly with your JSP projects using the application templates. Choose from a combination of technologies to include in your application as you build your application with the New Gallery Wizard. The application you choose determines the project folders created and the libraries added to the folders as shown in Table 11–6.

11.4.1.1 JSP Core Components

JDeveloper comes with a Component Palette stocked with standard JSP components that you can easily drag and drop onto your JSP pages as shown in Figure 11–21 and Table 11–16.

Figure 11–21 JSP Core Components Palette

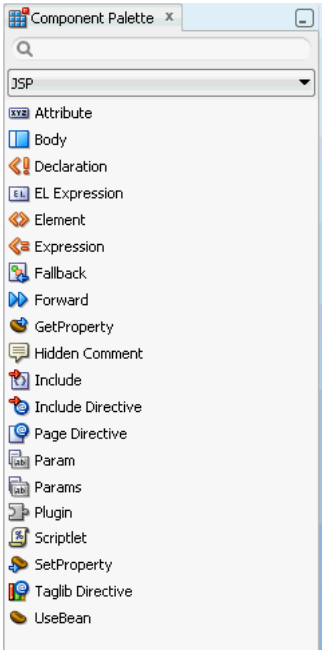


Table 11–16 JSP Core Components

Tag	Description
Attribute	Defines the value of a tag attribute in the body of an XML element instead of in the value of an XML attribute.
Body	Specifies the body of the tag.

Table 11–16 (Cont.) JSP Core Components

Tag	Description
Declaration	Declares a method or variable valid in the scripting language used in the JSP page.
EL Expression	Contains an expression in the JSP Expression Language (EL) to provide easy access to application data stored in JavaBeans components.
Element	Dynamically defines the value of the tag of an XML element. This action can be used in JSP pages, tag files and JSP documents
Expression	Contains an expression valid in the scripting language used in the JSP page. The expression is evaluated, converted to a String, and inserted into the response where the expression appears in the JSP page.
Fallback	Displays a text message if the dialog to initiate the download of plug-in software fails. A translation error will occur if the element is used elsewhere.
Forward	Forwards the request object containing the client request information from one JSP page to another resource. The target resource can be an HTML file, another JSP page, or a servlet, as long as it is in the same application context as the forwarding JSP page.
GetProperty	Gets a bean property value using the property's getter methods and insert the value into the response.
Hidden Comment	Documents the JSP page without inserting the comment in the response.
Include	Sends a request to an object and include the result in a JSP file.
Include Directive	Inserts a static file of text or code in a JSP page at translation time, when the JSP page is compiled.
Page Directive	Defines attributes that apply to the entire JSP page.
Param	Passes one or more name/value pairs as parameters to an included resource.

Table 11–16 (Cont.) JSP Core Components

Tag	Description
Params	Provide key value information.
Plugin	Executes an Applet or JavaBean in the specified plugin.
Scriptlet	Inserts a code fragment valid in the page scripting language.
SetProperty	Sets a property value or values in a JavaBean
Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.
UseBean	Locates or instantiate a JavaBean with a specific name and scope.

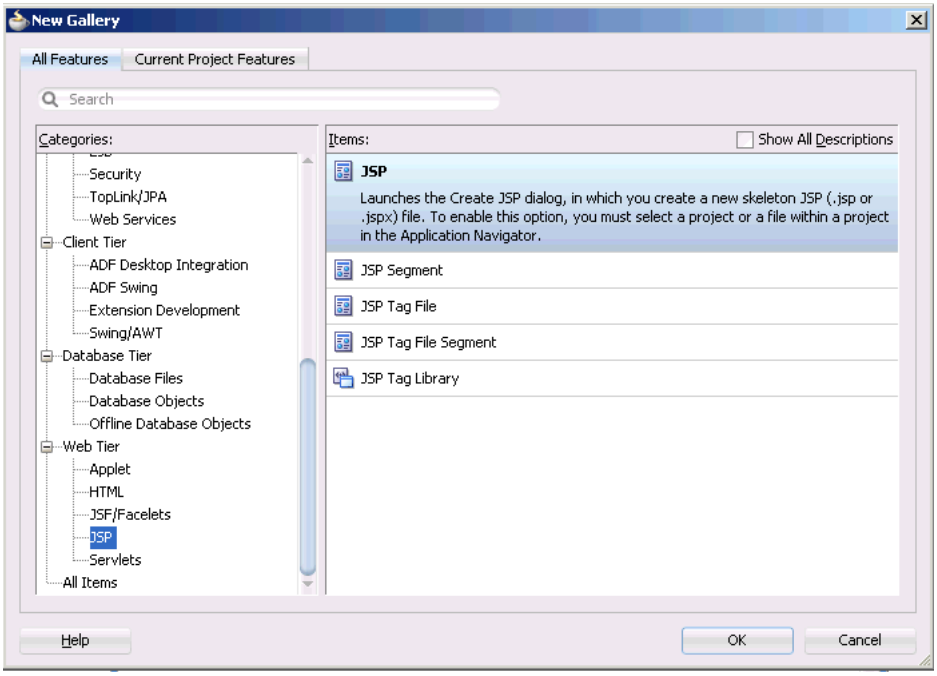
11.4.1.2 How to Create JSP Pages

The New Gallery wizard walks you through all of the necessary steps to build the web pages for of your application.

To create a new JSP page:

1. In the Application Navigator, select the project to create the new JSP.
2. Choose File > New to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **JSP,** as shown in Figure 11–22. A simple JSP is generated and appears in your active project. The deployment descriptor file `web.xml` is also added to your project. The deployment descriptor file is used by the Integrated WebLogic Server when you run the JSP.

Figure 11–22 Create JSP New Gallery Option



11.4.1.3 How to Register a Servlet Filter in a JSP Page

The Create Servlet Filter wizard available from the Web Tier category in the New Gallery creates a new filter you can use to process requests or responses to or from your JavaServer Page.

To register a servlet filter in a JSP page:

1. In the Application Navigator, select the project in which you want to create the new servlet listener, usually the project which includes your JSP.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **Servlets**.
4. In the **Items** list, double-click **Servlet Filter** to open the Create Servlet Filter wizard. This starts the Create Servlet Filter wizard which will create the servlet filter for you based on information you specify, including the implementation class and initialization parameters.
5. Click **Next** if the Welcome page displays.
6. Enter the **Filter Name**, **Filter Classname** and **Package**. Then click **Next.**
7. Select **Map to Servlet** or **JSP**, and select the name of the JSP from the dropdown list. Then click **Next**.
8. Click **New,** and enter the name and value for each initialization parameter. Then click **Finish**.

   A new servlet filter is generated and appears in your active project. The deployment descriptor file web.xml is updated with the <filter> element. The deployment descriptor file is used by the embedded web server in JDeveloper when you run the JSP.

11.4.1.4 Understanding Flow Control in JSP Pages

Web applications implement flow control by directing the display content of the web browser in response to specific user actions. Typically, web application developers create separate JSP pages or sets of pages for each task the application provides. The user makes choices in one page and clicks a link to submit their choices on the Request object. The link they click directs the Request object to the page responsible for handing the action.

How Can I Handle Flow Control in JDeveloper?

You decide the way your application handles the Request object. JDeveloper supports various options for implementing JSP page flow control:

- You can write JSP pages that use a combination of HTML generating code and Java scriplet code to link back to themselves and handle the actions. In this case, the entire action handling code is contained in the JSP page that also displays the content. This mixes HTML and flow control logic within the same file.

  Or

- You can cleanly separate JSP pages and their actions by implementing the controller outside of the JSP page.

What Features Can I Use for These Approaches?

The following approaches are supported:

- In all-in-one JSP page development, JDeveloper helps to reduce the amount of Java code visible in your JSP pages through tag libraries that provide JSP tags which encapsulate complex behavior such as implementing databound, performing data actions (such as query, browse, edit, and update), and generating reports.
- If you use JSP includes, you can benefit from the Oracle Business Components Data Tag library that implements a set of JSP page-level tags (known as component tags) that handle common actions such as navigation, querying, browsing, editing, and scrolling.
- If you fully separate the JSP display content and JSP action-handler classes, JDeveloper supports two Java EE frameworks.
  - JavaServer Faces page navigation.
  - JDeveloper provides full support to allow you to visually design page flows for web applications based on either framework.
- When you want to build applications for the web and benefit from a framework that implements many of the Java EE design patterns for interactive applications, JDeveloper provides the Oracle Application Development Framework (Oracle ADF). One of its central features is a data binding layer that uses a standard declarative way to bind data from a business service, such as web services, EJB, JavaBeans, and Oracle ADF Business Components, to UI components, such as Oracle ADF Faces components and standard HTML elements.

11.4.2 How to Debug and Deploy JSPs

JDeveloper supports deploying Web applications on any Java EE application server through the creation of a Web Module Archive (WAR). There is additional support for deployment to Integrated WebLogic Server.

To debug a JSP:

1. In the Navigator, select the JSP file you want to run.
2. Debug a JSP in any of these ways:
   - Choose **Debug | Debug <source_file>.jsp** from the main menu.
   - Right-click the JSP file and choose **Debug** from the context menu. The JSP is launched.
3. Debug your JSP as you would any other Java application.

JDeveloper performs the following functions when debugging a JSP:

- Translates the JSP into a servlet and compiles it.
- Starts the Integrated WebLogic Server process.
- Runs the resulting classes directly from the output project directory.
- Invokes the JSP in your default Web browser. For example, your browser is launched as follows:

  ```
  http://<your_machine_IP_address>:<http_port>/<context_
  root>/<path_to_JSP>
  ```

  for example:

  ```
  http://127.0.0.1:8988/Project1-context-root/untitled1.jsp
  ```

To create a web deployment descriptor:

1. In the Applications Navigator, select the project for which you want to create a web deployment descriptor.
2. Add a JSP file to the project. JDeveloper adds the web.xml file to the WEB-INF project folder the first time you create a JSP file.

   Or, to add the web deployment descriptor file yourself:

   In the New Gallery **Categories** tree, expand **General** and select **Deployment Profiles**. In the **Items** list, select **web.xml (Web Deployment Descriptor).** Click **OK.**

   If the desired item is not enabled, check to make sure the project does not already have a web deployment descriptor: a project may have only one instance of a descriptor.
3. The web deployment descriptor will be created and added to the WEB-INF folder in the project, and it will be opened in an XML editor window.

To inspect or change web deployment descriptor properties:

1. In the Applications Navigator, select the web deployment descriptor in the WEB-INF folder.
2. Right-click and choose **Properties.**
3. Select items in the left pane to open dialog pages in the right pane. Configure the descriptor by setting property values in the pages of the dialog. Click **OK** when you are done.

To edit a web deployment descriptor as an XML file:

1. In the Applications Navigator, select the web deployment descriptor in the `WEB-INF` folder.

2. Right-click and choose **Open.** The file opens in an XML editor.

11.4.3 How to Run a JSP

The Integrated WebLogic Server is responsible for running JSPs. After building your JSP, you can run it in a few easy steps.

To run a JSP:

1. In the Navigator, select the JSP file you want to run.
2. Run the JSP in any of these ways:
   - Choose **Run > Run <source_file>.jsp** from the main menu.
   - Right-click the JSP file and choose Run from the context menu.

   The JSP is launched.
3. The Configure Default Domain dialog appears if this is the first time you run or start the domain when the server has not yet been created. Enter your new password.

JDeveloper performs the following functions when a JSP is run.

- Translates the JSP into a servlet and compiles it.
- Runs the resulting classes directly from the output project directory.
- Edits the Integrated WebLogic Server web.xml file to include the servlet name and class information.
- Invokes the JSP in your default Web browser. Your browser is launched using this format:

  `http://<your_machine_IP_address>:<http_port>/<context_root>/<path_to_JSP>` for example,

  `http://127.0.0.1:8988/Project1-context-root/untitled1.jsp.`

Dynamically Modifying JavaServer Pages Files While Running

When running your JSP in the Integrated WebLogic Server, you can modify and view changes that you make to your JSP files without having to restart WebLogic Server. To view changes in your browser, you can either reload the page from the browser or you can run the page again in JDeveloper because the WebLogic Server is able to change only the file, running from JDeveloper is much faster than reloading the page from the browser.

Running JSPs with ADF Business Components Application Modules

If you are running JSPs with business components application modules in both the Integrated WebLogic Server and in a remote server instance, and have two JSPs contained in two different projects that depend on the same middle tier project, you must declare that middle tier is running inside of a WebLogic Server instance with the jbo.server.in_wls=true property.

Working with Timestamps on Source JSPs

When developing, compiling, and running JSPs, if the timestamp of a source JSP file is ever changed to an earlier timestamp, the JSP will not automatically be recompiled by JDeveloper or by WebLogic Server. It must be forced to recompile. To force recompilation, right-click on the JSP and select Rebuild, use Build->Rebuild , Build->Rebuild All, Build->Clean , or Build->Clean All.

Timestamps can go backwards in time when using source control systems (restoring an older version) or using timestamp preserving copy commands like xcopy or mv.

11.4.4 Understanding JSP Segments

A JSP fragment is a JSP page that can be included in another JSP page.

JSP segments use `.jspf` as a filename extension. By default JSP fragment files are placed with the rest of the static content in the web application folder. JSP segments that are not complete pages should always use the `.jspf` extension.

JSP segments are defined using JSP syntax as the body of a tag for an invocation to a SimpleTag handler, or as the body of a `<jsp:attribute>` standard action specifying the value of an attribute that is declared as a fragment, or to be of type JspFragment in the TLD.

11.5 Developing Applications with Java Servlets

A servlet is a platform-independent, server-side Java component used to extend the capabilities of a web server. Using servlets, you can dynamically tailor content, function, and the look and feel of your web pages. Servlets process client requests and can respond by returning any MIME type to the requesting client, including images, XML, and HTML. Servlets run inside web servers, so they do not require a graphical user interface. They are typically used to dynamically generate HTML content and present it to the requesting client. You can think of a servlet as the server-side counterpart to an applet.

Servlets are based on a standard API and protocol defined by JavaSoft. To run a servlet, your environment needs a web server that supports the JavaSoft servlet API, such as Oracle WebLogic Server, JavaSoft Java Server, and Apache Tomcat, among others. JDeveloper provides support for servlet filters and listeners (Servlet API 2.5). When you use the Create Filter wizard and Create Listener wizard, it updates the web.xml with filter and listener entries. The web.xml can also be manually edited to include or modify these entries.

For more information, see the *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

11.5.1 Understanding Servlet Support in JDeveloper

Servlets are often used to process HTTP requests submitted by a client, and to provide dynamic content by returning results of a database query to a client. This type of Java servlet is known as an HTTP servlet. A typical runtime scenario for an HTTP servlet is as follows:

- A client sends an HTTP request to the servlet. The client could be a web browser or some other application or applet.
- The servlet processes the request and responds by returning data to the client. In the case of HTML servlets, these servlets generate and send dynamic HTML content back to the client. If the servlet is designed to do so, it may request data from a database server on behalf of the client, then package and return the results to the client in an HTML form. This can be done using JDBC or by working with Oracle ADF Business Components.
- The client user can then interactively view and respond to the generated HTML content, perhaps making additional requests through the generated HTML form.

11.5.1.1 What You May Need to Know About Servlet Filters

A filter is a reusable piece of code that can transform the content of HTTP requests, responses, and header information. Filters do not usually create a response; instead you use filters to modify the requests or responses, or to perform some other action based on the requests or responses, including:

- Examining a request before calling a servlet.
- Modifying the request or response headers or data (or both) by providing a custom version of the object that wraps the real request or response objects.
- .Performing some action before the servlet is invoked, after it completes, or both (for example, logging).
- Intercepting a servlet after the servlet is called.
- Blocking a servlet from being called.

By default, the Create Servlet Filter wizard available from the Web Tier Servlets category in the New Gallery creates a filter that dynamically intercepts requests and responses to transform or use the information contained in the requests or responses.

11.5.1.2 What You May Need to Know About Servlet Listeners

A listener can be used to monitor and react to events on a servlet's life cycle by defining listener objects whose methods get invoked when life cycle events occur. Application event listeners are classes that implement one or more of the servlet event listener interfaces. Servlet event listeners support notification for state changes in the `ServletContext` and `HttpSesion` objects, specifically:

- Servlet context listeners are used to manage resources or state held at a VM level for the application.
- HTTP session listeners are used to manage state or resources associated with a series of requests made into a web application from the same client or user.

You can have multiple listener classes listening to each event type and specify the order in which the container invokes the listener beans for each event type.

The Create Servlet Listener wizard available from the **Web Tier > Servlets** category in the New Gallery creates a new listener you can use with your servlet or other web components; you can run this wizard multiple times to create additional listeners.

11.5.1.3 How to Generate an HTTP Servlet

1. In the Applications Navigator, select the web deployment descriptor in the `WEB-INF` folder.
2. Right-click and choose **Open.** The file opens in an XML editor.

1. In the Application Navigator, select the project in which you want to create the new servlet.
2. From the main menu, choose **File > New**, or right-click and choose **New**. The New Gallery opens.
3. In the Categories tree, select **Web Tier**.
4. In the Items list, double-click **HTTP Servlet** to launch the Create HTTP Servlet wizard.

This wizard will create the servlet for you based on information you specify, including the methods and parameters for the servlet. Click the **Help** button to obtain context-sensitive help in the wizard panels.

A simple servlet is generated and appears in your active project. The deployment descriptor file web.xml is also added to your project. The deployment descriptor file is used by the Integrated WebLogic Server in JDeveloper when you run the servlet.

11.5.2 Implementing Basic Methods for an HTTP Servlet

When you use the Create HTTP Servlet wizard to create an HTTP servlet, the wizard creates a Java class for the servlet. This class contains an initialization method and the HTTP methods you specified for the servlet when using the wizard. To customize the servlet, you must implement the servlet's HTTP methods.

The following methods are available from the Create HTTP Servlet wizard:

- `doGet` handles `GET`, conditional `GET`, and `HEAD` requests.
- `doPost` handles `POST` requests.
- `doPut` handles `PUT` requests.
- `doDelete` handles `DELETE` requests.
- `service` handles `Service` requests.

JDeveloper creates skeleton code for these methods. These methods take two objects as arguments `HttpServletRequest` and `HttpServletResponse`. You can also pass in additional parameters and get them programmatically by calling the `ServletRequest.getParameter` method within your servlet's Java code.

11.5.2.1 How to Use the HTTPServletRequest Object

The first HTTP argument in a basic servlet method is an `HttpServletRequest` object. This object provides methods to access

- HTTP header data, including cookies found in the request.
- The HTTP method used to make the request.
- The arguments sent by the client as part of the request.

The methods you call when implementing your servlet methods depend on the kind of HTTP request the servlet will receive. Table 11–17 summarizes the relationship between the possible kinds of HTTP requests and the corresponding methods you should use when implementing your servlet methods.

Table 11–17 Types of HTTP Requests

Possible Client HTTP Requests	Corresponding Client Data Access Methods and Techniques to Use in Your Servlet Code
Any HTTP request	Use the `getParameter` method to get the value of a named parameter. Use the `getParameterNames` method to get the parameter names. Alternatively, you can manually parse the request. You should use either the `getParameter` method or one of the methods that allow you to parse the data yourself. You can not use them together in a simple request. To retrieve cookies from the request, you can use the `getCookies` method.
HTTP GET request	Use the `getQueryString` method to return a `String` to be parsed.
HTTP POST, PUT, and DELETE requests	In general, use the `BufferedReader` returned by the `getReader` method for text data. For binary data, use the `ServletInputStream` returned by the `getInputStream` method.

11.5.2.2 How to Use the HTTPServletResponse Object

The second HTTP argument in a basic servlet method is an `HttpServletResponse` object. This object encapsulates the information from the servlet to be returned to the client. This object supports the following ways of returning data to the client:

- A writer for text data (via the `getWriter` method)
- An output stream for binary data (via the `getOutputStream` method)

You can also send a cookie in the response using the `addCookie` method.

To change the HTTP Response Type:

By default, the Create HTTP Servlet wizard creates a servlet that dynamically generates HTML content (MIME type: `text/html`). You can change to another MIME type by selecting the desired type from the **Generate Content Type** dropdown in the Create HTTP Servlet wizard. The wizard adds the `setContentType` method in the servlet's Java file with the selected type to set. For example, if you choose the XML content type, the wizard generates:

```
public class HelloWorld extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/xml; charset=windows-1252";
  private static final String DOC_TYPE;
  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
  }
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
  {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    if (DOC_TYPE != null)
    {
      out.println(DOC_TYPE);
    }
    out.close();
  }
}
```

11.5.3 How to Create a Servlet Filter

The Create Servlet Filter wizard available from the **Web Tier - Servlets** category in the New Gallery creates a new filter you can use to process requests or responses to or from your servlet or JavaServer Page.

To create a servlet filter:

1. In the Application Navigator, select the project in which you want to create the new servlet listener, usually the project which includes your servlet or JSP.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **Servlets**.
4. In the **Items** list, double-click **Servlet Filter** to open the Create Servlet Filter wizard.

The Create Servlet Filter wizard will create the servlet filter for you based on information you specify, including the implementation class and initialization parameters. Press F1 or click **Help** to obtain context-sensitive help in the wizard.

A new servlet filter is generated and appears in your active project. The deployment descriptor file `web.xml` is updated with the `<filter>` element. The deployment descriptor file is used by the Integrated WebLogic Server in JDeveloper when you run the servlet or JSP.

11.5.4 How to Create a Servlet Listener

The Create Servlet Listener wizard available from the Web Tier - Servlets category in the New Gallery creates a new listener you can use with your servlet or other web components.

To create a servlet listener:

1. In the Application Navigator, select the project in which you want to create the new servlet listener, usually the project which includes your servlet or other web component.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **Servlets**.
4. In the **Items** list, double-click **Servlet Listener** to open the Create Servlet Listener wizard.

The Create Servlet Listener wizard creates the servlet listener for you based on information you specify, including the implementation class and interface. Press F1 or click **Help** to obtain context-sensitive help in the wizard.

A new servlet listener is generated and appears in your active project. The deployment descriptor file `web.xml` is updated with the `<listener>` element. The deployment descriptor file is used by the Integrated WebLogic Server in JDeveloper when you run the servlet.

11.5.5 Registering a Servlet Filter in a JSP Page

The Create Servlet Filter wizard available from the Web Tier category in the New Gallery creates a new filter you can use to process requests or responses to or from your JavaServer Page.

To register a servlet filter in a JSP page:

1. In the Application Navigator, select the project in which you want to create the new servlet listener, usually the project which includes your JSP.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **Servlets**.
4. In the **Items** list, double-click **Servlet Filter** to open the Create Servlet Filter wizard.

   This will start the Create Servlet Filter wizard which will create the servlet filter for you based on information you specify, including the implementation class and initialization parameters. Press F1 or click **Help** to obtain context-sensitive help in the wizard panels.
5. Click **Next** if the Welcome page displays.
6. Enter the **Filter Name, Filter Classname** and **Package**. Then click **Next**.

7. Select **Map to Servlet or JSP**, and select the name of the JSP from the dropdown list. Then click **Next**.
8. Click **New**, and enter the name and value for each initialization parameter. Then click **Finish**.

A new servlet filter is generated and appears in your active project. The deployment descriptor file `web.xml` is updated with the `<listener>` element. The deployment descriptor file is used by the Integrated WebLogic Server in JDeveloper when you run the JSP.

11.5.6 How to Run a Servlet

A servlet is a Java program that runs in a Java EE application server. Think of a servlet as the server-side counterpart to a Java applet. The Integrated WebLogic Server is responsible for running servlets in JDeveloper.

As an alternative to running your servlets inside the Integrated WebLogic Server, your servlet can contain a main() routine that lets you run the servlet class as an application. That declaration is: `public static void main(String[] args)`

This is useful when you want to test servlet classes without running under the Oracle WebLogic Server.

To run a servlet:

After building your servlet, you can run it by executing the run command in one of the following ways:

1. In the Navigator, select the Java file containing your servlet that you want to run.
2. Run a servlet in any of these ways:
   - Choose **Run** from the main menu.
   - Right-click the Java file containing your servlet and choose **Run**. **<servletname>.java** (and the desired option for running when more than one way to run exists) from the context menu.
   - Select the Java file containing your servlet and click **Run** on the toolbar.
3. If you set up your servlet to run as an application, use the dialog to select the way you want to start the target servlet:
   - **As an Application**: The servlet is launched as a standalone Java application.
   - **In Integrated WebLogic Server**: the embedded server is started and the servlet is run in the server.

   Select the option you desire, then click **OK**.

JDeveloper performs the following functions when a servlet is run in Integrated WebLogic Server:

- Compiles the servlet source code.
- Starts the embedded Integrated WebLogic Server process.
- Runs the resulting classes directly from the output project directory.
- Edits the embedded Integrated WebLogic Server web.xml file to include the servlet name and class information.
- Invokes the servlet in your default Web browser. For example, your browser is launched as follows:

```
http://<your_machine_IP_address>:<http_port>/<context_
root>/servlet/<servlet_full_class_name>
```

For example :

```
http://127.0.0.1:8988/Project1-context-root/servlet/package1.
Servlet1
```

11.5.7 How to Debug a Servlet

You can debug a servlet using the embedded Integrated WebLogic Server in JDeveloper. The Debug command attempts to debug the selected Java file containing your servlet. In JDeveloper, you can set breakpoints within servlet source code and the debugger will follow calls from servlets into JavaBeans.

To debug a servlet:

1. Select the servlet Java file in the Navigator and select **Debug | Debug <project_name>** from the JDeveloper main menu, or click the **Debug** icon. Alternatively, right-click the servlet Java file and choose **Debug**.

   When you debug a servlet, JDeveloper opens the default Web browser and invokes the servlet.

2. Debug your servlet by setting breakpoints as you would any other Java application.

3. When you are finished running and testing the servlet, you can terminate the server by choosing **Run | Terminate - Integrated WebLogic Server** from the main menu.

JDeveloper performs the following functions when a debugging a servlet:

- Compiles the servlet source code.
- Starts the Integrated WebLogic Server process.
- Runs the resulting classes directly from the output project directory.
- Invokes the servlet in your default Web browser. For example, your browser is launched as follows:

  ```
  http://<your_machine_IP_address>:<http_port>/<context_
  root>/servlet/<servlet_full_class_name>
  ```

  For example :

  ```
  http://127.0.0.1:8988/Project1-context-root/servlet/package1.
  Servlet1
  ```

11.5.8 How to Deploy a Servlet

JDeveloper supports deploying your Servlet applications on any Java EE application server through the creation of a Web Module Archive (WAR).

For more information, see the *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

11.6 Developing Applications with Script Languages

JDeveloper provides scripting functionality including support for basic JavaScript when working with JSP and HTML pages. This section discusses support in Oracle

JDeveloper for script languages, how to work with script languages, and how to refactor JavaScript code.

11.6.1 Script Language Support in JDeveloper

JDeveloper supports script languages, specifically JavaScript and JSON, by offering code insight, breadcrumb support, and the JDeveloper structure pane. JDeveloper JavaScript Code Insight completes labels, variables, parameters and functions when typing inside a script region, or inside an HTML event handler. Breadcrumb support displays the location of a selected JavaScript function in the hierarchy as you work on the file. And the Structure Pane shows the hierarchy of functions defined in the file, and also of the variables defined in the functions.

11.6.1.1 How to Work with JavaScript Code Insight

The JDeveloper JavaScript Code Insight completes labels, variables, parameters and functions when typing inside a script region, or inside an HTML event handler.

The JavaScript Code Insight feature displays a dynamic list of possible completions for a given JavaScript function at the bottom of the editing pane. As you type, the Code Insight feature will display a list of possible values appropriate to the values you have already typed. To see a list of possible entries that have already been used or defined in your project, click on the drop-down arrow and then select Show.

JavaScript Code Insight is available when editing an `.html`, `.jsp`, or `.jspx` source file, or an included `.js` file for both user-defined and built-in JavaScript functions. The assist window displays any referenced `.js` files as well as any `.js` file in the project not yet included.

In a normal JavaScript Code Insight invocation, a large list of potential properties may display. However, you can invoke JavaScript Smart Code Insight in the source editor by pressing **Ctrl-Alt-space**. JDeveloper then attempts to smartly figure out the type of the object on which insight is invoked, and then show only those properties.

The JavaScript Code Insight feature creates templates for the following elements.

To invoke Code Insight:

Type the JavaScript element or its abbreviation:

- `case`
- `for`
- `foreach`
- `if`
- `ife (if-else)`
- `sw (switch)`
- `wh (while)`
- `fori` (for loop with range)
- `try`
- `trycf`
- `tryf`
- `al` (alert)
- `fn` (function)

- `fne` (function-expression)
- `dne` (do-while loop)

JavaScript Code Insight is DOM-based and browser-aware, displaying one or more browser icons for Internet Explorer, Mozilla, or Safari, to indicate browser support for a method or variable.

11.6.1.2 How to Use Breadcrumb Support

When you are editing a JavaScript file in the Source Editor and have the cursor located in a function, JDeveloper displays a breadcrumb trail in the lower margin of the Source Editor window.

This breadcrumb trail shows the position of this function in the JavaScript hierarchy, along with its subelements such as methods, parameters, and such. JDeveloper also displays breadcrumbs for `if`, `if-else`, `do`, `while`, `for`, and `try/catch/finally` (just as it does for Java).

To explore available functions within the hierarchy:

- From the breadcrumb trail, click on a dropdown (at the file level) to go into the functions defined within that parent.

11.6.1.3 How to Use Structure Pane Support

While you are editing a JavaScript file, JDeveloper tracks the location in the structure of the project or application you are building and displays it in the Structure Pane.

The Structure Pane shows the hierarchy of functions defined in the file, and also of the variables defined in the functions.

To find a location in the code editor from the Structure Pane:

- Double-click any element in the Structure pane to take your focus to the corresponding place in the code editor. If there are errors in the file, they also show up in the Structure Pane.

11.6.2 Working with Script Languages

Working with script languages not only includes the direct use of script elements inside an HTML or JSP page, but also involves using references to script files which are associated with the overall application.

The JDeveloper code editor provides a syntax highlighting feature which assists in determining the proper code for a script or script-language element.

Other elements of working with script languages include creating a JavaScript Object Notation (JSON) file.

11.6.2.1 How to Create a Script

You can create a client-side script to include or embed in an HTML or JSP page.

To create a script in JDeveloper:

1. If not already done, open a JSP or HTML page by double-clicking its icon from the Application Navigator.
2. In the Component Palette, select the HTML palette, Common page from the dropdown list.

3. In the Source editor or Structure window, place your cursor in the location where you want to create the script and select the Script element. Alternatively, drag the Script element to the desired location on the HTML or JSP page.
4. In the Script dialog, either enter the location of an external script file, or select the scripting language (`text/javascript`, `text/tcl`, `text/vbscript`) and enter the script code. For additional assistance, press F1 or click **Help** in the dialog.
5. Click **OK**.

A script element that references the external script file is embedded in the page similar to the following:

```
<script language="JavaScript" src="foo.js"></script>
```

or

The script code is embedded in the page and visible from the Source editor similar to the following:

```
<SCRIPT type="text/vbscript">
  <!--
   >Sub foo()
   ...
   End Sub
   ' -->
   </SCRIPT>
```

11.6.2.2 How to Add Script Language Elements to an HTML or JSP Page

JDeveloper provides basic JavaScript support when working with HTML and JSP pages. In addition to drag and drop support, you can change the text presentation of the JavaScript code in the Java Code Editor and associate file extensions for JavaScript file recognition in JDeveloper.

To insert a JavaScript into a JSP or HTML page:

1. Choose **File > New**.
2. Select the **Web Tier** category.
3. In the Items list, select **JavaScript File**.
4. In the Create JavaScript File dialog, enter a name and location for the JavaScript (`.js`) file.
5. In the Java Code Editor for the JavaScript file, enter the JavaScript code and save it.

   The JavaScript file appears in the Application Navigator below the HTML or JSP project's Web Content folder.
6. If not already done, open the HTML or JSP page in the JSP/HTML Visual Editor.
7. From the Application Navigator, drag a JavaScript onto the page where appropriate. If you drag a JavaScript from the Component Palette, you are prompted to copy the JavaScript file to the current project's document root.

   JDeveloper creates a script element that references the JavaScript file.

Note: You can also import a JavaScript file into the project.

11.6.2.3 How to Set Syntax Highlighting

Syntax highlighting is a JDeveloper feature that lets you more easily identify syntax elements (such as brace matching) while you are editing Java, JavaScript, and JSON files.

To set syntax highlighting options for JavaScript in the Code Editor:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, expand the Code Editor node.
3. Select the Syntax Colors node.
4. For the Language category, select **JavaScript**.

   The display on the page changes to reflect the JavaScript style settings.
5. Change any of the available style settings as appropriate.
6. Click **OK**.

   For detailed help on any field, press F1 or click **Help**.

When you return to work in the Java Code Editor, JavaScript syntax is highlighted according to these style settings.

11.6.2.4 How to Associate JavaScript File Extensions

By default, JDeveloper recognizes files with the `.js` file extension as JavaScript. You can associate any other file extension for JDeveloper to recognize.

To add or remove file extensions for JavaScript file recognition in JDeveloper:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select **File Types**.
3. In the Recognized File Type list, select the JavaScript Source node.
4. The `.js` file extension is associated.

   Click **Add** to create a JavaScript file extension.

   In the Add Extension dialog, enter the file extension you want to associate to a JavaScript file.
5. Click **Remove** to delete a file association.
6. Click **OK**.

   For detailed help on any field, press F1 or click **Help**.

When you open a file with any of these extensions, JDeveloper recognizes the file as JavaScript.

11.6.2.5 How to Create a JSON File

You can create a JSON (JavaScript Object Notation) file in JDeveloper. A JSON file allows you to pass structured data easily between applications or between files within an application, in a lightweight format that is easily readable by humans and easily interpreted by dozens of programming languages.

To create a JSON file:

1. Select **File > New > Web Tier > HTML > JSON File**.
2. Supply the following data about your file:

File Name

The name of your JSON file. By default, this is `untitled.json`. The `.json` extension makes it possible for other parsers to read the JSON format of the data inside your file.

Directory

The pathname in your local file system for storing the JSON file.

Browse

Opens the file system browser for selecting a path in your local file system.

3. Click **OK**.

The JSON file is now available to be edited in JDeveloper. Use the normal functions of the JavaScript editor to add content.

11.6.3 Refactoring JavaScript Code

JDeveloper provides support for renaming references to a function or variable. JDeveloper also replaces all occurrences of function names with the new name when you perform delete operations. This method of renaming and replacing function names is known as refactoring.

Refactoring is an editing technique that modifies code structure without altering program behavior. A refactoring operation is a sequence of simple edits that transforms a program's code but keeps it in a state where it compiles and runs correctly. JDeveloper provides a collection of automated refactoring operations for JavaScript code and files.

Use refactoring when you modify a program's source code to make it easier to maintain, extend, or reuse. Perform the modification as a series of refactoring steps. After each step you can rebuild and revalidate the program to insure that no errors have been introduced.

JDeveloper supports these refactoring operations for JavaScript code and files:

- Renaming references to a function, variable, or label. Each occurrence of the function or variable name is replaced by the new name.
- Safe deletion. The definition of the function is replicated, and all occurrences of the function name in the replicated definition are replaced by the new name.

11.6.3.1 Finding Usages of Code Elements

You can search within a JavaScript file for specific usages of code elements such as functions, variables and labels. This allows you, when refactoring, to determine where an element is used so that you can safely change it, or choose not to.

To search in a JavaScript file for a function, variable or label:

1. Place the cursor inside the function, variable or label you wish to search for and click the right mouse button.
2. Select **Find Usages**.

JDeveloper will search through the JavaScript file for the element you have selected.

You can make two optional selections while searching for the element:

Search in Comments
Select this if you want JDeveloper to search inside comments for the variable, label or function name. This can be useful if you have commented out a section of code that you plan to restore at a later date, or if you simply want to ensure that the comments reflect the updated name of the element involved in the refactoring.

New tab
Select this if you want JDeveloper to display the results of the search in a new tab. If you do not select this, JDeveloper displays the results in the Log window.

11.6.3.2 Renaming a JavaScript Code Element

While working with JavaScript code you can easily rename the definition and all references to a function or variable. If you wish, you can first generate a preview — a list of the usages that will be replaced. Use the preview to inspect and modify or exclude selected usages, before causing the rest to be renamed.

The scope of a renaming operation is the full scope of the element in the project. Function usages are replaced anywhere they appear in the project. Variables are renamed only in the lexical scope of their definitions; other elements with the same name are not modified.

By default, the operation will be run on JavaScript files, excluding comments (but not documentation comment tags that name code elements) and annotations. Usages that are not ambiguous will be replaced.

To rename a code element:

1. Select the element that is to be renamed:
   - In a JavaScript editor, select the function or variable name.

     or
   - In a script in an JSP or HTML page, select the function or variable name.
2. Invoke the command:
   - From the Main menu or the context menu, choose **Refactor > Rename**.

     or
   - Press **Ctrl+Alt+R**.
3. In the **Rename To** box, enter the new name. The name must be valid and not already in use.
4. Select **Search in Comments** to extend the operation to comments, the bodies of documentation comments, and to annotations.
5. Select **Preview** if you wish to inspect the usages that will be replaced before committing to the renaming operation.
6. Click **OK**. If you selected **Preview**, finish the renaming operation from the Preview Log window. Otherwise, all usages will be modified.

11.6.3.3 Deleting a JavaScript Code Element

While developing your JavaScript code, you can safely delete the definition of a function, label or variable. The deletion will not be performed without your confirmation if the element is still in use.

If the element is in use, a log showing the usages will be displayed. Use the list to inspect and resolve the usages. If you then confirm the deletion, any remaining usages will remain in the code as undefined references.

To delete a code element:

1. Select the element that is to be deleted:
   - In a JavaScript editor, select the function, label or variable name.

     or

   - In a script in a JSP or HTML page, select the function, label or variable name.
2. Invoke the command:
   - From the Main menu or the context menu, choose **Refactor > Delete Safely**.

     or

   - Press **Alt+Delete**.

   The Delete Safely dialog will open while the project files are searched for usages.
3. If the dialog closes, the element has been deleted. If it remains open after performing its search, the element has unresolved usages.
   - Click **View Usages** to inspect and resolve the usages. When finished, invoke the command again to delete the element.

     or

   - Click **OK** to delete the element's definition.

11.6.3.4 How to Preview a Refactoring Operation

When performing a refactoring operation that may modify many usages, it is useful to preview the refactoring to identify those usages that should be modified by hand or be excluded. You have the option, before committing these operations, of having usages listed in the Preview Log window, from which you can inspect and resolve them. Once you have confirmed the modifications, you can commit the operation.

The log displays a collapsible tree of packages and Java files. Under each file, the log displays lines of code containing modified usages. For more information about the Preview window, press F1.

To view a usage in an Edit window:

- Double-click the entry in the log.

To exclude a usage from the refactoring operation:

- Right-click the usage, and then select **Exclude**.

To commit the refactoring operation:

1. If you have made any edits that affect usages, click the **Refresh** button in the log toolbar to rerun the usages search.
2. Click the **Do Refactoring** button in the log toolbar.

11.6.3.5 How to Reformat JavaScript Code

Often when editing JavaScript, you can lose sight of the initial scheme for indentations, braces, and other visual cues that help you maintain a sense of the scope of the operation you are editing and where it fits in the overall structure of the

function. To aid clarity, JDeveloper can reformat your JavaScript code, causing parallel elements to line up and make it easier for you to find visual cues to the parts of the function you are editing. In addition, reformatting removes extraneous line breaks and other whitespace from the JavaScript, rendering it more compact, which can improve the efficiency of deployment by reducing file size.

To reformat a section of JavaScript code:

1. Place the cursor inside the section of code to be reformatted and click the right mouse button, or select a snippet of JavaScript code to be reformatted.
2. Select **Reformat**.

The selected section of JavaScript code is reformatted. When you save the file, the code will be saved in the new format.

11.6.3.6 How to Change Code Formatting Preferences

You can customize the code editor look and feel, general behavior, and Code Insight and Java Insight options.

To change code formatting preferences

- From the main menu, select **Tools** > **Preferences** > **Code Editor**.

> **Note:** From this dialog, you can also choose options for editing Java files in the Java source editor. Your selections apply to JavaScript as well as Java files.

11.6.3.7 How to Use Code Folding

You can also reformat a `.js` file if you have made modifications that affect readability or file size. In addition, code folding can help with readability, as it lets you concentrate only on specific areas of the file by "folding" selected logical elements (such as function definitions) of the file. When folded, only the initial few key words of the code element (such as the name of the function being defined) are displayed; the rest are indicated by ellipsis (...) after the initial keywords.

To use code folding:

- Click on the - sign to the left of the first column of text in the JavaScript editor.

  This folds the code in the selected element, and changes the - sign to a +.

To unfold a section of code:

- Click on the + sign.

  Note that all JavaScript code formatting and highlighting features, as well as code folding, also apply if you are editing or creating a JSON file.

11.6.3.8 How to Refactor and Move a File

When you move a file, references to that file need to change throughout your application. JDeveloper helps with this task during refactoring by changing references in the `<script src=...>` tag.

To refactor and move a JavaScript function:

1. Right-click on the file in the Application Navigator to be refactored and moved, and then select **Refactor Move**.

2. Enter the new name for the file into which you wish the function to be moved.
3. Click on **Do Refactoring** in the Rename log window.

On completion of the refactor, JDeveloper updates the `<script src=...>` tag in all HTML files affected by the refactoring.

11.7 Working with JSP and Facelet Tag Libraries

JDeveloper supports both JSP 2.0 and JSP 1.2, and Facelet 2.0 custom tag libraries, which enable the development of reusable modules called custom actions. Form processing, accessing databases or email, and flow control are examples of tasks that can be performed by custom actions. To invoke a custom action, you use a custom tag inside a JSP page. A collection of custom tags forms a custom tag library. A tag library descriptor (`.tld`) file is an XML document that describes your tag library and each tag in it.

11.7.1 How to Use Tag Libraries with Your Web Pages

There are several tools to simplify the task of creating new JSP or facelet custom tag libraries as well as importing and registering custom tag libraries from another source. Custom tag libraries are supported by JDeveloper Code (tag) Insight and can be added to the Component Palette. When working with custom tag libraries you can create custom tag libraries and tags. Register custom tag libraries in order to invoke Code (Tag) Insight for the tags while you are editing pages in the Java Code Editor. Add customized pages to the Palette to display the available tags on the Palette while you are editing pages.

The tags are common to many JSP or facelet applications. There is support for core iteration and control-flow features, text inclusion, internationalization-capable formatting tags, and XML-manipulation tags. Such standardization lets you learn a single tag and use it on multiple containers for easy recognition and optimization across containers. Using the expression language (EL) and a set of four standard tag libraries, JSTL lets you develop dynamic, Java-based web sites.

With JSTL, using the Business Components Data Tag library is simpler since tags such as `<jbo:showvalue>` and `<jbo:rowsetiterate>` are no longer required. Instead of spending time on coding these common operations, you can focus on developing tags and web pages that are specific to your own web application project.

You can manage your libraries, including locating the source for your tag libraries by going to **Tools > Manage Libraries > JSP Tag Libraries** or **Facelets Tag Libraries**.

Tag support includes these custom tag libraries that you can use to create JSP or Facelet pages:

- **JSTL Core**. This tag library provides tags related to expressions, flow control, and a generic way to access URL-based resources whose content can then be included or processed within the JSP page.
- **JSTL Format**. This tag library provides tags that support I18N and localized formatting and parsing.
- **JSTL SQL**. This tag library provides tags that allow direct database access from within JSPs.
- **JSTL XML**. This tag library provides tags that allow parsing and XSL transformation of XML documents.

- **Facelets 2.0.** This tag library provides tags that allow you to create, manage and handle UI components within a web page. For more information see the Facelets Tag Library documentation at: http://myfaces.apache.org/core20/myfaces-impl/tlddoc-facelets/ui/tld-summary.html
- **Trinidad Components 2.0.** For more information , see the Apache Trinidad page at: http://myfaces.apache.org/trinidad/index.html.

After you create a custom tag library, you can reuse it in other applications you are developing. JDeveloper includes a tag library as part of a deployment descriptor when you use it in an application.

To add, delete, or edit project level tag libraries

1. Choose **Application > Default Project Properties > JSP Tag Libraries**
2. Add, delete, or edit project tag libraries as necessary.

To browse to a JSP tag library descriptor (TLD) file:

1. In the Java Code Editor, right-click anywhere in the tag library declaration for the TLD file you want to browse. The tag library declaration begins with `<%@ taglib`.
2. From the context menu, choose **Browse Tag Library.** The JSP tag library descriptor file opens in another Java Code Editor window.

To browse pages or individual JSP tags:

1. Choose **Tools > Configure Palette** to open the Configure Component Palette dialog.
2. Select JSP as the **Page Type**, then click **OK**.
3. Select the page name in the **Pages** list.
4. Click Remove below this list to remove the page and all tags on the page. Or, select an individual tag name in the **Components** list and click **Remove** below this list to remove only the tag.
5. Click **Yes** when you see either the **Delete tags?** or **Remove items?** prompt.
6. After completing your changes, click **OK** to close the Configure Component Palette dialog.

11.7.2 How to Work with Custom Tag Libraries

To create a custom tag library, you will create the tag library descriptor file and then create simple tags or component tags. A tag library descriptor file (TLD) is an XML document that describes your tag library and each tag in it. It is used by a container to validate the tags. Once you create tags, you can add attributes and scripting variables to them.

To create a custom JSP or facelets tag library:

1. In the Application Navigator, select the project in which you want to create the new tag library.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **JSP or JSF/Facelet**.

4. In the **Items** list, double-click **JSP or Facelet Tag Library** to open the Create Tag Library wizard.
5. After completing the required information for creating a new tag library, click **Finish**.

To add pages and JSP or facelets tags to the Component Palette:

1. Choose **Tools > Configure Palette** to open the Configure Component Palette dialog.
2. Click **Add** under the Pages list to open the New Palette Page dialog.
3. Enter a **Page Name** and select JSP or facelet as the **Page Type**, then click **OK**. Your new page name is added to the bottom of the list of pages in the Configure Component Palette dialog.
4. Select the new page name in the **Pages** list and click **Add** under the Components list to open the **Add Tag Component** dialog.
5. Click **New** to open the JSP Library Manager dialog. A new JSP tag library descriptor file, **newTagLib**, is added to the **Tag Libraries** tree.
6. Enter the custom tag library descriptor (TLD) file, the location of the JAR or ZIP archive file, the URI, and prefix for the new tag library. The prefix that you enter will be updated on the **Tag Libraries** tree, replacing **newTagLib**, after you click OK.
7. Click **OK** to close the JSP Library Manager dialog. The new JSP tag library descriptor file is added to the **Tag Libraries** tree in the Add JSP Tag Component(s) dialog.
8. Under **Select a tag to add** in the **Tag Libraries** node, select the tag library that you want to add or expand the tag library node to display the tags included in it.
9. To add all the tags, select the tag library name in the tree and click **OK**. To add an individual tag, select a single tag name in the tree and click **OK**. To add multiple tags, use Ctrl-click or Shift-click to select them and click **OK**.
10. Click **Yes** when you see the **Install tags?** prompt.
11. After adding tags, click **OK** to close the Configure Component Palette dialog. The name of the new page you added displays in the dropdown list in the Palette. All the tags you added display with angle brackets (< >) as the icon, if you accepted the default icon. If you do not see any tag names on the Palette, right-click in the Palette and choose **List View.**

To modify a custom TLD file using the Tag Library Descriptor property editor:

1. In the Structure window, select the TLD file you want to modify.
2. Right-click the file and choose **Properties**. The Tag Library Descriptor property editor is displayed.
3. After completing your changes, click **OK**.

To edit a TLD file in the XML Source Editor:

1. In the Navigator, double-click or right-click a file and choose **Open.** Click the Source tab if not selected by default for that file. While you are typing, you can invoke Code Insight by pausing after typing the < (opening bracket) or by pressing Ctrl+Space (if you are using the default keymapping). Code Insight opens a list with valid elements, based on the grammar.

2. After selecting an element, enter a space and then either pause or press Ctrl+Space to open a list of valid attributes from which you can select. After you enter the opening quote for the attribute value, either the required type of value or a list of available values is provided.

   Tip: To edit a TLD file with the Component Palette, choose **View > Component Palette** to open the Palette and select **Tag Lib** or one of the available pages from the dropdown list. Then choose elements from the page.

To register a JSP or facelet custom tag library:

1. Choose **Tools > Manage Libraries** to open the Manage Libraries dialog.
2. Select the **JSP Tag Libraries** or **Facets Tag Libraries** tab.
3. Click **New** to add a new JSP tag library descriptor file to the **JSP Tag Libraries** or **Facelets Tag Libraries** tree.
4. Enter the custom tag library descriptor (TLD) file, the location of the JAR or ZIP archive file, the URI, and prefix for the new tag library. The prefix that you enter will be updated on the **JSP Tag Libraries** or **Facelets Tag Libraries** tree after you click **OK**.
5. Click **OK** to close the Manage Libraries dialog.

To add a scripting variable to a tag:

1. In the Application Navigator, select the `Tag.java` or `WebTag.java` file.
2. Right-click the tag and choose **Add Scripting Variable**. The Add New Tag Scripting Variable dialog opens.
3. After completing the required information for adding a scripting variable, click **OK**. The new variable.java file that defines the attributes is created and opened in the Java Code Editor. The new scripting class is also added to the pre-existing tag handler class.

To deploy your custom JSP tag library or facelets tag library as a JAR File:

1. In the Application Navigator, select the Deploy file you want to deploy.
2. Right-click the file and choose **Deploy to JAR File**. By default, the tag library is deployed in the current project directory.

12

Developing with EJB and JPA Components

This chapter describes how to use JDeveloper tools to build the business tier of a J2EE enterprise application using Enterprise JavaBeans (EJB) 3.0 and Java Persistence API (JPA) components.

This chapter includes the following sections:

12.1 About Developing with EJB and JPA Components

JDeveloper includes step-by-step wizards for creating EJB projects, entities, persistence units, session beans, and message-driven beans. You can build entities from online or offline database definitions and from application server data source connections. There is also seamless integration with JPA and TopLink technology to provide a complete persistence package.

12.2 Support For EJB Versions and Features

JDeveloper supports EJB 3.0, as well as versions 1.0 through 2.1. The current JDeveloper documentation, including this chapter of the *User Guide* and the embedded online help, focus on EJB 3.0 development tasks.

Previous versions of the JDeveloper documentation tell how to work with EJB 2.1 and earlier. For those versions of the documentation, see `http://www.oracle.com/webapps/online-help/jdeveloper/10.1.3` and search for and navigate to the topic "Developing Enterprise JavaBean Components." Be aware that EJB application development interfaces may change from version to version, and some historical help content will be outdated for the current version.

For the complete EJB Java Community Process specifications and documentation for all versions, see the Oracle Technology Network at http://www.oracle.com/technetwork/java/docs-135218.html.

> **Note:** If you are using EJB 3.0, you may be using annotations instead of some deployment files. Include deployment descriptors to override annotations or specify options not supported by annotations.

Supported EJB 3.0 Features

The key differences between EJB 3.0 and previous versions are:

- **Simplified EJBs** - EJB 3.0 eliminates the need for home and component interfaces and the requirement for bean classes for implementing `javax.ejb.EnterpriseBean` interfaces. The EJB bean class can be a pure Java class (POJO), and the interface can be a simple business interface. The bean class implements the business interface.
- **Use of Annotations Instead of Deployment Descriptors** - Metadata annotation is an alternative to deployment descriptors. Annotations specify bean types, different attributes such as transaction or security settings, O-R mapping and injection of environment or resource references. Deployment descriptor settings override metadata annotations.
- **Dependency Injection** - The API for lookup and use of EJB environment and resource references is simplified, and dependency injection is used instead. Metadata annotation is used for dependency injection.
- **Enhanced Lifecycle Methods and Callback Listener Classes** - Unlike previous versions of EJB, you do not have to implement all unnecessary callback methods. Now you designate any arbitrary method as a callback method to receive notifications for lifecycle events. A callback listener class is used instead of callback methods defined in the same bean class.
- **Interceptors** - An interceptor is a method that intercepts a business method invocation. An interceptor method is defined in a stateless session bean, stateful session bean, or a message-driven bean. An interceptor class is used instead of defining the interceptor method in the bean class.
- **Simple JNDI Lookup of EJB** - Lookup of EJB is simplified and clients do not have to create a bean instance by invoking a `create()` method on EJB and can now directly invoke a method on the EJB.

Session Beans

- **Simplified Beans** - Session beans are pure Java classes and do not implement `javax.ejb.SessionBean` interfaces. The home interface is optional. A session bean has either a remote, local, or both interfaces and these interfaces do not have to extend `EJBObject` or `EJBLocalObject`.
- **Metadata Annotations** - Metadata annotations are used to specify the bean or interface and run-time properties of session beans. For example, a session bean is marked with `@Stateless` or `@Stateful` to specify the bean type.
- **Lifecycle Methods and Callback Listeners** - Callback listeners are supported with both stateful and stateless session beans. These callback methods are specified using annotations or a deployment descriptor.

- **Dependency Injection** - Dependency injection is used either from stateful or stateless session beans. Developers can use either metadata annotations or deployment descriptors to inject resources, EJB context or environment entries.
- **Interceptors** - Interceptor methods or interceptor classes are supported with both stateful and stateless session beans.

Message-Driven Beans (MDBs)

- **Simplified Beans** - Message-driven beans do not have to implement the `javax.ejb.MessageDriven` interface; they implement the `javax.jms.MessageListener` interface.
- **Metadata Annotations** - Metadata annotations are used to specify the bean or interface and run-time properties of MDBs. For example, an MDB is marked with `@MessageDriven` for specifying the bean type.
- **Lifecycle Methods and Callback Listeners** - Callback listeners are supported with MDBs. These callback methods are either specified using annotations or the deployment descriptor.
- **Dependency Injection** - Dependency injection is used from an MDB. You either use metadata annotations or deployment descriptors to inject resources, EJB context, or environment entries used by an MDB.
- **Interceptors** - Interceptor methods or interceptor classes can be used with MDBs.

Entities - Java Persistence API (JPA)

- **Simplified Beans (POJO Persistence)** - EJB 3.0 greatly simplifies entity beans and standardizes the POJO persistence model. Entity beans are concrete Java classes and do not require any interfaces. The entity bean classes support polymorphism and inheritance. Entities can have different types of relationships, and container-managed relationships are manually managed by the developer.
- **Entity Manager API** - EJB 3.0 introduces the `EntityManager` API that is used to create, find, remove, and update entities. The `EntityManager` API introduces the concept of detachment/merging of entity bean instances similar to the Value Object Pattern. A bean instance may be detached and may be updated by a client locally and then sent back to the entity manager to be merged and synchronized with the database.
- **Metadata Annotations** - Metadata annotations greatly simplify development of entities by removing the requirement of deployment descriptors. The entity annotation is used to specify a class to be an entity bean. Annotations are used to specify transaction attributes, security permissions, callback listeners and annotated queries.
- **Query Language Enhancements** - EJB 3.0 greatly improves the query capability for entities with Java Persistence Query Language (JPQL). JPQL enhances EJB-QL by providing additional operations such as bulk updates and deletes, JOIN operations, GROUP BY HAVING, projection and sub-queries. Also dynamic queries can be written using EJB QL.
- **Lifecycle Methods and Callback Listeners** - Callback listeners are supported with entity beans. Callback methods are either specified using annotations or a deployment descriptor.

12.3 Building EJB 3.0 Applications and Development Process

JDeveloper includes a complete set of features to set up the EJB business layer of an enterprise application.

You can start by using the step-by-step wizard to create the framework for your EJB web application, setting up the model layer of your enterprise application. You can then use wizards to create entities that correspond to database tables. You can then use a wizard to create session beans and facades and to build a persistence unit. Oracle ADF provides components to enable data controls. When you are ready, you can use the JDeveloper integrated server capabilities to test it.

12.3.1 EJB 3.0 Application Development Process

JDeveloper includes tools for developing EJB applications, as described in the following sections.

- Section 12.3.1.1, "Creating Entities"
- Section 12.3.1.2, "Creating Session Beans and Facades"
- Section 12.3.1.3, "Deploying EJBs"
- Section 12.3.1.4, "Testing EJBs Remotely"
- Section 12.3.1.5, "Registering Business Services with Oracle ADF Data Controls"

12.3.1.1 Creating Entities

Use the entity wizards to create entities or to create entities from tables using online, offline, or application server data source connections. Use the Entity Beans from Tables Wizard to reverse-engineer entities from database tables. In the entity wizards you can select or add a a persistence unit and a database connection, or you can select a database to emulate. You can also select database tables for your entity. For more information, see Section 12.6.2, "How to Create JPA Entities".

You can create entities from existing tables, or manually in the Java Source Editor. If you create entities from existing tables, the mapping is done automatically. If you create entities manually, you have more control over the mapping, but you must code it by hand. You can create entities using wizards or by using an EJB diagram.

12.3.1.2 Creating Session Beans and Facades

You can use session beans to implement the session facade design pattern. A session facade aggregates and presents data, provides a place for business logic, and has a transactional context via the container. For more information, see Section 12.7, "Implementing Business Processes in Session Beans" and Section 12.7.1, "Using Session Facades".

When you create a session bean with the wizard, you have the option of generating session facade methods for every entity in the same project. You can choose which core transactional methods to generate, `get()` and `set()` accessors, and finder methods on the entities. If you create new entities or new methods on entities, you can update your existing session facade by right-clicking it in the Navigator and choosing **Edit Session Facade**.

12.3.1.3 Deploying EJBs

JDeveloper provides Oracle WebLogic Server as a container for deployed EJBs. A JDeveloper server-specific deployment profile is generated by default. You can also

create a WebLogic-specific deployment profile. For more information, see Section 12.9, "Deploying EJB Modules and JPA Persistence Units".

12.3.1.4 Testing EJBs Remotely

JDeveloper can also create a sample client for use with a remote server. You generate the sample client in the same manner as a local client, providing the remote connection details. For more information, see Section 12.10.2, "How to Test EJB/JPA Components Using a Remote Server".

12.3.1.5 Registering Business Services with Oracle ADF Data Controls

ADF provides components for enabling data controls for your entities. Your Java EE application integrates selective components as you manually add a data control for your entities. For more information, see "Using ADF Model Data Binding in a Java EE Web Application" in *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

12.4 How to Work with an EJB Business Services Layer

Create a model business services layer for a web-based EJB 3.0 application.

To create a web-based application:

- Choose **File**, then **New**, then **General Applications**.

  A list of available applications appears. For EJB projects you can choose to build either a custom application or the Java EE Application. The Java EE Application creates an EJB/JPA data-bound web application.

To create JPA entities:

1. In the Application Navigator, select **Model**.
2. Choose **File**, then **New**, then **Business Tier**, then **EJB**, then **Entity** or **Entity from Tables**.
3. When you get to the Persistence Unit page, click **Next** to automatically create a default persistence unit, `persistence.xml`, or click **New** to create a new persistence grouping within the existing `META-INF/persistence.xml` file.
4. Follow the remaining steps in the wizard to create JPA entities.

To implement a session facade:

1. In the Application Navigator select **Model**.
2. Choose **File**, then **New**, then **Business Tier**, then **EJB**, then **Session Bean**.
3. Follow the steps in the wizard.

   When you get to the EJB Name and Options page, be sure to check **Generate Session Facade Methods**.This automatically adds the session facade methods to your session bean. Note that you can create and edit session facade methods for all entities in your project by right-clicking your session bean and choosing **Edit Session Facade**. JDeveloper automatically recognizes new entities in your project and new methods on the entities.

To register the business services model project with the data control:

- Right-click your session bean in the Navigator and choose **Create Data Control**.

This creates a file called `DataControls.dcx` which contains information to initialize the data control to work with your session bean.

To run and test your application:

- You have now created the basic framework for the model layer for a web-based EJB application. Use this framework to test your application as you continue building it. For more information, see Section 12.10, "Running and Testing EJB/JPA Components".

To deploy your application:

The integrated server runs within JDeveloper. You can run and test EJBs using this server and then deploy your EJBs with no changes to them. You do not need to create a deployment profile to use this server, nor do you have to initialize it. Create the deployment descriptor, `ejb-jar.xml` using the Deployment Descriptor wizard, and then package your EJB modules for deployment with your application.

12.5 Using Java EE Design Patterns in Oracle JDeveloper

The Java EE design patterns are a set of best practices for solving recurring design problems. Patterns are ready-made solutions that can be adapted to different problems, and leverage the experience of successful Java EE developers.

JDeveloper can help you implement the following Java EE design patterns in your EJB applications:

- **MVC** - The MVC pattern divides an application into three parts, the Model, View, and Controller. The model represents the business services of the application, the view is the portion of the application that the client accesses, the controller controls the flows and actions of the application and provides seamless interaction between the model and view. The MVC pattern is automatically implemented if you choose the Fusion Web Application (ADF) or Java EE Web Application template when you begin your project.
- **Session Facade** - The session facade pattern contains and centralizes complex interactions between lower-level EJBs (often JPA entities). It provides a single interface for the business services of your application. For more information, see Section 12.7, "Implementing Business Processes in Session Beans".
- **Business Delegate** - The business delegate pattern decouples clients and business services, hiding the underlying implementation details of the business service. The business delegate pattern is implemented by the data control, which is represented in JDeveloper by the Data Control Palette. For more information, see "Using ADF Model Data Binding in a Java EE Web Application" in *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

12.6 Building a Persistence Tier

The persistence tier is the part of your EJB application that contains all of the persistent data object that represent tables in a database. These business components are called JPA entities since the entity model introduced in EJB 3.0 is defined in the Java Persistence API.

12.6.1 About JPA Entities and the Java Persistence API

JPA entities adopt a lightweight persistence model designed to work seamlessly with Oracle TopLink and Hibernate.

The major enhancements with JPA entities are:

- JPA Entities are POJOs
- Metadata Annotations for O-R Mapping
- Inheritance and Polymorphism Support
- Simplified EntityManager API for CRUD Operations
- Query Enhancements

12.6.1.1 JPA Entities are POJOs

JPA entities are now POJOs (Plain Old Java Objects) and there are no component interfaces required for them. JPA entities support inheritance and polymorphism as well.

Example 12–1 contains the source code for a simple JPA entity.

Example 12–1 Source code for a simple JPA entity

```
@Entity
@Table(name = "EMP")
public class Employee implements java.io.Serializable
{
   private int empNo;
   private String eName;
   private double sal;
   @Id
   @Column(name="EMPNO", primaryKey=true)
     public int getEmpNo()
     {
       return empNo;
     }
public void setEmpNo(int empNo)
{
  this.empNo = empNo;
}
 public double getSal()
{
   return sal;
}
...
}
```

Note that the bean class is a concrete class, not an abstract one, as was the case with CMP 2.x entity beans.

12.6.1.2 Metadata Annotations for O-R Mapping

The O-R mapping annotations allow users to describe their entities with O-R mapping metadata. This metadata is then used to define the persistence and retrieval of entities. You no longer have to define the O-R (object Relational) mapping in a vendor-specific descriptor.

The example above uses the `@Entity`, `@Table`, and `@Column` annotations to specify at the class level that this is an entity, and to specify the underlying database table and column names for the entity. You can also use mapping annotations to define a relationship between entities, as shown in Example 12–2.

Example 12–2 Mapping Annotations

```
@ManyToOne(cascade=PERSIST)
@JoinColumn(name="MANAGER_ID", referencedColumnName="EMP_ID")
public Employee getManager()
{
  return manager;
}
```

12.6.1.3 Inheritance and Polymorphism Support

Inheritance is very useful in many scenarios. The two types of inheritance that are commonly used and supported by Oracle Application Server for JPA entities are:

- Single table per class hierarchy
- Joined sub class strategy

The inheritance can be expressed using annotations. Example 12–3 contains code that uses the joined sub class strategy.

Example 12–3 Joined Subclass Strategy

```
@Entity
@Table(name="EJB_PROJECT")
@Inheritance(strategy=JOINED, discriminatorValue="P")
@DiscriminatorColumn(name="PROJ_TYPE")
public class Project implements Serializable
{
...
}
@Entity
@Table(name="EJB_LPROJECT")
@Inheritance(discriminatorValue="L")
public class LargeProject extends Project
{
...
}
@Entity
@Table(name="EJB_PROJECT")
@Inheritance(discriminatorValue="S")
public class SmallProject extends Project
{
...
}
```

12.6.1.4 Simplified EntityManager API for CRUD Operations

The `javax.persistence.EntityManager` API is used for CRUD (Create, Read, Update, and Delete) operations on entity instances. You no longer have to write code for looking up instances and manipulating them. You can inject an instance of EntityManager in a session bean and use `persist()` or `find()` methods on an EntityManager instance to create or query entity bean objects, as show in Example 12–4.

Example 12–4 EntityManager in a Session Bean

```
@PersistenceContext
private EntityManager em;
private Employee emp;
```

```
    public Employee findEmployeeByEmpNo(int empNo)
    {
       return ((Employee) em.find("Employee",empNo));
     }
public void addEmployee(int empNo, String eName, double sal)
{
   if (emp == null) emp = new Employee();
   emp.setEmpNo(empNo);
   ...
   em.persist(emp);
}
}
```

12.6.1.5 Query Enhancements

Queries are defined in metadata. You may now specify your queries using annotations, or in a deployment descriptor. JPA entities support bulk updates and delete operations through JPQL (Java Persistence Query Language). For more information, see Section 12.6.7, "JDK 5 Annotations for EJB/JPA".

12.6.2 How to Create JPA Entities

JDeveloper offers you two easy wizards to create your JPA entities. You can create entities from online or offline databases, add a persistence unit, define inheritance strategies, and select from available database fields. The Entity from Tables wizard allows you to create entities from online or emulated offline databases, as well as from a application server data sources.

To create entities or entities from tables:

1. Choose **File** menu, then **New**, then **Business Tier**, then **EJB**, then **Entity** or **Entities from Tables**.
2. Follow the steps in the wizard.

To create EJBs in an existing module:

1. In the Navigator, right-click an EJB module and choose **New**, then **EJB**, then **Entity** or **Entities from Tables**.

   Or, choose **File** menu, then **New**, then **Business Tier**, then **EJB**, then **Entity** or **Entities from Tables**.
2. Follow the steps in the wizard.

To create EJBs in a new EJB module:

1. Choose **File** menu, then **New**, then **General**, then **Projects**.
2. Select the type of project you want to create and click **OK**.
3. In the Navigator, right-click on the new project and choose **New**.

   Or, select the module and choose **File** menu, then **New**, then **Business Tier**, then **EJB**, then **Entity** or **Entities from Tables**.

12.6.3 About SDO For EJB/JPA

JDeveloper provides support for the SDO (Service Data Objects) data application development framework.

Use the SDO 2.0 framework and API to easily modify business data regardless of how it is physically accessed. SDO encapsulates the backend data source, offers a choice of static or dynamic programming styles, and supports both connected and disconnected access. SDO handles XML parser operations, and automatically integrates the data parsing logic with the application. For more information, "Integrating Service-Enabled Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The SDO architecture supported by JDeveloper offers the following:

- Simplifies the J2EE data programming model
- Abstracts data in a service oriented architecture (SOA)
- Unifies data application development by creating a standard way of passing data between clients
- Supports and integrates XML
- Incorporates J2EE patterns and best practices

SDO is a unified framework for data application based on the concept of disconnected data graphs. A data graph is a collection of tree-structured or graph-structured data objects. To enable development of generic or framework code that works with Data Objects, it is important to be able to introspect on Data Object metadata, which exposes the data model for the Data Objects. As an alternative to Java reflection, SDO provides APIs to access metadata stored in XML schema definition (XSD) files that you create, based on the entity or data model information detailed in your EJB session beans.

12.6.4 Using an EJB/POJO-based ADF-BC Service for Deployment to the SOA Platform

The SDO feature in JDeveloper can be used as an EJB service or as an ADF-BC service. If you choose to use an ADF-BC service you need add the listener reference to your `weblogic-application.xml` file. For more information, see Section 12.6.5, "How to Create an SDO Service Interface for JPA Entities".

For more information and specifications on SDO, see the OSOA (Open Service Oriented Architecture) at
`http://osoa.org/display/Main/Service+Data+Objects+Home`

12.6.5 How to Create an SDO Service Interface for JPA Entities

You can easily create a service interface API to access JPA entity data through either an EJB session bean or a plain old Java object (POJO). This service class exposes operations for creating, retrieving, updating, and deleting the JPA entities in your JDeveloper J2EE application.

To create a SDO service interface:

1. Start with an EJB session bean, or an ordinary Java class (POJO), that exposes CRUD methods for one or more JPA entities.

   You can use the wizard to create your session beans. For more information, see Section 12.7.2, "How to Create a Session Bean".

2. In the Structure window, right-click your EJB session Bean or POJO and choose **Create Service Interface**.

3. Select the methods you want to make available in your service API.

   By default all of the methods in your session bean interface are selected. Click the checkbox to select or unselect a method.

4. In this release, when you create a service interface, your original session bean file and the remote (or local) interface are modified. New methods are added that match the original ones, but they reference newly defined SDO data objects instead of JPA entities. These SDO data objects match the JPA entities and are defined in XSD files, which are also added to your project, and their names are appended with SDO, such as `DeptSDO` or `EmployeeSDO`. Select **Backup File(s)** to create a backup of your original session bean file.
5. Click **OK**.

12.6.5.1 How to Configure an EJB/POJO-based ADF-BC Service for Deployment to the SOA Platform

To use an EJB/POJO SDO ADF-BC service from a fabric composite using SDO external bindings, you need to set up the Weblogic application deployment listener to invoke the `ServiceRegistry logic`. Set this up by adding the listener reference to your `weblogic-application.xml` file.

To add the listener reference:

Add the code in Example 12–5 to the `weblogic-application.xml` which by default is located in `<workspace-directory>/src/META-INF`.

Example 12–5 Code Added to weblogic-application.xml

```
<listener>
<listener-class> oracle.jbo.client.svc.ADFApplicationLifecycleListener
</listener-class>
</listener>
```

Once this listener is added, JDeveloper automatically registers the SDO service application name `_JBOServiceRegistry_` into the fabric service registry in the `composite.xml`.

12.6.5.2 File Types Created to Support Your SDO Architecture

When you create your SDO service interface, the necessary files to support your service interface are automatically created. These files include the following:

- **SessionEJBBeanWS.wsdl** - This file describes the capabilities of the service that provides an entry point into an SOA application or a reference point from an SOA application. The WSDL file provides a standard contract language and is central for understanding the capabilities of a service.
- **SessionEJBBeanWS.xsd** - This is an XML schema file that defines your service interface methods in terms of SDO data types. All of the entities that were contained in your session bean interface will have a corresponding DataObject element in this schema file. At runtime, these DataObjects are registered with the SDO runtime by calling `XSDHelper.INSTANCE.define()` method. A static type-specific DataObject is defined for each SDO type.

12.6.6 How to Generate Database Tables from JPA Entities

When you deploy the JDeveloper integrated server, database tables are automatically created for every entity that does not have a corresponding existing mapped table. One database table will be generated per unmapped JPA entity.

Note: Primary key referential integrity constraints will be generated, but other constraints may not be.

To generate database tables from JPA entities:

1. Create your JPA entity using the modeling tools or the Create Entity wizards. For more information, see Section 12.6.2, "How to Create JPA Entities.".
2. Modify the entities as necessary, adding fields and constraints.
3. Name the tables:
   - EJB 3.0 - Annotate the bean class to provide a table name. For more information, see the Enterprise JavaBean specification at `http://www.oracle.com/technetwork/java/docs-135218.html`.
4. Deploy the persistence unit. For more information, see Section 12.9, "Deploying EJB Modules and JPA Persistence Units.".

12.6.7 JDK 5 Annotations for EJB/JPA

Annotations can simplify your development tasks by reducing the number of deployment descriptors needed for your application components. Annotations can also be used to generate artifacts such as interfaces.

An annotation is a metadata modifier that is added to a Java source file. Annotations are compiled into the classes by the Java compiler at compile time, and can be specified on classes, fields, methods, parameters, local variables, constructors, enumerations, and packages. Annotations can be used to specify attributes for generating code, for documenting code, or for providing services like enhanced business-level security or special business logic during runtime.

Every type of annotation available for your EJB/JPA classes can also, alternatively, be added to an XML deployment descriptor file. At runtime the XML will override any annotations added at the class level.

Annotations are marked with the `@` symbol, such as this stateless session bean annotation:

```
@Stateless public class MySessionBean
```

For more information on annotations for EJB 3.0, see `http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html`

Note: Annotations are new to EJB 3.0, and not available for previous versions of EJB.

During design time, JDeveloper displays a list of available annotations through the Property Inspector. You can change any suitable Java class to an EJB or JPA component using the annotation feature. For more information, see Section 12.6.8, "How to Annotate Java Classes.".

12.6.7.1 EJB 3.0

Annotations are available to indicate the bean type. Adding your bean type annotation to a regular class turns it into an EJB.

The following types of annotations are available:

- Is Stateless Session Bean. Choose **TRUE** or **FALSE** to annotate your class as a stateless session bean.
- Is Stateful Session Bean. Choose **TRUE** or **FALSE** to annotate your class as a stateful session bean.
- Is Message Driven Bean. Choose **TRUE** or **FALSE** to annotate your class as a message driven bean.

12.6.7.2 JPA 1.0

Annotations support a new Java Persistence API as an alternative to entity beans.

The following types of annotations are available:

- Is JPA Entity. Choose **TRUE** or **FALSE** to annotate your class as a JPA entity.
- Is JPA Mapped Superclass. Choose **TRUE** or **FALSE** to annotate your class as a JPA mapped superclass.
- Is JPA Embeddable. Choose **TRUE** or **FALSE** to annotate your class as JPA embeddable.

Once you transform your regular Java class into an EJB/JPA component, or if you used one of the EJB/JPA wizards to create the component, the Property Inspector displays a different set of contextual options, which you can use to add or edit annotations for the various members within the component class.

12.6.8 How to Annotate Java Classes

During design time, JDeveloper provides you with the list of available annotations to insert into your classes. The options change depending on what type of class you are working on, and what member you have selected.

You can annotate any regular Java class to turn it into an EJB/JPA component. Once the class is defined with annotations as an EJB/JPA, you can easily customize the component with a variety of member-level annotations available to choose from in the JDeveloper Property Inspector.

> **Note:** Annotations are only available for EJB 3.0, and not available for previous versions of EJB.

To annotate your Java class as an EJB/JPA component:

1. In the Application Navigator, select the class you want to transform.
2. In the Structure window, double-click the class name.

   If your class is already open in the Java source editor, put your curser in the class definition line.
3. Open the Property Inspector, select the EJB/JPA tab and choose the type of component you want to create. Select **True**.

After your Java class is annotated as an EJB/JPA component, the EJB/JPA tab disappears from the Property Inspector and a new tab appears, specific to the component type you chose. To change the component back to a regular Java class, remove the annotation from the code to reset the EJB/JPA component types displayed in the Property Inspector.

Note: EJB or JPA components created through the wizards already contain the class type annotations. For more information, see Section 12.3, "Building EJB 3.0 Applications and Development Process.".

Once your Java class is transformed into an EJB/JPA component using a class-level annotation, use the Property Inspector to add or edit annotations to member fields or methods within that component.

To add or edit annotations in an EJB/JPA component:

1. In the Application Navigator, select the class you want to annotate.
2. In the Structure window, double-click the member you want to annotate.

   As an alternative, if your class is already open in the Java source editor, put your curser in the location where you intend to insert your annotation.
3. In the Property Inspector, choose the tab corresponding to your EJB/JPA type.
4. Choose from any of the annotations available for the specific member you have selected.

12.6.9 Representing Relationships Between Entities

When you create entities from database tables, foreign keys are interpreted as relationships between entities. You can further define these relationships, create new relationships, or map existing relationships to existing tables using the JDeveloper modeling tools. With the modeling tools you can represent relationships as lines between entities, and change the relationships by changing the line configurations. For more information, see Section 23.3, "Modeling EJB/JPA Components on a Diagram.".

12.6.10 Java Persistence Query Language

Java Persistence Query Language (JPQL) offers a standard way to define relationships between entity beans and dependent classes by introducing abstract schema types and relationships in the deployment descriptor. JPQL also defines queries for navigation using abstract schema names and relationships.

The JPAQL query string consists of two mandatory clauses: SELECT and FROM, and an optional WHERE clause. For example:

```
select d from Departments d where d.department_name = ?1
```

There are two kinds of methods that use JPQL, *finder* methods and *select* methods.

- Finder methods are exposed to the client and return either a single instance, or a collection of entity bean instances.
- Select methods are not exposed to the client, they are used internally to return an instance of `cmp-field` type, or the remote interfaces represented by the `cmr-field`.

12.6.11 JPA Object-Relational Mappings

The Java Persistence API lets you declaratively map Java objects to relational database tables in a standard, portable way that works both inside a Java EE 5 application server and outside an EJB container. This approach greatly simplifies Java persistence and provides an object-relational mapping approach.

With Oracle TopLink you can configure the JPA behavior of your entities using metadata annotations in your Java source code. At run-time the code is compiled into the corresponding Java class files.

To designate a Java class as a JPA entity, use the `@Entity` annotation, as shown in Example 12–6.

Example 12–6 Entity Annotation

```
@Entity
public class Employee implements Serializable {
...
}
```

You can selectively add annotations to override defaults specified in your deployment descriptors.

For more information on JPA Annotations, see the *TopLink JPA Annotation Reference* at `http://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html`.

12.6.12 How to Use Java Service Facades

A Java service facade implements a lightweight testing environment you can run without an application server.

With EJB 3.0 the Java service facade is similar to an EJB session facade, because you can generate facade methods for entities in the same persistence unit, without the container.

Separating workflow with Java service facades eliminates the direct dependency of the client on the participant JPA objects and promotes design flexibility. Although changes to participants may require changes in the Java service facade, centralizing the workflow in the facade makes such changes more manageable. You change only the Java service facade rather than having to change all the clients. Client code is also simpler because it now delegates the workflow responsibility to the session facade. The client no longer manages the complex workflow interactions between business objects, nor is the client aware of interdependencies between business objects.

You may choose to make the Java service class runnable by generating a sample Java client with a main() method.

Use the JDeveloper Java service facade wizard to create a Java class as a service facade to entities. To create a new Java service facade select the **File** menu, then **New**, then **Business Tier**, then **EJB**, then **Java Service Facade**.

You can also create a data control from a service facade. In the Application Navigator, right-click the name of the service facade, then select **Create Data Control**. From the **Bean Data Control Interface Chooser** dialog, you can choose to implement `oracle.binding.*` data control interfaces. The interfaces are `TransactionalDataControl`, `UpdatableDataControl`, and `ManagedDataControl`. For more information, select the **Help** button in the dialog.

12.7 Implementing Business Processes in Session Beans

A session bean represents a single client inside the application server. To access an application deployed on the server, the client invokes the session bean methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server. A session bean is similar to an interactive

session. A session bean is not shared and has only one client, in the same way that an interactive session can have only one user. Like an interactive session, a session bean is not persistent as it does not save data to the database. When the client terminates, its session bean appears to terminate and is no longer associated with the client.

Create your session beans and session bean facades using the JDeveloper Session Bean Wizard. For more information, see Section 12.7.2, "How to Create a Session Bean.".

There are two types of session beans:

- **Stateful**. A stateful session bean maintains conversational state on behalf of the client. A conversational state is defined as the session bean field values plus all objects reachable from the session bean fields. Stateful session beans do not directly represent data in a persistent data store, but they access and update data on behalf of the client. The lifetime of a stateful session bean is typically that of its client.
- **Stateless**. Stateless session beans are designed strictly to provide server-side behavior. They are anonymous because they contain no user-specific data. The EJB architecture provides ways for a single stateless session bean to serve the needs of many clients. All stateless session bean instances are equivalent when they are not involved in serving a client-invoked method. The term stateless means that it does not have any state information for a specific client. However, stateless session beans can have non-client specific state, for example, an open database connection.

12.7.1 Using Session Facades

With JDeveloper you can select to automatically generate your session facade methods any time you create a session bean through the session bean wizard. This creates a session bean that functions as a session facade for your business workflow. For more information, see Section 12.7.2, "How to Create a Session Bean.".

The session facade is implemented as a session bean. The session bean facade encapsulates the complexity of interactions between the business objects participating in a workflow by providing a single interface for the business services of your application.The session facade manages the relationships between numerous BusinessObjects and provides a higher level abstraction to the client.

Session facades can be either stateful or stateless, which you define while creating a session facade in the wizard.

For more information on session facades, see the Oracle Technology Network at `http://www.oracle.com/technetwork/java/sessionfacade-141285.html`

Use the wizard to automatically implement a session facade when you create a session bean, and to choose the methods you want to implement. Once you've created EJB entities, any session beans you create in the same project are aware of the entities and the methods they expose.

12.7.2 How to Create a Session Bean

Use the session bean wizard to create a new session bean or session facade bean. Or you can create a session bean using the modeling tools.

To create a session bean or session facade using a wizard:

1. In the Navigator, select **File**, then **New**.
2. In the New Gallery, select **Session Bean** from the **Business Tier** category under the **EJB** folder.

3. To make the bean a session facade select **Generate Session Facade Methods** on the EJB Name and Options page.

> **Note:** You must have already created a persistence unit before you can generate a session facade bean. To generate a persistence unit follow the same steps, but select **JPA Persistence Unit** instead of **Session Bean**.

4. Complete the remaining steps in the wizard.

To add or remove session bean facade methods:

1. In the Application Navigator, select the session bean you want to edit.
2. Right-click and choose **Edit Session Facade**.
3. In the Specify Session Facade Options dialog, check a method expose it through the facade, or unchecked a method so it will not be exposed.

For more information on session facades, see the *Core J2EE Pattern Catalog* at `http://www.oracle.com/technetwork/java/sessionfacade-141285.html` .

You can also create a session facade manually by creating a local reference between a session bean and an entity.

To create a local reference:

1. Create a session bean, if you have not already done so.
2. Create a local reference between the beans:
   - **In the bean class** - If you are using EJB 3.0, annotate the bean class to create a reference.
   - **Using the EJB Module Editor** - If you are using EJB 2.1 (and previous), select an EJB node in the Application Navigator, then double-click **Methods** in the Structure pane to open the EJB Module Editor. Select **EJB Local References**.

To create a session bean on an EJB diagram:

1. Open your EJB diagram.

   If you do not have an EJB diagram, select **File**, then **New**, then select **EJB Diagram** from the **Business Tier** category.
2. In the Component Palette, click **Session Bean**.

   If the Component Palette is not visible, from the **View** menu, choose **Component Palette**.
3. Click inside the EJB diagram (note that you do not drag and drop).

12.7.3 How to Create Session or Message-Driven Beans in Modules

You can create EJBs in both new and existing modules.

To create EJBs in an existing module:

1. In the Navigator, right-click an EJB module and choose **New EJB Session** or M**essage-Driven Bean**.

Or, select the module and choose **File** menu, then **New**, then **Business Tier**, then **EJB Session** or **Message-Driven Bean**.

2. Follow the steps in the wizard.

To create EJBs in a new EJB module:

1. In the Navigator, select **File**, then **New**, then **General**, then **Projects**.
2. In the New Gallery, select the type of project you want to create and click **OK**.
3. In the Navigator, right-click on the new project and choose **New**.
4. In the New dialog, expand the category for **Business Tier**.
5. Click **EJB**, then **Session Bean** or **Message-Driven Bean**.

   Or, select the module and choose **File** menu, then **New**, then **Business Tier**, then **EJB Session** or **Message-Driven Bean**.
6. Click **OK**.
7. Follow the steps in the wizard.

12.7.4 How to Add, Delete, and Edit EJB Methods

Once an EJB has been added to your project, you can add, delete, or edit the methods in it. Adding methods as described below ensures that changes are synchronized with remote and home interfaces, when defined.

To add methods (EJB 2.1):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, right-click the **Methods** node, then choose **New EJB Method**, then choose the type of method you want to create.
3. In the Method Details dialog, add details, as necessary.
4. When finished, click **OK**.

To add methods (EJB 3.0):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, right-click the EJB, then choose **Enterprise Java Beans (EJB)**, then choose **New Method**.
3. In the Bean Method Details dialog, add details, as necessary.
4. When finished, click **OK**.

To delete methods (EJB 2.1):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, expand the **Methods** node.
3. Right-click the method you want to remove. If you want to remove more than one, select them using Ctrl-click or Shift-click, then right click the selection.
4. Choose **Remove Methods...**.
5. Click **Yes** to confirm.

To delete methods (EJB 3.0):

1. In the Application Navigator, select an EJB.

2. In the Structure pane, double-click the method to locate it in the source file.
3. In the source file, delete the method.

To edit methods (EJB 2.1):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, expand the **Methods** node.
3. Right-click the method you want to edit, then choose **Properties**.
4. In the Method Details dialog, edit details, as necessary.
5. When finished, click **OK**.

To edit methods (EJB 3.0):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, right-click the EJB, then choose **Enterprise Java Beans (EJB)**, then choose **Properties**.
3. In the Bean Method Details dialog, edit details, as necessary.
4. When finished, click **OK**.

12.7.5 How to Add a Field to an EJB

You can add fields to EJBs on an EJB diagram or through the EJB Module Editor.

To add a field on an EJB Diagram (EJB 2.1):

1. Click in the fields compartment (the first compartment) on the EJB in the diagram.
2. Enter the name of the field and its type.

To add a field using the EJB Module Editor (EJB 2.1):

1. In the Application Navigator, right-click the EJB to which you want to add a field, then choose **Properties**.
2. In the EJB Module Editor, click to expand the EJB, then select **Fields**.
3. Click **Add**.
4. In the Field Details dialog, add details, as needed.

To add a field (EJB 3.0):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, right-click the EJB, then choose **Enterprise Java Beans (EJB)** node, then choose **New Field**.
3. In the Field Details dialog, add details, as necessary.
4. When finished, click **OK**.

12.7.6 How to Remove a Field From an EJB

You can remove fields from EJBs, as described below.

To remove a field on an EJB Diagram:

1. Click in the fields compartment (the first compartment) on an EJB.

2. Highlight the field and press the Delete key.

To remove a field using the EJB Module Editor (EJB 2.1):

1. In the Application Navigator, right-click the node for the EJB to which you want to add a field, then choose **Properties**.
2. In the EJB Module Editor, click to expand the EJB, then select **Fields**.
3. Select the field you want to remove.
4. Click **Delete**.

To remove a field (EJB 3.0):

1. In the Application Navigator, select an EJB.
2. In the Structure pane, double-click the field to locate it in the source file.
3. In the source file, delete the field.

12.7.7 Customizing Business Logic with EJB Environment Entries

Environment entries are name-value pairs that allow you to customize the bean's business logic. Since environment entries are stored in an enterprise bean's deployment descriptor, a bean's business logic can be changed without changing its source code.

For example, an EJB that calculates an order might give a discount depending on the number of items ordered, a certain status (silver, gold, platinum), or for a promotion. Before deploying the bean's application you could assign the discount a certain percentage. When the application runs, a method would call the environment entry to find out the discount value. If you wanted to change that percentage in a different deployment, you would not need to change the source code, you would just need to change the value in the environment entries for the deployment descriptor.

Environment entries are annotated in the source code.

For the complete EJB 3.0 Java Community Process specifications and documentation, see `http://www.oracle.com/technetwork/java/docs-135218.html`.

12.7.8 Exposing Data to Clients

Depending on how your develop your application, there are different methods of exposing data to clients.

- If you're using the Oracle ADF framework, the preferred method of exposing data to clients is to implement the session facade design pattern and drop the session bean onto the data control palette. This option vastly simplifies data coordination and is only available in the JDeveloper Studio release. For more information, see Section 12.7, "Implementing Business Processes in Session Beans" and Section 12.7.1, "Using Session Facades."
- If you are not using the Oracle ADF framework, you typically create a managed bean to coordinate connection to a JSF/JSP page. For more information, see Section 11.2, "Developing Applications with JavaServer Faces."

12.7.9 How to Identify Resource References

A resource reference is an element in a deployment descriptor that identifies the component's coded name for the resource. Resource references are used to obtain

connector and database connections, and to access JMS connection factories, JavaMail sessions, and URL links.

To add or modify EJB 3.0 resource references:

Go to your source code to annotate resource references.

12.7.10 How to Define a Primary Key for an Entity

A primary key is a unique identifier with one or more persistent attributes. It identifies one instance of a class from all other instances of the same type. Use primary keys to define relationships and to define queries.

Each JPA entity instance must have a primary key. To accommodate your database schema, you can define simple primary keys from persistent fields or composite primary keys from multiple persistent fields. You can also define automatic primary key value generation to simplify your JPA entity implementation.

The simplest way to specify a simple primary key is to use annotations for a single primitive, or JDK object type entity field as the primary key. You can also specify a simple primary key at deployment time using deployment XML.

To configure a simple primary key using annotations:

1. In your JPA entity implementation, annotate the primary key field using the `@Id` annotation, as shown in Example 12–7.

Example 12–7 Configuring Primary Key Using Annotations

```
import javax.ejb.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Column;

@Entity
@Table(name = "EMP")
public class Employee implements java.io.Serializable {
private int empNo;
private String eName;
private String birthday;
private Address address;
private int version;

public Employee() {
{

@Id
@Column(name="EMPNO")
public int getEmpNo() {
return empNo;
}
...
}
```

2. Package and deploy your application.

To configure a simple primary key using deployment XML:

1. In your JPA entity implementation, implement a primary key field, as shown in Example 12–8.

Example 12–8 Configuring Primary Key Using Deployment XML

```
public class Employee implements java.io.Serializable {
private int empNo;
private String eName;
private String birthday;
private Address address;
private int version;

public Employee() {
{

public int getEmpNo() {
return empNo;
}
...
{
```

12.7.11 How to Specify a Primary Key for ADF Binding

For certain ADF Faces features, a designated primary key is required. For example, if you have an ADF Faces table that uses an `af:tableSelectMany` component, you will need to specify a primary key to be able to implement sorting. When you create EJB/JPA entities from tables (using EJB 3.0), the primary key is specified by default. But if you have to specify a primary key, do the following:

To specify an attribute as primary key:

1. If you have not already done so, you will need to create an ADF Data Control to create the XML definitions for each entity. For more information, see "Using ADF Model Data Binding in a Java EE Web Application" in *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.
2. In the Application Navigator, select an EJB entity XML file.
3. In the Structure pane, select an entity attribute and then from the **View** menu, choose **Property Inspector**.
4. In the Property Inspector, find the attribute you want as the primary key and set the **PrimaryKey** value to **true**.

12.7.12 How to Use ADF Data Controls for EJBs

JDeveloper automatically provides a complete set of data control components when you build an ADF Fusion web application. When you build a Java EE application, and/or an EJB project, you assign ADF data controls on your individual session beans. This adds a data control file with the same name as the bean.

For complete details on using ADF data controls for EJBs, see "Using ADF Model Data Binding in a Java EE Web Application" in *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

12.8 Modeling EJB/JPA Components on a Diagram

For information about modeling EJB and JPA components on a diagram, see Section 23.3, "Modeling EJB/JPA Components on a Diagram.".

12.9 Deploying EJB Modules and JPA Persistence Units

An EJB module is a software unit comprising one or more EJBs, a persistence unit, and an optional EJB deployment descriptor. A JDeveloper project contains only one EJB module. At deploy-time, the module is packaged as an `ejb.jar` file.

Entity beans were once packaged in the EJB JAR file along with the session and message-driven beans. Today, with JPA entities and the persistence unit technology, at deploy-time, they are packaged in their own JAR file, `persistenceunit.jar`.

Now your entity beans (JPA entities) are contained separately, in a JPA persistence archive JAR, which includes a `persistence.xml` file. The JPA persistence unit does not have to be part of the EJB module package, but can be bundled inside the `ejb.jar` file.

12.9.1 About EJB Modules

JDeveloper project can contain only one EJB module. When you create your first session or message-driven bean in a project, a module is automatically established, if one does not already exist. You are given the option of choosing the EJB version and the persistence manager for your new EJB module.

When you deploy your project you convert the aggregate of session and message-driven beans, plus deployment descriptor into an a EJB JAR file (.jar file), ready for deployment to an application server or as an archive file. By confining the persistence unit to its own JAR file, the persistence unit can easily be reused in other applications. For more information, see Section 9.1, "About Deploying Applications."

12.9.2 About JPA Persistence Units

A JPA persistence unit is comprised of a `persistence.xml` file, one or more optional `orm.xml` files, and the managed entity classes that belong to the persistence unit. A persistence unit is a logical grouping of the entity manager, data source, persistent managed classes, and mapping metadata. A persistence unit defines an entity manager's configuration by logically grouping details like entity manager provider, configuration properties, and persistent managed classes.

Each persistence unit must have a name. Only one persistence unit of a given name may exist in a given EJB-JAR, WAR, EAR, or application client JAR. You can package a persistence unit in its own persistence archive and include that archive in whatever Java EE modules require access to it.

The `persistence.xml` file contains sections or groupings, these groupings correspond to your entities, and run-time data related to the entities. When you create a new entity using the entity wizards, and if you have an existing persistence unit in the project, the entity will be inserted into its own section in the persistence.xml. If you do not have an existing persistence unit, one will be created automatically, with a section included for the entity definitions.

The JAR file or directory, whose META-INF directory contains the persistence.xml file, is called the root of the persistence unit. An EJB 3.0 application that uses entities must define at least one persistence unit root either explicitly or using the OC4J default persistence unit. When you deploy your persistence unit, a JAR file is created called `persistenceunit.jar`. For more information, see Section 9.1, "About Deploying Applications."

12.9.3 How to Create a JPA Persistence Unit

You can easily create a persistence unit for your entities using the JDeveloper Persistence Unit wizard. Or, when you create a JPA entity, a default persistence unit is created for you, if you do not already have one.

To create a JPA persistence unit:

1. Select a project in the Application Navigator.
2. Choose **File** menu, then **New**, then **Business Tier**, then **EJB**, then **JPA Persistence Unit**.
3. Complete the steps in the wizard.

12.9.4 How to Remove EJBs in a Module

To remove an EJB from an EJB module, select the EJB in the System Navigator and press **Delete**.

12.9.5 How to Import EJBs into JDeveloper

You can import existing EJBs from a JAR file or from a deployment descriptor.

To import an EJB module, or a subset of EJBs within an EJB module into a project:

1. From the **File** menu, choose **Import**.
2. In the Import dialog, choose **EJB JAR (.jar) File**.
3. Follow the steps in the wizard.

To import an EJB deployment descriptor (ejb-jar.xml) file:

1. From the **File** menu, choose **Import**.
2. In the Import dialog, choose **EJB Deployment Descriptor (ejb-jar.xml) File**.
3. Follow the steps in the wizard

> **Note:** If you import a deployment descriptor using this wizard, and then use the wizard to import more files, the wizard caches the last used descriptor file, JAR file, and descriptor source directory in the IDE preferences file for convenience. This makes it easier to do tasks such as splitting an EJB module into multiple modules, importing multiple JAR files residing in the same directory, etc.

To import a WebLogic deployment descriptor (weblogic-ejb-jar.xml) file:

1. From the **File** menu, choose **Import**.
2. In the Import dialog, choose **EJB Deployment Descriptor (ejb-jar.xml) File**.
3. Follow the steps in the wizard
4. After completing the wizard, in the Navigator, right-click on **weblogic-ejb-jar.xml** and choose **Export to OC4J**.

To avoid conflicts, if an EJB with the same name already exists in your existing module, that EJB will not be imported.

12.9.6 How to Modify EJB/ADF Applications to Deploy to Websphere Application Server

NOTE: Could not access this topic in Help Center.

12.10 Running and Testing EJB/JPA Components

To test your EJBs you need to run a client program that can create or find EJB instances and call their remote interface methods. JDeveloper provides a sample client utility that will help you create clients quickly. You can run and test EJBs using either the integrated server or a remote server; the sample client utility can be used to create a client for either type.

12.10.1 How to Test EJB/JPA Components Using the Integrated Server

The integrated Oracle WebLogic Server runs within JDeveloper. You can run and test EJBs quickly and easily using this server, and then deploy your EJBs with no changes to them. You do not need to create a deployment profile to use this server, nor do you have to initialize it.

To run a sample client on the integrated Oracle WebLogic Server:

1. In the Application Navigator, right-click on an EJB and choose **Run**.

   Notice in the Message pane that Oracle WebLogic Server has been launched.

2. Right-click on an EJB and choose **Create Sample Java Client** from the context menu.

3. The default choice is to create a client for the integrated Oracle WebLogic Server, so click **OK**.

   The client is created and opens in the code editor.

   If your session bean serves as a facade over JPA entities, code is generated to instantiate the query methods. If you exposed methods on your bean, the generated client contains methods that can be uncommented to call them.

4. After your EJB has been successfully started from the Application Navigator, right-click on the sample client and choose **Run**.

12.10.2 How to Test EJB/JPA Components Using a Remote Server

To test EJBs on a remote server you need to deploy the EJB and then create a sample client. If you deploy first, the framework picks up the deployed applications, which populates the client pick list.

Note: You cannot mix different version EJBs in the same module.

To run a sample client on a remote server:

1. Launch your application server.

2. In the Application Navigator, right-click your project node and choose **New**.

3. In the New dialog box, click the **Deployment Profiles** category and choose **Business Components EJB Session Bean**.

   The new deployment profile is displayed in the Application Navigator.

4. Right-click the deployment profile and choose **Deploy to New Connection**.
5. In the dialog box, specify the application server you want to use. (The Oracle WebLogic Server that ships with JDeveloper is selected by default.)
6. Click **OK**.
7. In the Application Navigator, right-click on the deployment profile and choose **Deploy to** <***named connection***>.
8. In the Application Navigator, right-click on an EJB and choose **Create Sample Java Client**.
9. In the dialog box, choose to connect to a **Remote Ape Server**. Choose one of the deployed Java EE applications listed in the combo box.
10. Click **OK**.

    The client is created and displays in the Application Navigator.
11. Right-click the client and choose **Run**.

    The Message pane shows you the running output.

12.10.3 How to Test EJB Unit with JUnit

JDeveloper provides support for JUnit regression testing for your EJBs. JUnit is an open source Java regression testing framework that comes as an optional feature in JDeveloper. To use this feature you'll need to install the JUnit extension.

Use JUnit to write and run tests that verify your code. After you install the JUnit extension, you can use the simple wizard to select your session bean or Java class files, to select the methods that you want to test within those files, and then to start the JUnit test.

To run a JUnit test on an EJB:

1. Install the Junit extension from the JDeveloper Help menu. For more information, see Section 18.12.1, "How to Install JUnit."
2. Select your EJB session bean or an ordinary Java class (POJO) in the Application Navigator. Or you can navigate to it from within the wizard.
3. Click **File** menu, then **New**, then **Business Tier**, then **EJB**, then **EJB JUnit TestCase**.
4. Start the JUnit wizard.
5. Complete the steps in the wizard.

For detailed information about JUnit, visit the JUnit website, `http://www.junit.org/`.

13 Developing TopLink Mappings

This chapter describes how to develop TopLink Mappings within Oracle JDeveloper. Using the TopLink Editor, you can quickly and easily configure TopLink descriptors and mappings for your Java classes, EJBs, and JPA entities to data source elements (such as database tables or XML schema elements). With the TopLink Editor, you can create this information without writing Java code.

This chapter includes the following sections which describe the general process for creating TopLink mappings and integrating them in a JDeveloper project:

- Section 13.1, "About Developing TopLink Mappings"
- Section 13.2, "Developing TopLink JPA Projects"
- Section 13.3, "Developing TopLink Relational Projects"
- Section 13.4, "Developing TopLink XML Projects"
- Section 13.5, "Developing TopLink EIS Projects"
- Section 13.6, "Developing TopLink Sessions"
- Section 13.7, "Developing TopLink Applications"

For more information, see the following:

- Chapter 10, "Getting Started with Developing Java EE Applications"
- Chapter 11, "Developing Applications Using Web Page Tools"
- Chapter 12, "Developing with EJB and JPA Components"

13.1 About Developing TopLink Mappings

Oracle TopLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality.

Using the TopLink Editor available within JDeveloper you can configure and map your Java classes, EJBs, and JPA entities to different data sources, including relational databases, enterprise information systems (EIS), and XML schemas without using Java code. The TopLink Editor supports multiple standards, including JPA, JAXB, and Java EE.

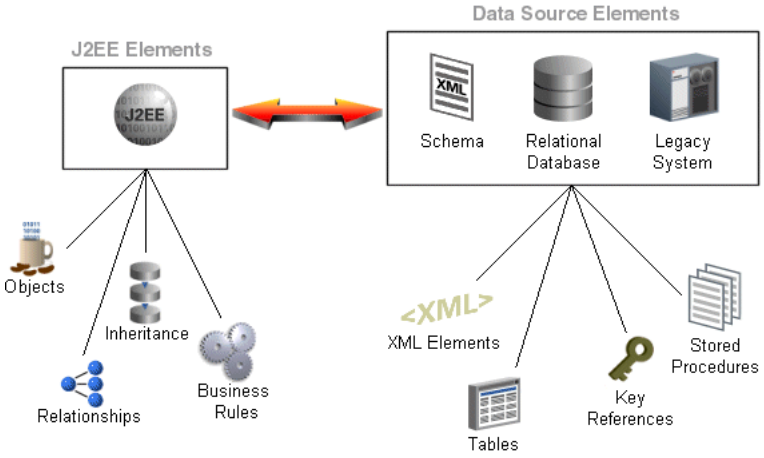
TopLink links object-oriented programs with relational data structures. Using TopLink, you can build high-performance applications that store persistent object-oriented data in a relational database. TopLink successfully transforms object-oriented data into either relational data or XML documents. Using TopLink, you can integrate persistence

and object-transformation into your application, while staying focused on your primary domain problem by taking advantage of an efficient, flexible, and field-proven solution.

13.1.1 Considering the Impedance Mismatch

TopLink enables you to address the disparity between Java and data sources, known as impedance mismatch. While object-relational databases consist of such elements as tables, rows, columns, and primary and foreign keys, Java and Java EE include entity classes (regular Java classes or Enterprise JavaBeans (EJB) entity beans), business rules, complex relationships, and inheritance. In a nonrelational data source, you must match your Java entities with EIS records or XML elements and schemas. These differences (as shown in Figure 13–1) are known as the object-persistence impedance mismatch.

Figure 13–1 Solving Object-Persistence Impedance Mismatch



13.1.2 Designing TopLink Applications

You can use TopLink to perform a variety of persistence and data transformation functions on any enterprise architecture that uses Java, including:

- Java EE
- Spring
- Java web servers such as Tomcat
- Java clients such as Java SE and web browsers

13.1.3 Using TopLink in Application Design

TopLink can be used in the following ways:

- **Relational Database Usage**: You can use TopLink to persist Java objects to relational databases that support SQL data types accessed using JDBC.
- **Oracle XML Database (XDB) Usage**: You can use TopLink to persist XML documents to an Oracle XML database using TopLink direct-to-XMLType mappings.
- **Enterprise Information System (EIS) Usage**: You can use TopLink to persist Java objects to an EIS data source using a JCA adapter. In this scenario, the application invokes EIS data source-defined operations by sending EIS interactions to the JCA

adapter. Operations can take (and return) EIS records. Using TopLink EIS descriptors and mappings, you can easily map Java objects to the EIS record types supported by your JCA adapter and EIS data source. This usage is common in applications that connect to legacy data sources and is also applicable to web services.

- **XML Usage**: You can use TopLink for in-memory, nonpersistent Java object-to-XML transformation with XML Schema (XSD) based XML documents and JAXB. You can use the TopLink JAXB compiler with your XSD to generate both JAXB-specific artifacts (such as content and element interfaces, implementation classes, and object factory class) and TopLink-specific artifacts (such as sessions and project XML files).

13.1.4 Creating TopLink Metadata

The TopLink metadata is the bridge between the development of an application and its deployed runtime environment. You can capture the metadata using:

- JDeveloper Mapping Editor, which creates TopLink `sessions.xml` and `project.xml` files that you pass to the TopLink runtime environment.
- JPA annotations, `persistence.xml`, `orm.xml`, and TopLink JPA annotation and TopLink property extensions. The TopLink JPA persistence provider interprets these metadata sources of metadata to create an in-memory TopLink session and project at runtime.
- Java and the TopLink API (this approach is the most labor-intensive).

The metadata enables you to pass configuration information into the runtime environment, which uses the information in conjunction with the persistent classes (Java objects, JPA entities, or EJB entity beans) and the code written with the TopLink API, to complete the application.

Using TopLink JPA, you also have the option of specifying your metadata using TopLink `sessions.xml` and `project.xml` while accessing your persistent classes using JPA and an EntityManager.

The TopLink metadata architecture provides many important benefits, including the following:

- By using the metadata, TopLink does not intrude in the object model or the database schema.
- Allows you to design the object model as needed, without forcing any specific design.
- Allows DBAs to design the database as needed without forcing any specific design.
- Does not rely on code-generation (which can cause serious design, implementation, and maintenance issues).
- Is unobtrusive: adapts to the object model and database schema, rather than requiring you to design their object model or database schema to suit TopLink.

Using TopLink JPA, you have the flexibility of expressing persistence metadata using standard JPA annotations, deployment XML, or both. Optionally, you can take advantage of TopLink JPA annotation and persistence unit extensions.

13.1.5 Creating Project Metadata

A TopLink project contains the mapping metadata that the TopLink runtime uses to map objects to a data source. The project is the primary object used by the TopLink runtime. The principal contents of project metadata include the following:

- Descriptors
- Mappings
- Data Source Login Information

Using JPA, TopLink runtime constructs an in-memory project based on the employed annotations, `persistence.xml`, `orm.xml`, and TopLink JPA extensions.

13.1.6 Creating Session Metadata

The TopLink Session configuration file (`sessions.xml`) allows you to easily manage all of the sessions for a specific project. You can fully customize the information for each session, including your data source login information, JTA transaction usage, and caching.

A TopLink session contains a reference to a particular `project.xml` file, plus the information required to access the data source. The session is the primary object used by your application to access the features of the TopLink runtime.

The agent responsible for creating and accessing session metadata differs, depending on whether or not you are creating a CMP project. In a POJO project, your application acquires and accesses a session directly. In a CMP project, your application indirectly accesses a session acquired internally by the TopLink runtime.

Using TopLink JPA, the TopLink runtime constructs an in-memory session based on any combination of JPA annotations, `persistence.xml`, `orm.xml`, and TopLink JPA annotation and persistence.xml property extensions. The use of a `sessions.xml` file is optional.

13.1.7 Using TopLink Descriptors

TopLink uses descriptors to store the information that describes how a particular class can be represented by a data source. Descriptors own mappings that associate class instance variables with a data source and transformation routines that are used to store and retrieve values. As such, the descriptor acts as the connection between a Java object and its data source representation.

Two objects – a source (parent or owning) object and a target (child or owned) object are related by aggregation if there is a strict one-to-one relationship between them, and all the attributes of the target object can be retrieved from the same data source representation as the source object. This means that if the source object exists, then the target object must also exist, and if the source object is destroyed, then the target object is also destroyed.

JDeveloper enables you to create the following TopLink descriptor types:

- Relational Descriptors
- EIS Descriptors
- XML Descriptors

13.1.7.1 Relational Descriptors

Relational descriptors describe Java objects that you map to tables in a relational database. Using relational descriptors in a relational project, you can configure relational mappings. In a relational project, you can designate the descriptor as an aggregate, enabling you to configure an aggregate mapping, one that associates data members in the target object with fields in the source object's underlying database tables.

When you designate a relational descriptor as an aggregate, TopLink lets you specify a mapping type for each field in the target class, but defers associating the field with a database table until you configure the aggregate object mapping in the source descriptor. In other words, the target class descriptor defines how each target class field is mapped, but the source class descriptor defines where each target class field is mapped. This lets you share an aggregate object among many parent descriptors mapped to different tables.

13.1.7.2 EIS Descriptors

Describes Java objects that you map to an EIS data source by way of a JCA adapter. EIS descriptors enable you to configure EIS mappings when creating an EIS project.

13.1.7.3 XML Descriptors

Describes Java objects that you map, in memory, to complex types in XML documents defined by an XML schema document (XSD). Using XML descriptors in an XML project, you can configure XML mappings in memory, to XML elements defined by an XSD.

13.1.8 Using TopLink Mappings

TopLink transforms the data from an object representation to a representation specific to a data source. This transformation is called mapping and it is the core of a TopLink project. A mapping corresponds to a single data member of a domain object. It associates the object data member with its data source representation and defines the means of performing the two-way conversion between the object and data source. A TopLink map belongs to a TopLink session, the facade through which applications access TopLink functionality. The available mapping types may vary, depending on the TopLink map and TopLink descriptor.

13.1.8.1 Relational Mapping Types

The relational mappings transform any object data member type to a corresponding relational database representation in any supported relational database. Use them to map simple data types including primitives (such as int), JDK classes (such as String), and large object (LOB) values. You can also use them to transform object data members that reference other domain objects by way of association where data source representations require object identity maintenance (such as sequencing and back references) and possess various types of multiplicity and navigability. The appropriate mapping class is chosen primarily by the cardinality of the relationship

Table 13–1 illustrates the relational mapping types build maps using the TopLink concepts of directionality, transformers, converters, and EJB 2.n CMP relational mapping.

Table 13–1 Relational Mapping Types

Mapping Type	Description
Direct-to-field	Map a Java attribute directly to a database field.
Direct-to-XMLType	Map Java attributes to an `XMLType` column in an Oracle Database.
One-to-one	Map a reference to another persistent Java object to the database.
Variable one-to-one	Map a reference to an interface to the database.
One-to-many	Map Java collections of persistent objects to the database.
Many-to-many	Use an association table to map Java collections of persistent objects to the database.
Direct collection	Map Java collections of objects that do not have descriptors
Direct map	Direct map mappings store instances that implement `java.util.Map`.
Aggregate object	Create strict one-to-one mappings that require both objects to exist in the same database row.
Transformation	Create custom mappings where one or more fields can be used to create the object to be stored in the attribute.

13.1.8.2 EIS Mapping Types

TopLink enterprise information system (EIS) mappings provide support for accessing legacy data sources and enterprise applications through Java EE Connector architecture (JCA) adapter. TopLink EIS mappings use the JCA Common Client Interface (CCI) to access the EIS through its resource adapter. This provides the ability to directly map from an existing Java object model to any transactional data source, such as mainframes with flat file/hierarchical data. An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

Table 13–2 illustrates the EIS mapping types that TopLink provides:

Table 13–2 EIS Mapping Types

Mapping Type	Description
Direct mapping	Map a simple object attribute directly to an EIS record.
Composite direct collection mapping	Map a collection of Java attributes directly to an EIS record.
Composite object mapping	Map a Java object to an EIS record in a privately owned one-to-one relationship. Composite object mappings represent a relationship between two classes.
Composite collection mapping	Map a `Map` or `Collection` of Java objects to an EIS record in a privately owned one-to-many relationship.
One-to-one mapping	Define a reference mapping that represents the relationship between a single source object and a single mapped persistent Java object.
One-to-many mapping	Define a reference mapping that represents the relationship between a single source object and a collection of mapped persistent Java objects.
Transformation mapping	Create custom mappings where one or more EIS record fields can be used to create the object to be stored in a Java class's attribute.

13.1.8.3 XML Mapping Types

The XML mappings transform object data members to the XML elements of an XML document whose structure is defined by an XML schema document (XSD). You can map the attributes of a Java object to a combination of XML simple and complex types using a wide variety of XML mapping types. TopLink stores XML mappings for each class in the class descriptor. TopLink uses the descriptor to instantiate objects mapped from an XML document and to store new or modified objects as an XML document.

Table 13–3 indicates the XML mapping types you can use to map the attributes of a Java object to a combination of XML simple and complex types:

Table 13–3 XML Mapping Types

Mapping Type	Description
XML Direct Mapping	Map a simple object attribute to an XML attribute or text node.
XML Composite Direct Collection Mapping	Map a collection of simple object attributes to XML attributes or text nodes.
XML Composite Object Mapping	Map any attribute that contains a single object to an XML element. The TopLink runtime uses the descriptor for the referenced object to populate the contents of that element.
XML Composite Collection Mapping	Map an attribute that contains a homogenous collection of objects to multiple XML elements. The TopLink runtime uses the descriptor for the referenced object to populate the contents of those elements.
XML Any Object Mapping	The XML Any Object mapping is similar to the XML Composite Object mapping except that the reference object may be of different types (including `String`), not necessarily related to each other through inheritance or a common interface.
XML Any Collection Mapping	The XML Any Collection mapping is similar to the XML Composite Collection mapping except that the referenced objects may be of different types (including `String`), not necessarily related to each other through inheritance or a common interface.
XML Transformation Mapping	Create custom mappings where one or more XML nodes can be used to create the object to be stored in a Java class's attribute.

13.1.9 Understanding the TopLink Editor

Use the TopLink editor to configure and map Java classes to different data sources, including relational databases, enterprise information systems (EIS), and XML schemas without using code. The TopLink editor supports multiple mapping standards, including EJB 3.0 JPA.

The TopLink editor displays the information or properties specific to the element selected in the Application Navigator or the Structure view. For example, selecting TopLink project elements in the Application Navigator, such as a the TopLink Map or the sessions configuration file (sessions.xml), enables you to configure their properties in the TopLink editor. Likewise, selecting TopLink Maps, descriptors, and mapped or unmapped attributes in the Structure view results in the display of their respective properties in the TopLink editor.

13.1.9.1 Managing TopLink Maps

The TopLink Map contains the information about how classes map to database tables or XML schema. Use the TopLink editor to edit each component of the mappings, including:

- Database information, such as driver, URL, and login information.

- Mapping defaults, such as identity map and cache options.

To configure a TopLink Map, choose Application Navigator context menu for a TopLink Map (for example, tlMap) Open or choose the Structure view for TopLink Map. The TopLink editor displays the properties for the object map depending on its type, such as relational, or EIS. When using the TopLink editor for relational object maps, for example, you can configure the sequencing policy. For more information about TopLink editor, see Oracle Fusion Middleware Developer's Guide for Oracle TopLink.

TopLink mappings use descriptors to store the information that describes how an instance of a particular class can be represented in the data source. To configure a map's descriptors, choose Structure view for tlMap descriptor. For example, using the editor, you can improve application performance by creating named queries and also prevent users from overwriting each other's work by configuring locking policies.

TopLink mappings define how an object's attributes are represented in the data source. The Structure view enables you to configure the mappings for the descriptor's attributes by choosing Structure view for tlMap descriptor attribute Map as context menu mapping type.

13.1.9.2 Managing TopLink Sessions

The TopLink Sessions configuration file (sessions.xml) enables you to manage all of the sessions for a specific project. For more information about TopLink sessions, see Oracle Fusion Middleware Developer's Guide for Oracle TopLink.

By choosing Structure view for sessions.xml Open, you can use the TopLink editor to fully customize the information for each session, such as data source login information, JTA transaction usage, and caching. You can also use the TopLink editor to create and configure individual sessions and the session brokers that manage them. To manage session brokers, Structure view for sessions.xml session broker.

13.1.9.3 Managing Persistence Configurations

The TopLink editor enables you to configure the persistence.xml file, which packages entities in TopLink JPA projects. By choosing Application Navigator for persistence.xml Open, you can create persistence units.

The Structure window displays JPA descriptors and persistence units. By choosing Structure view for persistence.xml persistence unit, you can configure the persistence unit.

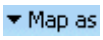
13.1.9.4 The TopLink Structure View Toolbar

The Structure view displays detailed information about the TopLink element selected in Application Navigator or TopLink editor. For example:

- When working with an EJB or Java class, the Structure view displays the related TopLink descriptor and its mapping attributes.
- When working with a TopLink sessions configuration file, the Structure view displays sessions and session brokers.
- When working with a persistence configuration, the Structure view displays JPA descriptors and persistence units.

The Structure view contains a toolbar that provides access to modify descriptors, mapping, sessions, and persistence units. This toolbar is context-sensitive; the buttons displayed vary depending on the element that you select in the Structure view.

Table 13–4 Icons in the TopLink Structure View Toolbar

Icon	Name	Function
	**Add or Remove Descriptors**	Adds or removes descriptors from the TopLink map
	**Automap**	Attempts to automap the selected descriptor or attribute to a similarly named database field.
	**Aggregate Descriptor**	Changes the descriptor type to aggregate descriptor, meaning that the descriptor's definitions for table, primary key and other options are from the owning descriptor.
	**Class Descriptor**	Changes the descriptor type to class descriptor.
	**Map As**	Selects a mapping type for the selected attribute.
	**New Persistence Unit**	Click to create a new persistence unit.
	**Create a New Database or Server Session**	Click to create a session within the sessions configuration file.
	**Create Session Broker**	Click to create a new session broker.
	**Create a New Named Connection Pool**	Click to create a new named connection pool, a connection pool used for any purpose, but typically for security purposes.
	**Add the Sequence Connection Pool**	Click to add a connection pool exclusively used for sequencing. TopLink uses the sequence connection pool whenever it needs to assign an identifier to a new object.

13.1.9.5 TopLink Project Elements in the Application Navigator

The Application Navigator displays each element associated with your TopLink project, including the TopLink Map, deployment descriptors, and sessions configuration information.

TopLink project elements in the Application Navigator may include:

- TopLink folder
- Sessions configuration file (`sessions.xml`)
- TopLink map (tlMap)

13.1.9.6 TopLink Editor Tabs in the Editor Window

The TopLink Editor displays your TopLink mapping information. The information in the editor will vary, depending on the TopLink element you selected in the Application Navigator or Structure view.

13.1.9.7 TopLink Project Elements in the Structure View

The Structure view displays detailed information about the TopLink element selected in Application Navigator or TopLink Editor:

- When working with an EJB or Java class, the Structure view displays the related TopLink descriptor and its mapping attributes.
- When working with a TopLink sessions configuration file, the Structure view displays your sessions and session brokers.
- When working with a persistence configuration, the Structure view displays your JPA descriptors and persistence units.

When you select an item in the Structure view, the following properties appear in the TopLink Editor:

- TopLink map (tlMap)
- Descriptor
- Mapped Java attribute (one-to-one mapping)
- Unmapped attribute

You can perform specific functions for an item by selecting the item in the Application Navigator and then:

- Right-clicking the object in Structure view and selecting the function from the pop-up menu.
- Selecting the object in Structure view and clicking a button in the Structure toolbar.

13.1.9.8 Using the TopLink Structure View Toolbar

The TopLink Editor Structure view contains a toolbar that offers quick access to modify descriptors and mappings. This toolbar is context-sensitive; the actual buttons displayed will vary, depending on which element in the Structure view is selected.

13.1.9.9 TopLink Mapping Status Report in Message Log

Error and status messages from the TopLink Editor appear in the TopLink Problems window.

13.1.9.10 Configuring TopLink Preferences

You can configure which persistence provider to use, which JPQL editor to use, and query types and formats.

To configure TopLink Editor preferences:

1. Select **Tools** > **Preferences**.
2. In the **Categories** list, expand **TopLink Customization**.
3. Configure JPA and Mappings options.
4. Complete each field and click **OK**.

13.1.9.11 How to Create a TopLink Mapping Project

JDeveloper stores the TopLink descriptors (for more information, see Section 13.1.7, "Using TopLink Descriptors") and mappings (for more information, see Section 13.1.8, "Using TopLink Mappings") in a TopLink map (`.mwp` file), and sessions in the sessions.xml file. The TopLink map contains the information about how classes map to

database tables. Use the TopLink Editor to edit each component of the mappings, including:

- Database information, such as driver, URL, and login information.
- Mapping defaults, such as cache options.

When you select a TopLink map (or an element in a TopLink map), its attributes display in the TopLink Editor.

TopLink maps persistent entities to the database in the application using the descriptors and mappings you build with JDeveloper Mapping Editor. The Mapping Editor supports such approaches to project development as:

- Importing classes and tables for mapping.
- Importing classes and generating tables and mappings.
- Importing tables and generating classes and mappings.
- Creating both class and table definitions.

Although JDeveloper Mapping Editor offers the ability to generate persistent entities or the relational model components for an application, these utilities are intended only to assist in rapid initial development strategies–not complete round-trip application development.

To create a new TopLink-enabled project:

1. Select **File** > **New**.
2. In the **Categories** list, choose **General** > **Projects**.
3. In the **Items** list, select **TopLink Project**.
4. Click **OK**.

   The New TopLink-Enabled Project dialog displays.
5. Complete each field and click **OK**.

   JDeveloper creates a new project, including an object map.

To add a TopLink map to an existing JDeveloper project:

1. Right-click an existing project in the Application Navigator and choose **New**.
2. In the Categories list, choose **Business Tier** > **TopLink/JPA**.
3. In the Items list, select **TopLink Object Map**.
4. Complete each field and click **OK**.

   JDeveloper creates a TopLink map file in an existing project.

13.1.9.12 How to Use Converter Mappings

TopLink no longer uses the following direct mapping types:

- Type conversion
- Object type
- Serialized object

Instead, TopLink uses a direct-to-field mapping with a specialized converter. To generate backward-compatible deployment XML files, use the **Generate Deprecated Direct Mappings** option on the General page of the TopLink Map options.

13.1.9.13 How to Automap TopLink Descriptors

The TopLink Automap wizard can automatically map your Java class attributes to a similarly named database field. The Automap wizard only creates mappings for unmapped attributes; it does not change previously defined mappings.

You can use the Automap wizard for an entire project or for specific classes or descriptors.

To automap TopLink descriptors:

1. In the Application Navigator, select a **TopLink Map**.

   The TopLink Map (and its attributes) appear in the Structure window.

2. In the Structure window, right-click the TopLink Map (or a specific Java class or TopLink descriptor) and choose **Automap**.

   The Automap Wizard wizard displays. Complete each page of the wizard.

13.1.9.14 Data Source Login Information

For TopLink mappings, you can configure a session login in the session metadata that specifies the information required to access the data source.

13.2 Developing TopLink JPA Projects

Use a TopLink JPA (Java Persistence API) project for persisting Java objects based on Plain Old Java Objects (POJOs).

The Java Persistence API is a lightweight framework for Java persistence based on Plain Old Java Objects. JPA is a part of EJB 3.0 specification. JPA provides an object-relational mapping approach that enables you to declaratively define how to map Java objects to relational database tables in a standard, portable way. In addition, this API enables you to create, remove and query across lightweight Java objects within both an EJB 3.0-compliant container and a standard Java SE 5 and Java SE 6 environment.

The TopLink implementation of JPA is provided by EclipseLink. For more information, see `http://wiki.eclipse.org/EclipseLink`.

You can perform object relational mapping with TopLink JPA through the following:

Table 13–5 Methods for Performing Object Relational Mapping with TopLink JPA

Method	Description
Using Metadata Annotations	An annotation is a simple, expressive means of decorating Java source code with metadata that is compiled into the corresponding Java class files for interpretation at run time by a JPA persistence provider to manage persistent behavior.You can use annotations to configure the persistent behavior of your entities.
Using XML	You can use XML mapping metadata on its own, or in combination with annotation metadata, or you can use it to override the annotation metadata.
Defaulting Properties	Each annotation has a default value. A persistence engine defines defaults that apply to the majority of applications. To override the default value, you need only to supply the appropriate values. A configuration value is not a requirement, but the exception to the rule. This is known as *configuration by exception*.

Table 13–5 (Cont.) Methods for Performing Object Relational Mapping with TopLink JPA

Method	Description
Configuring an Entity	You can configure an entity's identity, as well as the locking technique and sequence generation option for the entity.
Declaring Basic Property Mappings	Simple Java types are mapped as part of the immediate state of an entity in its fields or properties. Mappings of simple Java types are called basic mappings. By default, TopLink persistence provider automatically configures a basic mapping for simple types.
Mapping Relationships	TopLink persistence provider requires that you map relationships explicitly. Use such annotations as `@OnetoOne`, `@ManytoOne`, `@OnetoMany`, `@ManytoMany`, `@Mapkey`, and `@Orderby` to specify the type and characteristics of entity relationships that fine-tune how the database implements relationships.
Mapping Inheritance	By default, TopLink persistence provider assumes that all persistent fields are defined by a single entity class. Use the `@Inheritance`, `@MappedSuperclass`, `@DiscriminatorColumn`, and `@DiscriminatorValue` annotations if your entity class inherits some or all persistent fields from one or more superclasses.
Mapping Embedded Objects	An embedded object does not have its own persistent identity. It is dependent upon an entity for its identity. By default, TopLink persistence provider assumes that every entity is mapped to its own table. Use the following annotations to override this behavior for entities that are owned by other entities: ■ `@Embeddable` ■ `@Embedded` ■ `@AttributeOverride` ■ `@AttributeOverrides` ■ `@AssociationOverride` ■ `@AssociationOverrides`

13.2.1 How to Create and Configure a JPA Persistence Descriptor (persistence.xml)

Use the persistence configuration file (persistence.xml) file to package your entities.

To create a persistence configuration:

1. Select **File** > **New**.
2. In the **Categories** list, select **Business Tier > TopLink/JPA**.
3. In the Items list, select **JPA Persistence Descriptor**.
4. Click **OK**.

   The New JPA Persistence Descriptor dialog displays.
5. Complete the fields to create a default persistence unit for the new JPA persistence descriptor file (`persistence.xml`) and click **OK**.

   Example 13–1 contains a sample persistence configuration.

Example 13–1 Sample Persistence Configuration (persistence.xml)

```
<?xml version="1.0" encoding="windows-1252" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
```

```
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
 <persistence-unit name="myPersistenceUnit">
  <properties>
     <property name="toplink.target-database" value="Oracle11g"/>
      <property name="toplink.target-server" value="WebLogic_10"/>
  </properties>
 </persistence-unit>
 <persistence-unit name="myPersitenceUnitName">
...
 </persistence-unit>
</persistence>
```

To configure a persistence configuration (persistence.xml) file:

1. Select the **persistence.xml** node in the Application Navigator.
2. In the Structure window, select the **JPA Persistence Descriptor**.
3. Complete the **General** and **Metadata Preferences** tabs on the JPA Persistence descriptor (`persistence.xml`) page.

13.2.2 How to Create Persistence Units

To create a persistence unit:

1. Double-click the persistence configuration file (`persistence.xml`) in the Application Navigator or Structure window.
2. On the General page, click **Create New Persistence Unit** to create a new persistence unit.
3. Complete each field on the New Persistence Unit dialog.
4. On the Metadata Preferences page, specify how to persist new mapping metadata. You can specify annotations or JPA mapping descriptors. In order to make this choice, you must first create at least one `orm.xml` mapping descriptor file. (See Section 13.2.4, "How to Create JPA Descriptors.")

   Example 13–2 contains a sample persistence unit.

Example 13–2 Example Persistence Unit

```
...
 <persistence-unit name="myPersitenceUnitName"
          transaction-type="RESOURCE_LOCAL">
    <mapping-file>META-INF/orm.xml</mapping-file>
    <exclude-unlisted-classes/>
    <properties>
      <property name="eclipselink.jdbc.driver"
                value="oracle.jdbc.OracleDriver"/>
      <property name="eclipselink.jdbc.url"
                value="jdbc:oracle:thin:@localhost:1521:XE"/>
      <property name="eclipselink.jdbc.user" value="scott"/>
      <property name="eclipselink.jdbc.password"
                value="3E20F8982C53F4ABA825E30206EC8ADE"/>
      <property name="eclipselink.target-database" value="Oracle11g"/>
      <property name="eclipselink.logging.level" value="FINER"/>
      <property name="eclipselink.jdbc.native-sql" value="true"/>
      <property name="eclipselink.target-server" value="WebLogic_10"/>
    </properties>
```

```
 </persistence-unit>
...
```

13.2.3 How to Configure Persistence Units

The tabs of the Persistence Unit page (accessed by first selecting `persistence.xml` in the Application Navigator and then expanding the JPA descriptor in the Structure view) enable you to configure a persistence unit.

Configuring Persistence Units encompasses many steps, such as configuring:

- General information
- Connection information
- TopLink information
- Schema generation information
- Properties
- Metadata information

To configure the general information for a JPA persistence unit:

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **General** tab to specify how the persistence unit connects to the application server and database.

To configure the connection information for a JPA persistence unit:

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **Connection** tab to select a persistence provider and configure its general properties (such as JPA mapping descriptors, Java archives, and mapped classes).

To configure the TopLink session-specific information for a JPA persistence unit:

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **TopLink Customization** tab to specify TopLink-specific information for the persistence unit.

To configure the DDL generation options:

Although most JPA persistence providers provide this support, these options are TopLink-specific.

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **Schema Generation** tab to specify how the TopLink generates the DDL scripts.

To configure the non-TopLink specific properties for the persistence unit:

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **Properties** tab to specify the general, non-TopLink specific properties.

To configure metadata overrides for a persistence unit:

1. Select the JPA persistence descriptor (`persistence.xml`) in the Application Navigator.
2. Select the persistence unit in the Structure window.

   The Persistence Unit page displays.
3. Complete the fields on the **Metadata Preferences** tab to specify the information for the mapping descriptor.

   This tab is available only if the persistence unit contains a JPA mapping descriptor.

13.2.4 How to Create JPA Descriptors

The JPA mapping descriptor is used as an alternative to annotations. Any information you add as a JPA mapping descriptor will override the Java annotations.

To create new JPA mapping descriptors:

1. Select the persistence configuration (`persistence.xml`) in the Application Navigator.
2. Select the **General** tab.
3. In the JPA Mapping Descriptors area, click the **Create New JPA Mapping Descriptor** button.
4. Complete the fields on the dialog and click **OK**.

   JDeveloper adds the Mapping Descriptors (`orm.xml`) to the project.
5. Complete the following tabs for each JPA descriptor on the ORM (`orm.xml`) page:
   - General
   - Persistence Unit Defaults
   - Generators
   - Queries

To configure the general information for a JPA mapping descriptor:

1. Select the JPA mapping descriptor (`orm.xml`) in the Application Navigator.
2. Select the descriptor in the Structure window.

   The ORM (`orm.xml`) page appears.
3. Complete the fields on the **General** tab to select a persistence provider and configure its general properties (such as mapped classes, development database, and other defaults).

13.2.4.1 How to Configure Persistence Unit Defaults

You can configure the settings that apply to persistence units and associated entities that include this mapping descriptor. These values will be overridden by any configuration settings at the persistence unit-level.

To configure persistence unit defaults:

1. Select the JPA mapping descriptor (`orm.xml`) in the Application Navigator.
2. Select the descriptor in the Structure window.

   The ORM (`orm.xml`) page appears.
3. Complete the fields on the **Persistence Unit Defaults** tab to configure the access type, entity listeners, and other defaults.

13.2.4.2 How to Configure Generators

You can define the generators used by this mapping descriptor.

To configure generators:

1. Select the JPA mapping descriptor (`orm.xml`) in the Application Navigator.
2. Select the descriptor in the Structure window.

   The ORM (`orm.xml`) page appears.
3. Complete the fields on the **Generators** tab to configure the database sequence and table generators.

13.2.4.3 How to Configure Queries

You can define the JPQL and native queries in this mapping descriptor for use in associated persistence units.

To configure queries:

1. Select the JPA mapping descriptor (`orm.xml`) in the Application Navigator.
2. Select the descriptor in the Structure window.

   The ORM (`orm.xml`) page appears.

13.2.5 Using JPA Mappings

Oracle TopLink provides a complete, JPA 2.0-compliant JPA implementation. It provides complete compliance for all of the mandatory features, many of the optional features, and some additional features.

TopLink offers support for deployment within an EJB 3.0 container or outside the container. This includes Web containers, other non-EJB 3.0 Java EE containers, and the Java SE environment.

Through its pluggable persistence capabilities TopLink can function as the persistence provider in any compliant EJB 3.0 container.

13.2.6 Using TopLink Extensions

The Java Persistence API (JPA), part of the Java Enterprise Edition 5 (Java EE 5) EJB 3.0 specification, greatly simplifies Java persistence. It provides an object relational mapping approach that allows you to declaratively define how to map Java objects to relational database tables in a standard, portable way that works both inside a Java EE 5 application server and outside an EJB container in a Java Standard Edition (Java SE) 5 application.

TopLink JPA provides extensions to what is defined in the JPA specification. These extensions come in persistence unit properties, query hints, annotations, TopLink's own XML metadata, and custom API.

13.3 Developing TopLink Relational Projects

The TopLink Editor provides complete support for creating relational projects that map Java objects to a conventional relational database accessed using JDBC. Use a TopLink relational project for transactional persistence of Java objects to a conventional relational database or to an object-relational database that supports data types specialized for object storage, both accessed using JDBC.

To create relational projects for an object-relational database, you must create the project using Java code. You can create a relational project for transactional persistence of Java objects to an object-relational database that supports data types specialized for object storage (such as Oracle Database) accessed using JDBC.

13.3.1 How to Create Relational Projects and Object Maps

To create relational projects for an object-relational database, you must create the project using Java code. You can create a relational project for transactional persistence of Java objects to an object-relational database that supports data types specialized for object storage (such as Oracle Database) accessed using JDBC.

To create a new relational project:

1. Select **File** > **New**.
2. In the Categories list select **General** > **Projects**.
3. In the Items list, select **TopLink Project**.
4. Click **OK**.

   The New TopLink-Enabled Project dialog displays.
5. Complete the fields on the dialog to specify the project name, project location, and TopLink map.
6. In the Data Source area, select **Database**, then specify your specific database information.
7. Click **OK**.

JDeveloper creates a new project, including the TopLink map and TopLink sessions configuration file (`sessions.xml`).

To create a new TopLink object map for a relational project:

1. Select **File** > **New**.
2. In the Categories list select **Business Tier** > **TopLink/JPA**.
3. In the Items list, select **TopLink Object Map**.
4. Click **OK**.

   The New TopLink Object Map dialog displays.
5. Complete the fields on the dialog to specify the TopLink map.
6. In the Data Source area, select **Database**, then specify your specific database information.
7. Click **OK**.

   JDeveloper creates a new project, including the TopLink map and TopLink sessions configuration file (`sessions.xml`).

13.3.2 How to Create Relational Descriptors

Relational descriptors describe Java objects that you map to tables in a relational database. In a relational project, you can designate the descriptor as a class descriptor or an aggregate descriptor.

A class descriptor is applicable to any persistent object, but not an aggregate object. Using a class descriptor, you can configure any relational mapping except aggregate collection and aggregate object mappings.

An aggregate object is an object that is strictly dependent on its owning object. Aggregate descriptors do not define a table, primary key, or many of the standard descriptor options as they inherit these from their owning descriptor. If you want to configure an aggregate mapping to associate data members in a target object with fields in a source object's underlying database tables, you must designate the target object's descriptor as an aggregate.

You can configure inheritance for a descriptor designated as an aggregate, however, in this case, all the descriptors in the inheritance tree must be aggregates. Aggregate and class descriptors cannot exist in the same inheritance tree.

You can change a class descriptor to an aggregate descriptor, or remove the aggregate designation from a relational descriptor and return it to its default type. For more information, see Section 13.3.3, "How to Configure Relational Descriptors.".

Note: When you change a class descriptor to an aggregate descriptor, the descriptor's existing information is permanently lost. If you convert the descriptor back to a class descriptor, you will have to configure it again.

To create new TopLink descriptors:

1. Right-click the TopLink Map in the Application Navigator and select **Add** or **Remove Descriptors**.
2. Select the packages and classes from which to create TopLink descriptors and click **OK**.

JDeveloper adds the descriptors to the TopLink Map in the Structure window.

13.3.3 How to Configure Relational Descriptors

You can configure a relational descriptor as a Class type or an Aggregate type. By default, when you add a Java class to a relational project, JDeveloper automatically creates a relational class descriptor for it.

You can change a class descriptor to an aggregate descriptor.

To configure a TopLink relational class descriptor to an aggregate descriptor:

1. Select the TopLink Map in the Application Navigator.
2. In the Structure window, right-click the descriptor and from the **Descriptor Type** submenu, select **Aggregate**.

   The selected descriptor is now an aggregate descriptor.
3. To convert an aggregate descriptor to a class descriptor, right-click the descriptor and from the **Descriptor Type** submenu, select **Class**.

13.4 Developing TopLink XML Projects

Use an XML project for nontransactional conversions between Java objects and XML documents using JAXB (Java Architecture for XML Binding) which defines annotations to control the mapping of Java objects to XML.

The TopLink runtime performs XML data conversion based on one or more XML schemas. In an XML project, the TopLink Editor directly references schemas in the deployment XML and exports mappings configured with respect to the schemas you specify.

TopLink provides an extra layer of functions on top of JAXB. In particular, TopLink provides the TopLink JAXB compiler, which generates both JAXB- and TopLink-specific files.

The JAXB complier generates implementation classes that are named according to the content, element, or implementation of the name attribute in the XSD. The generated implementation classes are simple domain classes with private attributes for each JAXB property. Public get and set methods return or set attribute values.

The JAXB complier generates TopLink project files, `session.xml` files, and TopLink project XML files. The TopLink JAXB compiler generates a single class called `DescriptorAfterLoads` if any implementation class contains a mapping to a type safe enumeration.

TopLink can validate both complete object trees and subtrees against the XML schema that was used to generate the implementation classes. In addition, TopLink will validate both root objects (objects that correspond to the root element of the XML document) and non-root objects against the schema used to generate the object's implementation class.

JAXB provides a standard Java object-to-XML API. JAXB defines annotations to control the mapping of Java objects to XML. For more information, see `http://www.oracle.com/technetwork/java/index-jsp-137051.html`.

JAXB also defines a default set of mappings which TopLink uses to marshal a set of objects into XML, and unmarshall an XML document into objects. TopLink provides an extra layer of functions on top of JAXB. It allows for the creation and subsequent

manipulation of TopLink mappings from an existing object model, without requiring the recompilation of the JAXB object model.

13.4.1 How to Create XML Projects

To create a new XML project:

1. Select **File > New**.
2. In the **Categories** list select **General > PropertiesTopLink**.
3. In the Items list, select **TopLink Project**.
4. Click **OK**.

   The **New TopLink-Enabled Map** dialog appears.
5. Complete the fields on the dialog to specify the project name, project location, and TopLink map.
6. In the Data Source area, select **Database**, then specify your specific database information.
7. Click **OK**.

   JDeveloper adds the TopLink map and TopLink sessions configuration file (`sessions.xml`).

13.4.2 How to Create XML Object Maps

To create a new TopLink object map:

1. Select **File > New**.
2. In the **Categories** list select **Business Tier > TopLink/JPA**.
3. In the Items list, select **TopLink Object Map**.
4. Click **OK**.

   The **New TopLink-Enabled Map** dialog appears.
5. Complete the fields on the dialog to specify the project name, project location, and TopLink map.
6. In the Data Source area, select **XML**.
7. Click **OK**.

   JDeveloper adds the TopLink map and TopLink sessions configuration file (`sessions.xml`).

13.4.3 How to Create XML Descriptors

To create new TopLink descriptors for an XML project:

1. Right-click the TopLink Map in the Application Navigator and select **Add or Remove Descriptors**.
2. Select the packages and classes from which to create TopLink descriptors and click **OK**.

   JDeveloper adds the descriptors to the TopLink element in the Structure window.
3. Complete the fields on the XML Descriptor page to configure the descriptor.

13.4.4 How to Add XML Schemas

If you have an existing data model (XML schema document), but you do not have a corresponding object model (Java classes for domain objects), use this procedure to create your TopLink project and automatically generate the corresponding object model.

To add an XML schema:

1. Select the TopLink map in the Application Navigator.
2. In Structure window, right-click the Schemas element and select **Import Schema**.
3. Complete the fields on the dialog to specify the XML schema to import.
4. Click **OK**.

   JDeveloper adds the schema (tlmap) to the TopLink map

Using the TopLink JAXB compiler simplifies JAXB application development with TopLink by automatically generating both the required JAXB files and the TopLink files from your XML schema (XSD) document. Once generated, you can fine-tune XML mappings without having to recompile your JAXB object model.

13.5 Developing TopLink EIS Projects

Use a TopLink EIS project for transactional persistence of Java objects to a nonrelational data source accessed using a Java EE Connector Architecture (JCA) adapter and EIS records.

Oracle recommends using EIS projects to integrate TopLink with a legacy or nonrelational data source. TopLink provides support for mapping Java objects to EIS mapped, indexed, and XML records, through J2C, using the TopLink mappings. J2C provides a Common Client Interface (CCI) API to access nonrelational EIS. This provides a similar interface to nonrelational data sources as JDBC provides for relational data sources.

EIS includes legacy data sources, enterprise applications, legacy applications, and other information systems. These systems include such sources as Customer Information Control System (CICS), Virtual Storage Access Method (VSAM), Information Management System (IMS), ADABASE database, and flat files. Oracle recommends using EIS projects to integrate TopLink with a legacy or nonrelational data source. Other methods of accessing EIS data sources include:

- Using a specialized JDBC driver that allows connecting to an EIS system as if it were a relational database. You could use a TopLink relational project with these drivers.
- Linking to or integrating with the EIS data from a relational database, such as Oracle Database.
- Using a proprietary API to access the EIS system. In this case it may be possible to wrap the API with a JCA CCI interface to allow usage with a TopLink EIS project.

13.5.1 How to Create EIS Projects

Use an EIS project for transactional persistence of Java objects to a nonrelational data source accessed using a Java EE Connector Architecture (JCA) adapter and EIS records.

To create an EIS project:

1. Select **File** > **New**.
2. In the Categories list select **General** > **Projects**.
3. In the Items list select **TopLink Project**.
4. Click **OK**.
5. Complete the fields on the dialog to specify the project name, project location, and TopLink map.
6. In the Data Source area, select **EIS**, then specify your specific EIS platform.
7. Click **OK**.

   JDeveloper creates a new project, including the TopLink map and TopLink sessions configuration file (`sessions.xml`).

13.5.2 How to Create EIS Object Maps

An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

To create a new TopLink object map for an EIS project:

1. Select **File** > **New**.
2. In the Categories list select **Business Tier** > **TopLink/JPA**.
3. In the Items list select **TopLink Object Map**.
4. Click **OK**.
5. Complete the fields on the dialog to specify the TopLink map.
6. In the Data Source area, select **EIS**, then specify your specific EIS platform.
7. Click **OK**.

   JDeveloper adds the TopLink map to the project.

13.5.3 How to Create EIS Descriptors

EIS descriptors describe Java objects that you map to an EIS data source by way of a JCA adapter.

To create an EIS descriptor:

1. Select the TopLink map in the Structure window.
2. Click the **Add or Remove Descriptors from the Selected TopLink Map** button.
3. Select the classes from which to create an EIS descriptor and click **OK**.

   JDeveloper adds the EIS descriptors to the Structure window.
4. Complete the property tabs for the EIS Descriptor.

13.5.4 Using EIS Data Sources

For each EIS project, you must specify one of the following JCA data source platforms that you will be using:

- Oracle AQ

- Attunity Connect
- IBM MQSeries

This platform configuration is overridden by the session login, if configured.

13.6 Developing TopLink Sessions

Each TopLink map belongs to aTopLink session. A session is the facade through which an application accesses TopLink functionality. A session associates data source platform information, data source login information, and mapping metadata for a particular application. You can reuse mapping metadata in different applications by defining different sessions.

TopLink session provides the primary access to the TopLink runtime. It enables applications to perform persistence operations with the data source that contains persistent objects. A session associates data source platform information, data source login information, and mapping metadata for a particular application. You can reuse mapping metadata in different applications by defining different sessions.

TopLink provides different session types, each optimized for different design requirements and data access strategies. You can combine different session types in the same application.

The TopLink Editor provides the following TopLink sessions:

- **Server and Client Sessions** – Server sessions provide session management to a single data source (including shared object cache and connection pools) for multiple clients in a three-tier architecture using database or EIS platforms. This is the most flexible, scalable, and commonly used session. You acquire a client session from a server session at run time to provide access to a single data source for each client.
- **Database Session** – A database session provides a client application with a single data source connection, for simple, standalone applications in which a single connection services all data source requests for one user.
- **Session Broker and Client Sessions** – A session broker provides session management to multiple data sources for multiple clients by aggregating two or more server sessions (can also be used with database sessions).

  You acquire a client session from a session broker at run-time to provide access to all the data sources managed by the session broker for each client.

Other session types are can be configured directly in Java code. For more information about session types, see the *Oracle Fusion Middleware Developer's Guide for Oracle TopLink*.

13.6.1 How to Create a New Sessions Configuration File

Each TopLink sessions configuration (sessions.xml file) can contain multiple sessions and session brokers. In addition, you can specify a classpath for each sessions configuration that applies to all the sessions it contains.

To create a new sessions configuration file:

1. Select **File** > **New**.
2. In the Categories list, select **Business Tier** > **TopLink/JPA**.
3. In the Items list, select **TopLink Sessions Configuration**.

4. Click **OK**.

   The Create TopLink Sessions Configuration dialog appears.

5. Complete each field on the dialog and click **OK**.

   In Application Navigator, JDeveloper adds the `sessions.xml` file in the folder where it was created and the default session to the sessions configuration node in the Structure view.

13.6.2 How to Create Sessions

A TopLink session provides the primary access to the TopLink runtime. It is the means by which your application performs all persistence operations with the data source that contains persistent objects.

A session associates data source platform information, data source login information, and mapping metadata for a particular application. You can reuse mapping metadata in different applications by defining different sessions.

To create a new TopLink session:

1. In the Application Navigator, right-click a TopLink sessions configuration file and select **Open**.

   The TopLink sessions configuration file appears in the TopLink Editor, showing the existing sessions and session brokers in this sessions configuration file.

2. Click **Create a New Session**.

3. Complete each field in the New Session dialog and click **OK**.

   JDeveloper adds the new session to the sessions configuration node in the Structure view.

13.6.3 Acquiring Sessions at Runtime

After you create and configure sessions, you can use the TopLink session manager to acquire a session instance at run time. The TopLink session manager enables developers to build a series of sessions that are maintained under a single entity. The session manager is a static utility class that loads TopLink sessions from the `sessions.xml` file, caches the sessions by name in memory, and provides a single access point for TopLink sessions.

The session manager has two main functions: it creates instances of the sessions and it ensures that only a single instance of each named session exists for any instance of a session manager.

The session manager instantiates sessions as follows:

- The client application requests a session by name.
- The session manager looks up the session name in the `sessions.xml` file. If the session name exists, the session manager instantiates the specified session; otherwise, it raises an exception.
- After instantiation, the session remains viable until you shut down the application.

Once you have a session instance, you can use it to acquire additional types of sessions for special tasks. This is particularly useful for EJB applications in that an enterprise bean can acquire the session manager and acquire the desired session from it.

13.6.4 How to Create Session Brokers

The session *broker* is a mechanism that enables client applications to transparently access multiple databases through a single TopLink session. A session broker may contain both server sessions and database sessions. Oracle recommends that you use the session broker with server sessions because server sessions are the most scalable session type.

After you create and configure a session broker with server sessions, you can acquire a client session from the session broker at run time to provide a dedicated connection to all the data sources managed by the session broker for each client.

To create a new session broker:

1. In the Application Navigator, open the sessions configuration file (`sessions.xml`).

   The sessions configuration displays in the TopLink Editor.

2. Click **Create a New Session Broker**.

3. Complete each field in the dialog, select the sessions to add to the session broker, and then click **OK**.

13.6.5 How to Create Data Source Logins

The TopLink sessions configuration file (sessions.xml) overrides any login information that you specified in the TopLink map. You can create data source logins for relational database or EIS data sources.

To create a data source:

1. Select the TopLink sessions configuration (`sessions.xml`) in the Application navigator.

2. Expand the sessions node in the `sessions.xml` Structure view and then select the TopLink session.

   The TopLink session information appears in the TopLink Editor.

3. Select the **Login** tab.

4. Complete the Connection information.

13.6.6 How to Create Connection Pools

A *connection pool* is a service that creates and maintains a shared collection (pool) of data source connections on behalf of one or more clients. The connection pool provides a connection to a process on request, and returns the connection to the pool when the process is finished using it. When it is returned to the pool, the connection is available for other processes.

Because establishing a connection to a data source can be time-consuming, reusing such connections in a connection pool can improve performance. TopLink uses connection pools to manage and share the connections used by server and client sessions. Reusing connections to a single data source reduces the number of connections required and allows your application to support many clients.

To create a new connection pool.

1. Select the TopLink sessions configuration (`sessions.xml`) in the Application navigator.

2. Expand the **sessions** node in the `sessions.xml` Structure window and then select the TopLink session.
3. Right-click the session and select **New** > **Named Connection Pool** from the context menu.
4. Enter a name for the connection pool and click **OK**.

   JDeveloper adds the connection pool to the Structure window.
5. Select the newly created connection pool.

   Its properties appear in the Connection Pool page in the Editor window.

13.7 Developing TopLink Applications

Oracle TopLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality.

13.7.1 Using TopLink the Cache

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values.

TopLink uses the cache to:

- Improve performance by holding recently read or written objects and accessing them in-memory to minimize database access.
- Manage locking and isolation level.
- Manage object identity.

TopLink uses two types of cache:

- **Session Cache** – A shared cache that services clients attached to a given session. When a client session reads objects from, or writes them to, a data source, TopLink saves a copy of the objects in the parent server session's cache and makes them accessible to all other processes in the session.

  TopLink adds objects to the session cache from the following:

  – The data store, when TopLink executes a read operation.

  – The unit of work cache, when a unit of work successfully commits a transaction.
- **Unit of Work Cache** – Services operations within the unit of work. It maintains and isolates objects from the session cache, and writes changed or new objects to the session cache after the unit of work commits changes to the data source. TopLink updates the sessions cache when a unit of work commits to the data source.

13.7.1.1 Object Identity

TopLink preserves object identity through its cache using the primary key attributes of a persistent entity, which may or may not be assigned through sequencing. Oracle recommends that you always maintain object identity. Disable object identity only if absolutely necessary, for example, for read-only objects.

13.7.1.2 Querying and the Cache

A query that is run against the shared session cache is known as an *in-memory query*.

By default, a query that looks for a single object based on primary key attempts to retrieve the required object from the cache first, searches the data source only if the object is not in the cache. All other query types search the database first, by default. You can specify whether a given query runs against the in-memory cache, the database, or both.

13.7.1.3 Handling Stale Data

Stale data is an artifact of caching, in which an object in the cache is not the most recent version committed to the data source.

13.7.1.4 Explicit Query Refreshes

For systems that require several objects be current, you can specify that these objects be explicitly refreshed from the database without incurring the full cost of distributed cache coordination. To do this:

1. Configure a set of queries that refresh the required objects.
2. Establish an appropriate refresh policy.
3. Invoke the queries as required to refresh the objects.

13.7.1.5 Cache Invalidation

Use a cache invalidation policy to specify how or when a cached object becomes invalid. Using cache invalidation ensures that an application does not use stale data. You can configure the cache to invalidate objects at a certain time of day, mark an object as invalid after a specified time period after the object was read, or you can set the set the invalidation policy to invalidate an object only explicitly. You can set an invalidation policy to apply to all objects by configuring it at the project level, to certain objects by applying it at the descriptor level, or to the results returned by a query by applying it at the query level.

13.7.1.6 Cache Coordination

Cache coordination enhances performance by avoiding data source access. By enabling the instances of a session to broadcast object changes to one another so that each session's cache is kept current or notified that the cache must update an object from the data source the next time that it is read, it also reduces stale data. In addition, cache coordination reduces the optimistic lock exceptions in distributed environments as well as the number of failed or repeated transactions in an application. Use cache coordination for applications that are read-based, regularly request and update the same objects, and have changes performed by a single Java application with multiple, distributed sessions.

As an alternative to cache coordination, you can tune the TopLink cache for each read-only, read-mostly, and write-mostly classes using identity type, cache invalidation, or cache isolation. You can perform this tuning before cache coordination.

13.7.1.7 Cache Isolation

Isolated client sessions provide a mechanism for disabling the shared server session cache. Any classes marked as isolated only cache objects relative to the life cycle of their client session. These classes never utilize the shared server session cache. This is the best mechanism to prevent caching as it is configured on a per-class basis allowing caching for some classes, and denying it for others.

13.7.1.8 Cache Locking and Transaction Isolation

By default, TopLink optimizes concurrency to minimize cache locking during read or write operations. Use the default TopLink transaction isolation configuration unless you have a very specific reason to change it.

13.7.2 How to Configure the TopLink Cache

In JDeveloper, you can configure the TopLink cache for a specific TopLink map. The cache options will apply globally to all descriptors. You can override the map-level cache configuration by defining cache configuration at the descriptor level.

To configure the TopLink cache at the TopLink map-level:

1. Select the TopLink map in the Application Navigator.
2. In the Structure window, select the TopLink map.
3. Complete the Caching files on the **Defaults** tab.

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values.

To configure the TopLink cache at the descriptor-level:

1. Select the TopLink map in the Application Navigator.
2. In the Structure window, select the TopLink map.
3. Complete the Caching files on the **Caching** tab.

13.7.3 Using Queries

TopLink enables you to create, read, update, and delete persistent objects or data using queries in both Java EE and non-Java EE applications for both relational and nonrelational data sources. For more information about queries, see the *Oracle Fusion Middleware Developer's Guide for Oracle TopLink*.

Querying a data source means performing an action on, or interacting with, the contents of the data source. To do this, perform the following:

- Define an action in a syntax native to the data source being queried.
- Apply the action in a controlled fashion.
- Manage the results returned by the action (if any).

For TopLink, you must also consider how the query affects the TopLink cache.

13.7.3.1 TopLink Query Languages

TopLink enables you to express a query using any of the following query languages:

- SQL Queries
- EJBQL Queries
- JPQL Queries
- XML Queries
- EIS Interactions
- Query-by-Example
- TopLink Expressions

13.7.3.2 TopLink Query Types

- **Named Queries** – An instance of `DatabaseQuery` stored by name in a Session or a descriptor's `DescriptorQueryManager` where it is constructed and prepared once. Such a query can then be repeatedly executed by name.
- **Call Queries** – An instance of Call that you create and then either execute directly, using a special Session API to perform limited data source actions on data only, or execute indirectly in the context of a `DatabaseQuery`. TopLink supports Call instances for custom SQL, stored procedures, and EIS interactions.
- **Descriptor Query Manager** – The `DescriptorQueryManager defines` a default `DatabaseQuery` for each basic data source operation (create, read, update, and delete), and provides an API with which you can customize either the `DatabaseQuery` or its Call.
- **EJB 2.n CMP Finders** – A query defined on the home interface of an enterprise bean that returns enterprise beans. You can implement finders using any TopLink query type, including `JPAQLCall` and `EJBQLCall`, a call that takes JPA/EJB QL.

In most cases, you can compose a query directly in a given query language or, preferably, you can construct a DatabaseQuery with an appropriate Call and specify selection criteria using a TopLink Expression. Although composing a query directly in SQL appears to be the simplest approach (and for simple operations or operations on unmapped data, it is), using the DatabaseQuery approach offers the compelling advantage of confining your query to your domain object model and avoiding dependence on data source schema implementation details.

13.7.4 How to Create Queries

Some queries are implicitly constructed for you based on passed in arguments and executed in one step (for example, session queries) and others you create explicitly, configure, and then execute, such as database queries.

To create a query:

1. Select the TopLink map in the Application Navigator.
2. In the Structure window, select the descriptor.
3. On the **Queries** tab, create your desired query.

   The Queries tab allows you to create and manage queries associated with a TopLink descriptor. You can create a named query in the Named Queries section, or create a custom query in the Custom Calls section.

13.7.5 Using Basic Query API

The TopLink basic query API includes support for the following, most commonly used queries:

- Session Queries
- DatabaseQuery Queries
- Named Queries
- SQL Calls
- EJBQL Calls
- EIS Interactions
- Collection Query Results

- Report Query Results

13.7.6 Using Advanced Query API

The TopLink query API also allows the use of the following, more advanced query API calls and techniques:

- Redirect Queries
- Historical Queries
- Fetch Groups
- Read-Only Queries
- Interfaces
- Inheritance Hierarchy
- Additional Join Expressions
- EJB Finders
- Cursor and Stream Query Results

For more information about advanced query API, see the *Oracle Fusion Middleware Developer's Guide for Oracle TopLink*.

13.7.6.1 Redirect Queries

A redirect query is a named query that delegates query execution control to your application. redirect queried allow you to define the query implementation in code as a static method. To perform complex operations, you can combine query redirectors with the TopLink query framework.

13.7.6.2 Historical Queries

To make a query time-aware, you specify an `AsOfClause` that TopLink appends to the query. Use the `AsOfClause` class if your historical schema is based on time stamps or the `AsOfSCNClause` class if your historical schema is based on database system change numbers. You can specify an `AsOfClause` at the time you acquire a historical session so that TopLink appends the same clause to all queries, or you can specify an `AsOfClause` on a query-by-query basis.

13.7.6.3 Fetch Groups

You can use a fetch group with a ReadObjectQuery or ReadAllQuery. When you execute the query, TopLink retrieves only the attributes in the fetch group. TopLink automatically executes a query to fetch all the attributes excluded from this subset when and if you call a getter method on any one of the excluded attributes.

13.7.6.4 Read-Only Queries

In cases where you know that data is read-only, you can improve performance by specifying a query as read-only: this tells TopLink that any object returned by the query is immutable.

You can configure an object-level read query as read-only. When you execute such a query in the context of a `UnitOfWork`, TopLink returns a read-only, non-registered object. You can improve performance by querying read-only data in this way because the read-only objects need not be registered or checked for changes.

13.7.6.5 Interfaces

When you define descriptors for an interface to enable querying, TopLink supports querying on an interface, as follows:

- If there is only a single implementor of the interface, the query returns an instance of the concrete class.
- If there are multiple implementors of the interfaces, the query returns instances of all implementing classes.

13.7.6.6 Inheritance Hierarchy

When you query on a class that is part of an inheritance hierarchy, the session checks the descriptor to determine the type of the class, as follows:

- If you configure the descriptor to read subclasses (the default configuration), the query returns instances of the class and its subclasses.
- If you configure the descriptor not to read subclasses, the query returns only instances of the queried class, but no instances of the subclasses.
- If you configure the descriptor to outer-join subclasses, the query returns instances of the class and its subclasses.
- If you configure the descriptor to outer-join subclasses, the query returns instances of the class and its subclasses.

13.7.6.7 Additional Join Expressions

You can set the query manager to automatically append an expression to every query it performs on a class. For example, you can add an expression that filters the database for the valid instances of a given class. Use this to do the following:

- Filter logically deleted objects
- Enable two independent classes to share a single table without inheritance
- Filter historical versions of objects

13.7.6.8 EJB Finders

To create a finder for an entity bean that uses the TopLink query framework, you must define, declare, and configure it. For predefined finders, you do not need to explicitly create a finder. For default finders, you only need to define the finder method.

13.7.6.9 Cursor and Stream Query Results

Cursors and streams are related mechanisms that let you work with large result sets efficiently. A stream is a view of a collection, which can be a file, a device, or a Vector. A stream provides access to the collection, one element at a time in sequence. This makes it possible to implement stream classes in which the stream does not contain all the objects of a collection at the same time.

Large result sets can be resource-intensive to collect and process. To improve performance and give the client more control over the returned results, configure TopLink queries to use a cursor or stream. Cursors & streams are supported by all subclasses of `DataReadQuery` and `ReadAllQuery`.

13.7.7 How to Create TopLink Expressions

TopLink expressions let you specify query search criteria based on your domain object model. When you execute the query, TopLink translates these search criteria into the appropriate query language for your platform.

TopLink provides the following two public classes to support expressions:

- The `Expression` class represents an expression that can be anything from a simple constant to a complex clause with boolean logic. You can manipulate, group, and integrate expressions.
- The `ExpressionBuilder` class is the factory for constructing new expressions. You can specify a selection criterion as an `Expression` with `DatabaseQuery` method `setSelectionCriteria` and in a finder that takes an `Expression`.

A simple expression usually consists of the following parts:

- The *attribute*, which represents a mapped attribute or query key of the persistent class.
- The *operator*, which is an expression method that implements boolean logic, such as `GreaterThan`, `Equal`, or `Like`.
- The *constant* or comparison, which refers to the value used to select the object.

To create basic expressions for use in named queries:

1. Select the TopLink map in the Application Navigator.
2. In the Structure window, select the descriptor.
3. Select the named query and in the **Selection Criteria** area, edit the expression.

13.7.8 Understanding TopLink Transactions

A database transaction is a set of operations (create, update, or delete) that either succeed or fail as a single operation. The database discards, or rolls back, unsuccessful transactions, leaving the database in its original state. Transactions may be internal (that is, provided by TopLink) or external (provided by a source external to the application, such as an application server).

In TopLink, transactions are contained in the unit of work object. You acquire a unit of work from a session and using its API, you can control transactions directly or through a Java 2 Enterprise Edition (Java EE) application server transaction controller such as the Java Transaction API (JTA).

As a transaction is committed, the database maintains a log of all changes to the data. If all operations in the transaction succeed, the database allows the changes; if any part of the transaction fails, the database uses the log to roll back the changes.

Transactions execute in their own context, or logical space, isolated from other transactions and database operations. The transaction context is demarcated; that is, it has a defined structure that includes the following:

- A begin point, where the operations within the transaction begin. At this point, the transaction begins to execute its operations.
- A commit point, where the operations are complete and the transaction attempts to formalize changes on the database.

The degree to which concurrent (parallel) transactions on the same data are allowed to interact is determined by the level of transaction isolation configured. ANSI/SQL

defines four levels of database transaction isolation. Each offers a trade-off between performance and resistance from the following unwanted actions:

- Dirty read: a transaction reads uncommitted data written by a concurrent transaction.
- Nonrepeatable read: a transaction rereads data and finds it has been modified by some other transaction that was committed after the initial read operation.
- Nonrepeatable read: a transaction rereads data and finds it has been modified by some other transaction that was committed after the initial read operation.

13.7.9 TopLink Transactions and the Unit of Work

The unit of work isolates changes in a transaction from other threads until it successfully commits the changes to the database. Unlike other transaction mechanisms, the unit of work automatically manages changes to the objects in the transaction, the order of the changes, and changes that might invalidate other TopLink caches. The unit of work manages these issues by calculating a minimal change set, ordering the database calls to comply with referential integrity rules and deadlock avoidance, and merging changed objects into the shared cache. In a clustered environment, the unit of work also synchronizes changes with the other servers in the coordinated cache.

Like any transaction, a unit of work transaction provides the following:

- **Unit of Work Transaction Context** – Unit of work operations occur within a unit of work context, in which writes are isolated from the database until commit time. The unit of work executes changes on copies, or clones, of objects in its own internal cache, and if successful, applies changes to objects in the database and the session cache.
- **Unit of Work Transaction Demarcation** – In a TopLink application, your application demarcates transactions using the unit of work. If your application includes a Java EE container that provides container-managed transactions, your application server demarcates transactions using its own transaction service. You can configure TopLink to integrate with the container's transaction service by specifying a TopLink external transaction controller.
- **Unit of Work Transaction Isolation** – The unit of work does not directly participate in database transaction isolation. Because the unit of work may execute queries outside the database transaction, the database does not have control over this data and its visibility. However, by default, TopLink provides a degree of transaction isolation regardless of database transaction isolation configured on the underlying database. Each unit of work instance operates on its own copy (clone) of registered objects. In this case, because the unit of work provides an API that allows querying to be done on object changes within a unit of work, the unit of work provides read committed operations. Changes are committed to the database only when the unit of work commit method is called.

14

Developing Secure Applications

This chapter describes how you can develop, deploy, and administer secure Java EE applications in Oracle JDeveloper.

This chapter includes the following sections:

14.1 About Developing Secure Applications

The Fusion Middleware Suite lets you develop, deploy, and administer secure applications. You can secure Java EE applications using only container-managed security or, for Fusion web applications, you can use Oracle ADF Security. Fusion web applications are Java EE applications that you develop using the Oracle Application Development Framework (Oracle ADF).

14.1.1 Understanding Java EE Applications and Oracle Platform Security Services for Java (OPSS)

A Java EE application can be enhanced to use OPSS. In this scenario, you work with JDeveloper's declarative editors to configure users and roles. You secure application resources using Java EE container-managed security.

14.1.2 Understanding Fusion Web Applications and ADF Security

This scenario is a fully declarative implementation that adds ADF Security to enable fine-grained security policies for Oracle ADF resources. You work with JDeveloper's declarative editors to configure a file-based identity store, policy store, and credential store; and, because your application utilizes Oracle ADF, you also run a wizard to configure security for web pages associated with ADF resources (such as ADF task flows and ADF page definitions) and then use the `jazn-data.xml` policy editor to define security policies.

14.1.3 Understanding Container-managed Security

The Java EE security model is a role-based, declarative model based on container-managed security, where resources are protected by roles that are assigned to users. This model allows decoupling an application from its underlying security infrastructure since security can be specified separately from the application logic in an application deployment descriptor. The container, where an application runs, provides security for the application according to a specifications in the deployment descriptor. This model also allows embedding security data (annotations) in the application code that can be referenced in deployment descriptors.

For more information about container-managed security, see the *Oracle Fusion Middleware Security Guide*.

14.1.4 Additional Functionality

The Oracle ADF Security framework is the preferred technology to provide authentication and authorization services to the Fusion web application. A prime reason is that Oracle ADF Security is built on top of the Oracle Platform Security Services (OPSS) architecture, which provides a critical security framework and is itself well-integrated with Oracle WebLogic Server.

For more information about Oracle ADF security, see the "Enabling ADF Security in a Fusion Web Application" chapter of the *Oracle Fusion Middleware Developer's Guide for ADF*.

For more information on OPSS, see the *Oracle Fusion Middleware Application Security Guide*.

14.2 Securing Applications in Phases

When developing secure applications in JDeveloper it is often useful to think of development and deployment (to the production environment) as different phases, each with different needs. This is because during development and testing, JDeveloper supports easy to manage file-based security through integration with Oracle Platform Security Services (OPSS).

JDeveloper simplifies the application development life-cycle for security, and allows you to store the data in a flat file, for easy development. The `jazn-data.xml` file is JDeveloper's default file-based security provider for integration with OPSS. The `jazn-data.xml` file stores the users, groups, roles, and policies that you define the Fusion web application built using the Oracle Application Development Framework (Oracle ADF) and Oracle ADF Security. JDeveloper provides a dedicated editor for this file that simplifies creating the security data stores.

A feature of OPSS is the abstraction of users defined by the production environment's enterprise roles into application roles that are specific to the functions of your application. During development the application developer adds application roles and security policies that use application roles to the policy store of the `jazn-data.xml` file. Then, to simplify testing, the developer may add a few users to the identity store and directly assign these test users to application roles. Therefore, for testing the application, the `jazn-data.xml` can also be used as the identity store.

During development, your application does not ned to be aware of the enterprise roles defined in the production environment. After deployment an administrator will use Oracle Enterprise Manager Fusion Middleware Control to map the production-level enterprise roles to the application roles of your application's policy store. This

mapping will allow a user who is a member of a given enterprise role to have access to the resources that are accessible from the associated application role.

After you complete the application, you migrate the policy store to the production environment provider on Oracle WebLogic Server. At that point, you will replace your test user identity store with enterprise users configured in the Oracle WebLogic Server embedded LDAP server. In contrast to the `jazn-data.xml` file, the LDAP server supports a distributed application server configuration that may be employed in a production environment. For details about the LDAP server, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

Therefore, working with the file-based provider and OPSS in JDeveloper helps separate the demands of the production environment through:

- Declaratively defining test users and application roles
- Declaratively defining security policies for Oracle ADF resources
- Easily migrating from application-level security provider to `system-jazn-data.xml` security provider during deployment
- Delaying the mapping of enterprise roles until deployment

14.3 About Web Application Security and JDeveloper Support

Java EE declarative security in Oracle WebLogic Server is implemented with Oracle Platform Security Services (OPSS), Oracle's implementation of the JAAS standard. OPSS extends Java EE security to provide application developers, system integrators, security administrators, and independent software vendors with a portable, integrated, and comprehensive security platform framework for Java SE and Java EE applications.

To learn more about OPSS and its features, see *Oracle Fusion Middleware Security Guide*.

JDeveloper provides tools to support configuring Java EE security for web applications and for deploying secure web applications to an application server instance. A developer, while developing an application, can configure OPSS services from JDeveloper through wizards and editors.

JDeveloper provides specific editors to create and edit Oracle Platform Security configurations (`jps-config.xml`), JAAS configurations (`jazn-data.xml`), and Web application deployment descriptors (`web.xml`). JDeveloper also supports direct deployment of web applications to application servers. For more information, see Section 14.2, "Securing Applications in Phases."

When you develop web applications you may choose to use Oracle Application Development Framework (Oracle ADF) to work with data-aware components in the user interface. When your user interface contains ADF resources, such as ADF task flows and ADF page definitions, then you have the option to secure the web pages that rely on those resources through the ADF Security framework. JDeveloper tools support iterative development of security so you can easily create, test, and edit security policies that you create for ADF resources. You can proceed to create test users in JDeveloper and run the application in Integrated WebLogic Server to simulate how end users will access the secured resources. For more information, see. Section 14.5.2, "How to Secure ADF Resources Using ADF Security in Fusion Web Applications."

For more information on web application security, see *Oracle Fusion Middleware Programming Security for Oracle WebLogic Server*.

14.4 Handling User Authentication in Web Applications

Authentication in declarative security is enforced when a user requests a protected web application area.

14.4.1 About Authentication Type Choices

Authentication in declarative security is enforced when a user requests a protected web application area. If the user has not been authenticated before, the container will retrieve credentials from the user. Users stay authenticated throughout the server session.

The supported types of authentication are: FORM based authentication, BASIC authentication, and CLIENT-CERT authentication. The type of authentication is specified in the web.xml deployment descriptor using the `<login-config>` element.

14.4.1.1 BASIC authentication

BASIC authentication uses the browser login dialog for the user to enter his user name and password. This dialog form cannot be customized and thus varies in its look and feel depending on the type of browser used. The user credentials are stored in the browser session for the authenticated realm. A realm is a repository that contains a set of permissions for the authenticated user. The default realm in Oracle Platform Security Services is `jazn.com`.

The code snippet in Example 14–1 demonstrates how BASIC authentication is specified in the `web.xml` file:

Example 14–1 BASIC Authentication Specified in web.xml File

```
<login-config>
<auth-method>BASIC</auth-method>
<realm-name>jazn.com</realm-name>
</login-config>
```

14.4.1.2 FORM authentication

FORM based authentication allows the application developer to specify a custom login dialog. The username parameter must have a name of `j_username`, the password field must be named j_password. The login form action must have a value of `j_security_check` for the Java EE container to authenticate the request.

The code snippet in Example 14–2 demonstrates how FORM authentication is specified in the `web.xml` file:

Example 14–2 FORM Authentication Specified in web.xml File

```
<login-config>
       <auth-method>FORM</auth-method>
       <form-login-config>
             <form-login-page>loginform.jsp</form-login-page>
             <form-error-page>error.jsp</form-error-page>
       </form-login-config>
</login-config>
```

14.4.1.3 CLIENT-CERT authentication

CLIENT-CERT authentication uses the X.509 certificate to authenticate users. This type of authentication is also known as public key encryption.

For more information about authentication type choices, see the *Oracle Fusion Middleware Security Guide*.

For more information about authentication type using Oracle WebLogic Server, see *Oracle Fusion Middleware Programming Security for Oracle WebLogic Server*.

14.4.2 Encrypting Passwords for a Target Domain

As password encryption is specific to a WebLogic Server domain, you must manually add the password handling to the `weblogic-jdbc.xml` file. To encrypt a password, use the encrypt utility (`weblogic.security.Encrypt`) for the domain to which you want to deploy.

> **Note:** Passwords are domain-specific, so each time you want to deploy to a different domain you must re-encrypt the password for the target domain

The XML code you need to add to the weblogic-jdbc.xml should look something like this:

```
<password-encrypted>toystore</password-encrypted>
```

You can either put the clear text password or the encrypted password string in between the tags. This element goes inside of the `<jdbc-driver-params>` element, which will already be present in the `weblogic-jdbc.xml` if it has been edited using the Overview Editor.

14.4.2.1 weblogic.security.Encrypt

The `weblogic.security.Encrypt` utility encrypts cleartext strings for use with WebLogic Server. The utility uses the encryption service of the current directory, or the encryption service for a specified WebLogic Server domain root directory.

> **Note:** An encrypted string must have been encrypted by the encryption service in the WebLogic Server domain where it will be used. If not, the server will not be able to decrypt the string.

You can only run the `weblogic.security.Encrypt` utility on a machine that has at least one server instance in a WebLogic Server domain; it cannot be run from a client. Table 14–1 defines the arguments for the `weblogic.security.Encrypt` utility.

> **Note:** It is recommended that you run the utility from the Administration Server domain directory or on the machine hosting the Administration Server and specifying a domain root directory.

Syntax

```
java [ -Dweblogic.RootDirectory= dirname ]
[ -Dweblogic.management.allowPasswordEcho=true ]
weblogic.security.Encrypt [ password ]
```

Table 14–1 Arguments for the weblogic.security.Encrypt utility

Argument	Definition
`weblogic.RootDirectory`	Optional. WebLogic Server domain directory in which the encrypted string will be used. If not specified, the default domain root directory is the current directory (the directory in which the utility is being run).
`weblogic.management.allowPasswordEcho`	Optional. Allows echoing characters entered on the command line. weblogic.security.Encrypt expects that no-echo is available; if no-echo is not available, set this property to true.
`password`	Optional. Cleartext string to be encrypted. If omitted from the command line, you will be prompted to enter a password.

Examples

The utility returns an encrypted string using the encryption service of the domain located in the current directory:

```
java weblogic.security.Encrypt xxxxxx {3DES}Rd39isn4LLuF884Ns
```

The utility returns an encrypted string using the encryption service of the specified domain location:

```
java -Dweblogic.RootDirectory=./mydomain weblogic.security.Encrypt xxxxxx
{3DES}hsikci118SKFnnw
```

The utility returns an encrypted string in the current directory, without echoing the password:

```
java weblogic.security.Encrypt Password: {3DES}12hsIIn56KKKs3
```

14.4.3 How to Create an Identity Store

An identity store is a data store of users, enterprise roles (user groups), and login credentials. The credentials are verified during authentication and used to authorize the user's access to application functions.

Understanding Users, Roles, and Realms

A user is an end user accessing a service; it could be an individual or a software component. A enterprise role is a collection of users that you group with the purpose of conferring the same set of permissions. A realm is a collection of authenticated users and enterprise roles.

For more information about users, enterprise roles, and realms, see the *Oracle Fusion Middleware Security Guide*.

Understanding Identity Stores in JDeveloper

When you develop secure applications in JDeveloper, you work with a file-based data store to define the users you wish to allow to log on. The advantage of defining a file-based identity store through the `jazn-data.xml` file is that it supports easy testing yet remains compatible with deployment to your production environment through migration to the `system-jazn-data.xml` file. It also avoids the complexity

of setting up and maintaining an Oracle Internet Directory service for the LDAP-based identity store.

When you create a Fusion web application with Oracle ADF, the identity store will be created automatically when you run the Configure ADF Security wizard.

> **Note:** The LDAP-based identity store is a design time feature in JDeveloper, and is not available at runtime. JDeveloper's Integrated WebLogic Server overrides any LDAP identity store configuration.

For more information about identity stores, see the *Oracle Fusion Middleware Security Guide*.

To create an identity store:

1. Double-click the jps-config.xml file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Identity Store** tab in the `jps-config.xml` Overview Editor.
3. Click the **Add a New Identity Store** icon at the top of the page. The Create Identity Store dialog opens.
4. Choose the desired type of identity store option:
   - To create a file based identity store, choose **XML-Based Identity Store**, and enter the name for the store. By default, the file name is `idstore.xml`.
   - To create an LDAP based identity store, choose **LDAP-Based Identity Store**, and enter the name for the store. By default, the file name is `idstore.oid`.

     Note: The LDAP-based identity store is a design time feature in JDeveloper, and is not available at runtime. The Integrated WebLogic Server in JDeveloper overrides any LDAP identity store configuration.
5. When you are done, click **OK** to close the dialog.

14.4.4 How to Add Test Users to the Identity Store

The identity store is an XML file that stores users and enterprise roles, and is used while authenticating users. There can be an identity store at either the domain or application level.

To add users to the identity store:

1. Open the application in the Application Navigator.
2. Choose **Application > Secure >Users** to open the overview editor for the `jazn-data.xml` file.
3. On the **Users** page, click the **New User** icon.
4. Enter the new user name and password.
5. Select the user from the **Users** list and enter further details, such as display name and description.
6. Save your changes to the `jazn-data.xml` file.

14.4.5 How to Add Enterprise Roles to the Identity Store

An enterprise role is a set of users that you group with the intention of conferring the same permission grants. You add enterprise roles to the identity store. You add application roles to the policy store.

> **Note:** Before adding a user to an enterprise role, ensure that you have created users in the identity store

To add roles to the identity store:

1. Open your application in the Application Navigator.
2. Choose **Application > Secure > Groups** to open the Enterprise Roles page of the overview editor for the `jazn-data.xml` file.
3. Under **Enterprise Roles**, click the **New Role** icon. The new role appears in the **Enterprise Roles** list.
4. Select the role from the **Enterprise Roles** list and enter further details, such as display name and description.

To manage users assigned to enterprise roles:

1. Open the **Enterprise Roles** page of the overview editor for `jazn-data.xml` file.
2. Select the role from the **Enterprise Roles** list, and then click the **Members** tab.
3. In the **Members** section, add or remove other members or roles.

To view assigned enterprise roles:

1. Open the **Enterprise Roles** page of the overview editor for the `jazn-data.xml` file.
2. Select the role from the **Roles** list, and then click the **Assigned Roles** tab.

14.4.6 How to Create a Credential Store

A credential store is a wallet-based file for storage of system credentials required by Oracle Platform Security Services (OPSS) in connecting to external systems such as databases. In JDeveloper, the credential store is the `cwallet.sso` file. The file contains all your OPSS-based credentials, and will be used in JDeveloper to store credentials that you define for Oracle ADF security. This file is normally not edited directly.

JDeveloper checks for the existence of a credential store service instance and creates the store the first time the you create a connection, for example, a database connection, in the Application Resources panel of the Application Navigator.

For more information about credential stores, see the *Oracle Fusion Middleware Security Guide*.

To create a credential store:

1. Double-click the `jps-config.xml` file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Credential Store** tab in the `jps-config.xml` Overview Editor.
3. Click the **Add the Credential Store** icon at the top of the page. The Create Credential Store dialog opens.

4. Enter the name of credential store file, and click **OK**.

> **Note:** You can create only one credential store in an application.

14.4.7 How to Add a Login Module

A login module is a component that authenticates users and populates a subject with principals. Login modules can be plugged in and used by applications without changing application code. An application can use more than one login module.

The login authentication process occurs in two distinct phases:

1. The login module attempts to authenticate a user requesting, as necessary, a name and a password or some other credential data; only if this phase succeeds, the second phase is invoked.
2. The login module assigns relevant principals to a subject, which is eventually used to perform some privileged action.

All login modules in a domain are configured in the file `jps-config.xml` using the following elements:

- `serviceProvider` — to define a service provider for the login module.
- `serviceInstance` — to define one or more instances of the service provider
- `jpsContext` — to specify which instances to use

In JDeveloper, you can choose a pre-defined login module for your application, or create a new custom login module. Table 14–2 contains the pre-defined login modules that are available in JDeveloper:

Table 14–2 Predefined Login Modules

Module	Description
`saml.loginmodule`	Used for SAML token assertion and implements the `oracle.security.jps.internal.jaas.module.saml.JpsSAMLLoginModule` class.
`krb5.loginmodule`	Used for Kerberos token assertion and implements `com.sun.security.auth.module.Krb5LoginModule` class.
wss.digest.loginmodule	Used to authenticate the digest based user name token based on WSS Digest specification and implements `oracle.security.jps.internal.jaas.module.digest.WSSDigestLoginModule`. This is supported only for JSE use cases
certificate.authenticator.loginmodule	Used to assert the X509 certificates and implements `oracle.security.jps.internal.jaas.module.x509.X509LoginModule` class.
`user.authentication.loginmodule`	Used to authenticate the user based on valid user name and password, and implements `oracle.security.jps.internal.jaas.module.authentication.JpsUserAuthenticationLoginModule` class
user.assertion.loginmodule	Used to authenticate the user based on valid user name and password, and implements `oracle.security.jps.internal.jaas.module.assertion.JpsUserAssertionLoginModule` class.

Table 14–2 (Cont.) Predefined Login Modules

Module	Description
`idstore.loginmodule`	Used to authenticate JSE bases use cases and implements `oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule` class

For more information about login modules, see the *Oracle Fusion Middleware Security Guide*.

To add a login module:

1. Double-click the `jps-config.xml` file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Login Modules** tab in the `jps-config.xml` Overview Editor.
3. Click the **Choose from a list of pre-defined Login Modules** icon at the top of the page. The Add Login Modules dialog appears.
4. Select the checkbox of login modules you want to add. You can add more than one login module in an application.
5. Click **OK** when you are done.

14.4.8 How to Authenticate Through a Custom Login Module

A key Oracle Platform Security component is the login service. Conceptually, the login service is an adapter that ties the JAAS login module SPI (`javax.security.auth.spi.LoginModule`) to the Oracle Platform Security for Java framework (OPSS).

The primary role of the login service is to enable JAAS login module implementations to be configured and used in OPSS.

To add a custom login module:

1. Double-click the jps-config.xml file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Login Modules** tab in the `jps-config.xml` Overview Editor.
3. Click the **Create New Login Module** button at the top of the page.
4. Enter the **Login Module Name** then click **OK**.
5. Enter the classname for the login module. To search for an existing classname available to the project, click the **Search** button.
6. Select the **Login Control Flag**. This can be: REQUISITE, REQUIRED, SUFFICIENT, or OPTIONAL.
7. Select the **Log Level**. This can be: FINE, FINER, FINEST, CONFIG, INFO, WARNING, SEVERE.
8. Click **Debug** to define whether the login module will output debug messages.
9. Select **Add All Roles** to define whether all directly or indirectly granted roles of the user are added to the subject after authentication using the login module.
10. Enter the names and values for any other properties required by the login modules.

14.4.9 How to Add a Key Store

A key store is a repository of private keys and digital certificates.

If you have keys and certificates and wish to use them for secure services in your application, JDeveloper allows you to import a Java Key Store, Oracle Wallet (from a `*.sso` or `*.p12` file), or PCKS12 file (from a `*.p12` file). You cannot create a key store in JDeveloper.

For more information about key stores and key store providers, see the *Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server* guide.

To add a key store:

1. Double-click the jps-config.xml file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Key Stores** tab in the `jps-config.xml` Overview Editor.
3. Click the **Add a Key Store** icon at the top of the page. The Add Key Store dialog appears.
4. Import the key store file and complete the required fields. You can import a Java Key Store (from a `*.jks` file), Oracle Wallet (from a `*.sso` or `*.p12` file), or PCKS12 (from a `*.p12` file) file as a key store.
5. Click **OK** when you are done.

14.4.10 How to Enable an Anonymous Provider

The anonymous provider is an alternative to public pages in that unauthenticated user access can have permissions assigned that are more fine grained than allowing access to the whole (public) page.

Enabling the anonymous provider creates an anonymous JpsContext, which contains the anonymous service instance and the anonymous login module. Anonymous credentials will be used at runtime when the application user has not been authenticated and the application allows some resources to be accessible without authentication.

For more information about the anonymous provider, see the *Oracle Fusion Middleware Security Guide*.

To enable an anonymous provider for a web application:

1. Double-click the jps-config.xml file in the **Descriptors > META-INF** folder in the **Application Resources** panel of the Application Navigator.
2. Select the **Anonymous Provider** tab in the `jps-config.xml` Overview Editor.
3. Select **Enable Anonymous Provider**.
4. Select the **Security Contexts** tab and ensure that anonymous is automatically chosen as the Anonymous Provider.

14.4.11 How to Add Credentials to Users in the Identity Store

Credentials contain the authentication password for a user. The credentials appear in obfuscated form by default. Before adding credentials in the identity store, the member users must first be defined for the identity store.

To add credentials to users in the identity store:

1. Open the application in the Application Navigator.
2. Choose **Application > Secure > Users** to open the **Users** page of the overview editor for `jazn-data.xml`.
3. Select a user in the **Users** list, and add credentials to the **Password** field.

14.4.12 How to Choose the Authentication Type for the Web Application

Authentication in declarative security is enforced when a user requests a protected web application area. If the user has not been authenticated before, the container will retrieve credentials from the user. Users stay authenticated throughout the server session.

The supported types of authentication are: FORM based authentication, BASIC authentication, and CLIENT-CERT authentication. The type of authentication is specified in the web.xml deployment descriptor using the `<login-config>` element.

For more information on authentication types, see Section 14.4.1, "About Authentication Type Choices".

To select the authentication type for the web application:

1. Double-click the `web.xml` for the application in the Application Navigator.
2. Click the **Security** tab of the `web.xml` Overview Editor.
3. Expand the **Login Authentication** section and select the desired authentication type.

14.5 Securing Application Resources in Web Applications

Web pages and other resources of the web application should be secured. Depending on the type of application, you can secure your application in one of the two following ways:

- For a Java EE web application, use Oracle Platform Security Services (OPSS) to secure your web application.
- For an application developed using Oracle Application Development Framework (ADF), use Oracle ADF Security to secure your application.

Using OPSS Security

The following tasks outline the process of securing an application using Java EE security:

1. Specifying an authentication mechanism for users.
2. Managing users and groups in the realm.
3. Creating security roles for the application.
4. Mapping roles to users and groups.

Using Oracle ADF Security

You can use the Oracle ADF Security framework to provide authentication and authorization services to the Fusion web application.

For more information about Oracle ADF security, see the "Enabling ADF Security in a Fusion Web Application" chapter of the *Oracle Fusion Developer's Guide for ADF*.

14.5.1 How to Secure Application Resources Using the jazn-data.xml Overview Editor

JDeveloper enables you to secure your application resource types. The resource types can be known, that is, recognized by JDeveloper, or you can create your own resource type.

A resource type represents the type of a secured artifact, such as a flow, a job, or a web service, and, essentially, it is a template for creating resources of a particular type. All resources have an associated type and are filtered or grouped according to type.

To secure an application resource:

1. Open the application in the Application Navigator.
2. In the main menu, choose **Application > Secure > Resource Grants** to open the **Resource Grants** page in the overview editor for the `jazn-data.xml` file.
3. In the **Resource Type** dropdown list, select the resource type you want to secure, for example, **Task Flow**. The list will display all the resource types available in the selected projects. You can also create a new resource type.
4. Click the **Select Source Project** icon to select the source project. Instances of the selected resource type from the selected source projects will be displayed in the **Resources** list.
5. Add the grantees (application roles, enterprise roles, or code sources) that will be granted the resource permissions. You can grant resource permissions to users, application roles, enterprise roles, and code sources. Click the **Add Grantee** icon in the **Granted To** list to add grantees.
6. In the **Actions** list, select the actions that will be allowed on the resource.
7. Save your changes to the `jazn-data.xml` file.

14.5.2 How to Secure ADF Resources Using ADF Security in Fusion Web Applications

Security policies that you define in a Fusion web application support fine-grained access control for ADF security-aware resources, including ADF task flows and ADF page definitions. To enable ADF security policies, you begin by running the Configure ADF Security wizard on the user interface project.

After you enable ADF Security you must grant users access rights so that they may view the web pages of the Fusion web application. Access rights that you grant users are known as a security policy that you specify for the page's corresponding ADF security-aware resource. Ultimately, it is the security policy on the ADF resource that controls the user's ability to enter a task flow or view a web page:

- Do **not** define security policies for the individual web pages of a bounded task flow. When the user accesses the bounded task flow, security for all pages will be managed by the permissions you grant to the task flow. And, because the individual web pages (with associated page definitions) will be inaccessible by default, ADF Security prevents users from directly accessing the pages of the task flow. This supports a well-defined security model for task flows that enforces a single entry point for all users.
- Do define security policies for the individual web page only when the page is not a constituent of a bounded task flow. Page-level security is checked for pages that have an associated page definition binding file only if the page is directly accessed or if it is accessed in an unbounded task flow.

ADF security policies are maintained in the file-based jazn-data.xml policy store. Defining and updating ADF security policies in JDeveloper is supported by the

overview editor for this file. The resulting declarative ADF security policies are easy to read.

The detailed steps for securing Oracle ADF resources are in the "Enabling ADF Security in a Fusion Web Application" chapter of the *Oracle Fusion Developer's Guide for ADF*.

To define security policies for ADF resources:

1. Enforce ADF Security for the application by running the Configure ADF Security wizard.
2. Add application role names to the policy store.
3. Grant permission on the entire set of web pages contained in an ADF bounded task flows.
4. Grant permission on top-level web pages that are associated with an ADF page definition file and that are not associated with a bounded task flow.

   If your application contains top-level web pages that are not associated with an ADF resource because they do not contain data-aware components, you can optionally secure these pages too.
5. If necessary, grant permission on rows of data that are defined by an ADF entity object.
6. Provision the identity store by adding the users who will login to test security.
7. Associate the test users you created with one or more application roles.

14.6 Configuring an Application-Level Policy Store

Security policies for web application resources are stored in the application-level policy store.

14.6.1 About Policy Stores

A Policy Store is the repository of application and enterprise policies. A policy specifies the permissions granted to code running from a specific location.

An Application Policy Store is a repository of application policies together with application roles, application policies, principals, and permissions. Application roles can include application users and roles, and roles specific to the application (such as administrative roles). A policy can use any of these roles or users as principals. Similarly, a System Policy store is a repository of system policies, principals, and permissions. A system policy store does not contain roles.

When you create a Fusion web application with Oracle ADF, the policy store will be created automatically when you run the Configure ADF Security wizard.

The difference between an application policy store and a system policy store is in their scope. An application policy store is constrained within an application limiting it's accessibility, where as a system policy store can be accessed openly.

For more information on policy stores, see the *Oracle Fusion Middleware Security Guide*.

A Principal is an identity assigned to an entity; the entity could be a user or a role. A Permission is a set of operations allowed for a group of entities; the entity could be a principal too. A Grant, or a custom policy, includes permissions and principals. In JDeveloper, you cannot create a principal or a permission without creating a grant.

14.6.2 About Principals, Permissions and Grants

A Principal is an identity assigned to an entity; the entity could be a user or a role. A Permission is a set of operations allowed for a group of entities; the entity could be a principal too. A Grant, or a custom policy, includes permissions and principals. In JDeveloper, you cannot create a principal or a permission without creating a grant.

14.6.3 How to Add Application Roles to an Application Policy Store

Application roles are specific to an application and defined in the application policy store. They are used by the application directly (either a Java SE or Java EE application) and are not necessarily known to the Java EE container. In the file-based policy store in a `jazn-data.xml` file, these application roles are defined in `<app-role>` elements under `<policy-store>`, and then written to `system-jazn-data.xml` at the domain level during deployment.

To add application roles to the application policy store:

1. Open the application in the Application Navigator.
2. Choose **Application > Secure > Application Roles** to open the Application Roles page of the overview editor for the `jazn-data.xml` file.
3. Click the **Add** icon to create a new application role as a peer or child of the currently selected role, or to create a new role category. The new application role or category is listed in the **Roles** list.
4. Enter details of the role or role category in the **Name**, **Display Name**, and **Description** fields.
5. Save your changes to the `jazn-data.xml` file.

For more information, see the *Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server* guide.

14.6.4 How to Add Member Users or Enterprise Roles to an Application Role

Deployment users and roles are defined in the security provider that you use. For the file-based provider, deployment users and roles are defined in the `jazn-data.xml` file.

> **Note:** Before adding member users or member roles to an application role, the member users and member roles must first be defined for the identity store.

To add users or enterprise roles to an application role:

1. Open the application in the Application Navigator.
2. Choose **Application > Secure > Application Roles** to open the Application Roles page in the overview editor for the `jazn-data.xml` file.
3. From the **Application Roles** list, select the application role, and then click the **Members** tab.
4. To add a user, under **Member Users and Roles**, click the **Add User or Role** icon, and select **Add User**.
5. To add an enterprise role, under **Member Users and Roles**, click the **Add User or Role** icon, and select **Add Enterprise Role**.

6. Save your changes to the `jazn-data.xml` file.

14.6.5 How to Create Custom Resource Types

You can create custom resource types and specify them in the `jazn-data.xml` file.

A resource type represents the type of a secured artifact, such as a flow, a job, or a web service, and, essentially, it is a template for creating resources of a particular type. All resources have an associated type and are filtered or grouped according to type.

To create a custom resource type:

1. Open your application in the Application Navigator.
2. Choose **Application > Secure > Resource Grants** to open the Resource Grants page of the overview editor for the `jazn-data.xml` file.
3. In the Resource Grants page, click the **New Resource Type** icon next to the **Resource Type** field.
4. In the Create Resource Type dialog, specify the properties of the resource, such as name, display name, and associated actions. The **Actions** list in the Create Resource Type dialog is used to populate the checkable items list in the Resource Grants page for resources of this type.
5. Save the `jazn-data.xml` file.

14.6.6 How to Add Resource Grants to the Application Policy Store

You can add application resource grants to an application policy store by updating the Resource Grants page of the overview editor for `jazn-data.xml`.

A resource is an instance of a resource type that represents a concrete resource; it defines an application resource that can be secured by a policy, such as software components managed by a container (for example, URLs, EJBs, JSPs) or an application business (for example, Reports, Transactions, Revenue Charts).

To add a resource grant for the application policy store:

1. Open your application in the Application Navigator.
2. Choose **Application > Secure > Resource Grants** to open the Resource Grants page of the overview editor for the `jazn-data.xml` file.
3. To define the security policy, select an item in the **Security Policy** field. The application security policy is selected by default. To define global resource grants, select **Global**.
4. Select the resource type from the **Resource Type** dropdown menu, or click the **New Resource Type** icon to create one.
5. For the resource types that are filtered by project, the **Source Project** selector is enabled. You may need to change the source project selection to find the desired resources.
6. The resources that belong to the selected resource type are listed in the **Resources** list.
7. Manage the entities that the resource permissions have been granted to, by clicking the **Add Grantee** icon in the **Granted To** list. You can grant to an application role, a user, an enterprise role, or a code source.
8. View and select the actions allowed on the resource in the **Actions** list.

14.6.7 How to Add Entitlement Grants to the Application Policy Store

Using the Entitlement Grants page of the overview editor for `jazn-data.xml`, you can define a set of resource permissions and grant those permissions to multiple application roles without having to grant each permission to each application role individually.

An *entitlement* is a collection of permissions. Typically, it encapsulates the list of permissions needed to perform a given business function or task.

To add entitlement grants to an application policy store:

1. Open your application in the Application Navigator.
2. In the main menu, choose **Application > Secure > Entitlement Grants** to open the Entitlement Grants page in the overview editor for the `jazn-data.xml` file.
3. To add an entitlement, click the **Add Entitlement** icon in the **Entitlements** list.
4. To add a member resource, click **Resources**, and in the **Member Resources** list, click the **Add Member Resource** icon.
5. To select the application role to grant the entitlement to, select **Grants** and then click the **Add Role Grant** icon. In the Select Application Roles dialog, you can select an application role or create a new one.
6. Save the `jazn-data.xml` file.

Tips:

- You can view grants to resources that are members of an entitlement group in the Resource Grants page by clicking the **Show Grants from Entitlements** icon in the **Granted To** column. This option is selected by default.
- You can also add member resources to new or existing entitlements from the context menu in the Resource Grants page.

14.6.8 How to Create a Custom JAAS Permission Class

A new permission class is useful when you want to create your own JAAS permission for a logical artifact type to secure. For example, although Oracle ADF already provides built-in permission classes for the artifacts on which it enforces security (including task flows, page definitions, entity objects, and entity attributes), you might create a custom permission class for a set of UI components that you want to secure in the user interface. Once this class is created, you can add enforcement checks using Java, Expression Language (EL), or embedded Groovy expressions, and then you can grant the new custom permission class to application roles by editing the jazn-data.xml file directly. For example, you could define a security policy to limit access to a menu that your application displays and then associate the rendering of the menu with the user's granted custom permission using the EL value userGrantedPermission on the component's rendered property.

To create a custom JAAS-compliant permission class:

1. Open your application in the Application Navigator.
2. From the main menu, select **File > New** to open the New Gallery.
3. In the New Gallery, under **Categories**, select **Business Tier > Security**.
4. Under Items, select **JAAS Permission**.

5. In the Create JAAS Permission dialog, enter the details of the custom permission class. For any help from within the dialog, click **Help** or press F1.

14.6.9 How to Add Grants to the System Policy Store

Currently, this release does not provide an editor to add system permission grants to a system policy store; you will need to manually add grants in the source code for `jazn-data.xml`.

To add a grant to the system policy:

1. Open your application in the Application Navigator.
2. In the Application Navigator, double-click the `jazn-data.xml` to open the overview editor.
3. Click **Source** to open the source editor.
4. In the source code, inside the `<jazn-data>` element, create a `<jazn-policy>` element.
5. Inside the `<jazn-policy>` element create a `<grant>` element that defines the `<grantee>` with the desired application role and the `<permission>` with the fully qualified class name of the permission class, the name that you want to use as the target for the grant, and the action that you want to grant to the application role principal.
6. Save changes to the `jazn-data.xml` file.

14.7 Migrating the Policy Stores

JDeveloper is configured by default to deploy the security objects from your application repositories to Integrated WebLogic Server each time you run the application. You can change this behavior by selecting security deployment options in the Application Properties dialog to:

- Decide whether to overwrite the domain-level policies with those from the application `jazn-data.xml` file.
- Decide whether to overwrite the system credentials from the application's cwallet.sso file.
- Decide whether to migrate the identity store portion of the jazn-data.xml file to the domain-level identity store.

If you make no changes to the deployment settings, each time you run the application, JDeveloper will overwrite the domain-level security policies and system credentials. Additionally, JDeveloper will migrate new user identities you create for test purposes and update existing user passwords in the embedded LDAP server that Integrated WebLogic Server uses for its identity store. However, if you prefer to run the application without updating the existing security objects in Integrated WebLogic Server, you have this option.

14.7.1 How to Migrate the Policy Stores

When you are ready to deploy the application to standalone Oracle WebLogic Server, you can use the same configuration settings to control how JDeveloper handles migration of the security objects.

To configure deployment of security objects:

1. Choose **Application > Secure > Configure Security Deployment** to open the Application Properties dialog.
2. In the Application Properties dialog, under **Security Deployment Options**, select the security objects that you want to deploy with the application.

   By default, each time your run the application, JDeveloper will overwrite the application policies and credentials at the domain level with those from the application. If you prefer not to overwrite either of these repositories, deselect **Application Policies** or **Credentials**. When deselected, JDeveloper will merge only new policies or credentials into the domain-level stores. For further details, see the sections below.

   By default, each time you run the application, JDeveloper will migrate new user identities you create for test purposes and update existing user passwords in the embedded LDAP server that Integrated WebLogic Server uses for its identity store. You can disable migration of the application identity store by deselecting **Users and Groups**. For further details, see the sections below.
3. Click **OK**.

14.7.2 Migrating Application Policies

Application policies, specified in `jazn-data.xml`, can be migrated to a domain policy store when the application is deployed to a server in the Oracle WebLogic Server environment. If desired, the policies can also be removed from the domain policy store when the application is undeployed, or updated when the application is redeployed.

If **Application Policies** is selected in the Application Properties dialog, a `jps.policystore.migration` property is set to `OVERWRITE` in the packaged weblogic-application.xml when you deploy the application using JDeveloper. If **Application Policies** is unselected, the `jps.policystore.migration` setting will not be added to the packaged `weblogic-application.xml`, and will be removed if it is already present. This causes the default operation `MERGE` to be used by Oracle WebLogic Server. Merge will only migrate policies the first time the application is deployed if they do not already exist. If the policies for the application already exist, they will not be remigrated.

To find out more about automatic and manual migration of application policies, see the *Oracle Fusion Middleware Security Guide*.

14.7.3 Migrating Credentials

When you migrate your application policies, you might also want to migrate your credentials. Application credentials, specified in `cwallet.sso`, can be migrated to a domain credential store when the application is deployed or redeployed to a managed server in the WebLogic environment. Thus, credential migration includes the passwords for all connections created within JDeveloper, including those created for web services. (This is not related to user credentials specified in the identity store of the `jazn-data.xml` file. See Section 14.7.4, "Migrating Users and Groups" below for details about identity store migration.)

If **Credentials** is selected in the Application Properties dialog, a `jps.policystore.migration` property is set to `OVERWRITE` in the packaged `weblogic-application.xml` when you deploy the application in JDeveloper. If **Credentials** is unselected, the `jps.policystore.migration` setting will not be added to the packaged `weblogic-application.xml`, and will be removed if it is

already present. This causes the default operation `MERGE` to be used by Oracle WebLogic Server. Merge will only migrate credentials the first time the application is deployed if they do not already exist. If the credentials for the application already exist, they will not be remigrated.

The credential migration is possible only when the server is running in development mode only. In production mode, credential overwrite is prohibited. Application credentials must be manually migrated when you deploy using tools outside of JDeveloper.

14.7.4 Migrating Users and Groups

Users and roles, specified in `jazn-data.xml`, can be migrated to a domain identity store when the application is deployed to a server in the WebLogic environment.

If **Users and Groups** is selected in the Application Properties dialog, JDeveloper will make calls when you deploy the application to create Oracle WebLogic Server users and groups corresponding to the application's `jazn-data.xml` users and role. If the user already exists in the domain store, only the description and password will be remigrated during deployment. If a group exists in the domain store with the same name as the roles in the `jazn-data.xml` file, it will be replaced entirely. If **Users and Groups** is unselected, JDeveloper will not try to migrate the identity store from the application `jazn-data.xml`.

Note: Before migrating users and groups ensure that administrator roles (`admin`) and users (`weblogic`) are not used in the application `jazn-data.xml` file so that the domain identity store is not overwritten. When your application is ready for deployment to a production environment, you should remove the identities from the `jazn-data.xml` file or disable the migration of identities by deselecting **Users and Groups** from the Application Properties dialog.

14.8 Securing Development with JDBC

A JDBC database connection created in JDeveloper derives its encryption properties from the database client install on your machine. To create a secure connection using JDBC:

- Configure encryption support using the OCI driver by setting parameters in the sqlnet.ora file on your client machine.
- Use the thin JDBC driver to create a secure JDBC connection in JDeveloper. To do this, select **Enter Custom JDBC URL** in step 3 (Connection page) of the Create Database Connection Wizard, then enter your encryption parameters as part of a custom JDBC URL, as shown in Example 14–3.

Example 14–3 Encryption Parameters

```
jdbc:oracle:thin:@(description
=(address=(protocol=tcp)(host=myhost)(port=1521))(connect_data=
(sid=ORCL)(SQLNET.ENCRYPTION_CLIENT=REQUIRED)(SQLNET.ENCRYPTION_TYPES_
CLIENT=DES40)(SQLNET.CRYPTO_CHECKSUM_CLIENT=REQUESTED)
(SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENTMD5=MD5) ))
```

15

Developing Applications Using XML

This chapter describes how to create and update applications using the XML tools and editors provided by JDeveloper.

This chapter includes the following sections:

15.1 About Developing Applications Using XML

JDeveloper provides you with the tools you need to work with the XML files in your application. There is an XML source editor, an XML validator, and tools for working with XML schemas. You can also use JDeveloper to create and edit your XSQL files. It provides a robust XML editing environment that allows you to create and edit many different types of XML files in your application.

XML Schema development is easy with JDeveloper. You can create a schema document from scratch, generate schemas from XML documents or vice-versa. Once your schema is created, manage your elements using the XSD Visual Editor and the Component Palette. For more information, see Section 15.5, "Working with XML Schemas".

You can also create XSQL files from scratch or edit existing files. JDeveloper provides a complete development environment to simplify the task of developing databound XML pages with XSQL servlet. For more information, see Section 15.6, "Developing Databound XML Pages with XSQL Servlet".

15.2 Using the XML Editors

JDeveloper provides three different editors for working with your an XML files.

- The XML editor is a specialized schema-driven editor for editing XML languages, including XSQL, XSL, XSD, XHTML, and WSDL files.
- The Overview editor allows you to view and edit XML files. It visually displays aspects of your deployment-related XML files such as filters, security and

references. For more information, see Chapter 11, "Developing Applications Using Web Page Tools."

- The XSD Visual editor allows you to create or edit XML schemas. It visually displays the structure, content, and semantics of an XML document. For more information, see Section 15.5.3, "Understanding the XSD Component Display in the XSD Visual Editor."

15.2.1 Understanding XML Editing Features

Table 15–1 summarizes the editing features that are available when you're working with XML files.

Table 15–1 XML Editing Features

Feature	Purpose
Code Insight	While you are typing, you can invoke Code Insight by pausing after typing the < (opening bracket) or by pressing **Ctrl+Space** if you are using the default keymapping. Code Insight opens a list with valid elements based on the grammar. After selecting an element, enter a space and then either pause or press **Ctrl+Space** to open a list of valid attributes from which you can select. After you enter the opening quote for the attribute value, either the required type of value or a list of available values is provided.
XML Validation	In an open XML Source Editor window, or in the Application Navigator, right-click an XML file and choose Validate XML. The Validate XML command will validate the XML against a schema registered with JDeveloper defined in the XML file. To register a schema with JDeveloper choose **Tools** > **Preferences** > **XML Schemas**. This command on the context menu is disabled whenever an XML file does not have an XML namespace defined.
Quick Form Check	Right-click on an XML file and choose **Make** to check for well-formedness of the file.
XML Schemas Preferences	Use the options on the XML Schemas page in the Preferences dialog to view all the currently registered XML schemas, to add new schemas, to support additional namespaces and elements, to remove user-defined schemas, and to unload schemas from memory. To get to the Preferences dialog choose **Tools** > **Preferences** > **XML Schemas**.
XML Preferences	These features can be customized in the XML Preferences page. Choose **Tools > Preferences** > **Code Editor** > **XML and JSP/HTML** to display XML Preferences. If **Required Attribute Insertion** is selected, the required attributes of an element will also be inserted for you. If **End Tag Completion** is selected, the end tag will be automatically inserted when you close the start tag, for example if you have `<foo` and you type the `>`,`</foo>` is added automatically.
Component Palette	You can choose **View** > **Component Palette** to open the palette and select one of the available pages from the dropdown list. For example, while editing XSD files, you can select elements from the XML Schema pages on the palette.
Property Inspector	The Property Inspector displays attributes of elements in the file. You can edit the values of attributes in the Property Inspector to update your file.

Table 15–1 (Cont.) XML Editing Features

Feature	Purpose
Structure Window	A file's elements are displayed hierarchically in the Structure window, which also displays any XML syntax errors found as you type and edit. You can double-click on an element or error to edit it in the XML editor.
Validate XML	In an open XML editor window, or in the Application Navigator, right-click an XML file and choose **Validate XML**. The Validate XML command will validate the XML against the schema defined in the XML file. It validates the XML constraints and definitions but not XSDs. This context-menu command is disabled whenever an XML file does not have an XML namespace defined.
F2 Key	After creating an XML schema, select an element in the Structure window and press F2. The element now has focus in the XML design editor. You are automatically able to input new text for the element into the XML design editor.
Expand/Collapse Attributes	You can expand or collapse attributes that display under the complexType element. This is convenient because the list of attributes that display under the element can be large.

15.2.2 Understanding the XML Editor Toolbar

Table 15–2 contains the icons that display on the XML Editor toolbar.

Table 15–2 XML Editor Toolbar Icons

Icon	Name	Description
	Search (`Crtl + F`)	Enter search text in the XML Editor. Click the down arrow to view and set additional parameters for the search, including **Match Case** to perform a case-sensitive search, **Whole Word** to locate complete word matches only, and **Highlight Occurrences** to use shading to show the location of the match.
	Find Next (`F3`)	Click to locate the first occurrence of the text that meets the specified parameters in the file.
	Find Previous (`Shift + F3`)	Click to locate the previous occurrence of the text that meets the specified parameters in the file.

15.3 Creating XML Files in Oracle JDeveloper

The New Gallery offers several different XML file type options, as shown in Table 15–3. To create a new XML file choose **File menu** > **New** > **General** > **XML**.

Table 15–3 XML File Types

File Type	Description
XML Document	Create a new XML file that includes only the `<?xml version="1.0"?>` line at the top.
XML Document from XML Schema	Generates an XML document from an existing XML schema.
XML Localization File (XLIFF)	Creates an XML-based localization file with an .xlf extension. For more information, see Section 15.3.1, "Localizing with XML".
XML Document from XML Schema	Generates as XML Document from an XML Schema.

Table 15–3 (Cont.) XML File Types

File Type	Description
XQuery File	Creates an XQuery File with an .xq extension. For more information, see Section 15.4.2, "Using XQuery with XML".
XSL Map	Creates an XSL Map File with an .xsl extension.
XSL Map from XSL Stylesheet	Creates an XSL Map File with an .xsl extension from an XSL Stylesheet.
XSL Stylesheet	Creates an XSL Stylesheet with an .xsl extension. For more information, see Section 15.6.12, "How to Create an XSL Style Sheet for XSQL Files".
XSQL File	Creates an XSQL file with an .xsql extension. For more information, see Section 15.6.2, "How to Create an XSQL File".

15.3.1 Localizing with XML

JDeveloper has tools to support full localization for your application based on XML-based XLIFF technology. XLIFF supports a full localization process by providing tags and attributes that hold the data your translators and vendors will use when you internationalize your application.

15.3.1.1 How to Create a New XLIFF file

You create a new XMLFF file in the JDeveloper New Gallery under the XML node.

To create a new XLIFF file:

Choose **File** menu >**New** > **General** > **XML** > **XML Localization File**.

For more information on XLIFF, see the OASIS open standard website at, `http://www.oasis-open.org/home/index.php`

15.3.1.2 What You May Need to Know About XLIFF Files

The main elements in an XLIFF file are the trans-unit elements. These elements store localizable text and its translations. These elements represent segments (usually sentences in the source file that can be translated reasonably independently). The trans-unit elements contain source, target, alt-trans, and a handful of other elements.

There are also elements for review comments, the translation status of individual strings, and metrics such as word counts of the source sentences. The XLIFF file consists of one or more file elements. Each of these contains a header and a body section. The header contains project data, such as contact information, project phases, pointers to reference material, and information on the skeleton file.

JDeveloper uses Resource Bundles to hold all of the localization information, including the XLIFF files. When you create content in a JSF page, a resource bundle is automatically created for you in that project.

15.3.2 How to Import and Register XML Schemas

Use the options on the XML Schemas page in the Preferences dialog to view all the currently registered XML schemas, add new schemas to support additional namespaces and elements, remove user-defined schemas, and unload schemas from memory.

To import and register an XML schema:

1. From the main menu, choose **Tools** > **Preferences**.
2. Select the **XML Schemas** node.
3. Click **Add** to open the Add Schema dialog where you can specify a new schema to add to the list of user schemas.
4. Enter the name and location of the XML Schema file you are adding in the **Add a Schema from the file system or a URL** field.
5. Enter the file extension to register the schema for a specific file type in the **Extension** field. JDeveloper uses the extension to efficiently load the schema into memory and to display automatically created Component Palette pages based on the items in the schema.
6. Click **OK**.

   JDeveloper automatically validates the schema when you add it.
7. Confirm that the new schema has been added in the **User Schemas for XML Editing** list and click **OK**.

   Tips: You can only remove user-defined schemas with the **Remove** button.

   If a schema changes, you must use the **Clear Cache** button to unload all currently loaded schemas from memory. JDeveloper will then reload any needed schemas including the modified schema.

15.3.3 How to Add an XML Element to the Palette

You can add pages to the Component Palette in JDeveloper to include the elements from a registered schema or you can add elements to an existing page. Once you add the elements to the Palette, you can insert the elements into the XML file while you are editing, by selecting them from the Palette.

To add XML elements to the Component Palette:

1. From the main menu, choose **Tools** > **Configure Palette** to open the Configure Component Palette dialog.

   Skip to step 4 if you do not want to add a new page.
2. Click **Add** under the **Pages** list to open the New Palette Page dialog.
3. In the New Palette Page dialog, enter the name of the new page and select the appropriate type from the dropdown list, then click **OK**.

   Your new page name is added to the bottom of the **Pages** list in the Configure Component Palette dialog.
4. Select the new page name in the **Pages** list and click **Add** under the **Components** list to open the XML Elements dialog.

   The XML Elements dialog displays the Registered Schemas.
5. Expand the appropriate schema node to display the elements you can add to the Palette.
6. To add an individual element, select it in the tree. To add multiple elements, use **Ctrl-click** or **Shift-click** to select them. Then click **OK**.

   You can also click **Use Default Icon** or **Select Icon** to select the icon that will display for an individual element on the Palette, before you click **OK.**

7. After adding XML elements, click **OK** to close the Configure Component Palette dialog.

The name of the page you added displays in the dropdown list in the Palette. All the elements you added are displayed with angle brackets (< >) as the icon, if you accepted the default icon. If you do not see any element names on the Palette, right-click in the Palette and choose **List View**.

15.3.4 How to Generate Java Classes from XML Schemas with JAXB

In JDeveloper you can use JAXB (Java Architecture for XML Binding) to generate Java classes from XML schemas. JAXB is an easy way to incorporate XML data and processing functions in Java applications without having to know XML. You can generate a JAXB 1.0 or 2.0 content model, including the necessary annotations, from an XML schema.

When the JAXB binding compiler is run against an XML schema, JAXB packages, classes, and interfaces are generated. You can then use the generated JAXB packages and the JAXB utility packages in a binding framework to unmarshal, marshal, and validate XML content.

To generate Java classes from XML schemas with JAXB:

1. From the main menu choose **File** > **New** > **Business Tier** > **TopLink/JPA** and select either **JAXB 1.0** or **2.0 Content Model from XML Schema** to open the compilation dialog.
2. Select the schema file and optionally the JAXB customization file to use and the package to which the generated classes will be added.

   The JAXB package and generated classes are added to the Application Resources folder.

15.4 Editing XML Files in Oracle JDeveloper

The XML Source Editor in JDeveloper is a specialized schema-driven editor for editing XML languages including XSD, WSDL, XSQL, XHTML, and XSL files.

To edit an XML file in the XML Source Editor:

1. In the Navigator, right-click a file and choose **Open**.
2. Click the **Source** tab if not selected by default for that file.
3. While you are typing, you can invoke Code Insight by pausing after typing the < (opening bracket) or by pressing **Ctrl+Space** (if you are using the default keymapping). Code Insight opens a list with valid elements, based on the schema.
4. After selecting an element, enter a space and then either pause or press **Ctrl+Space** to open a list of valid attributes from which you can select. After you enter the opening quote for the attribute value, Tip Insight displays the type of value that is required.

   Tip: To edit an XML document with the Component Palette, choose **View** > **Component Palette** to open the Palette and select one of the available pages from the dropdown list. Then choose elements from the page.

15.4.1 How to Set Editing Options for the XML Editor

When editing XML files in the XML Editor you can set two editing options.

To customize editing options for the XML Editor:

1. Choose **Tools** > **Preferences**.
2. Expand the **Code Editor** node.
3. Select the **XML** node.
4. Select the XML node. On the XML Preferences page, select **Required Attribute Insertion** or **End Tag Completion** to enable the desired options.
5. Click **OK**.

15.4.2 Using XQuery with XML

You can create and edit your XML-based XQuery files in JDeveloper. XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents.

15.4.2.1 How to Create a New XQuery File

You create a new XQuery file in the JDeveloper New Gallery under the XML node.

To create a new XQuery file:

Choose **File** menu, > **New**, > **General** > **XML** > **XQuery File**.

For more information on XQuery, see the W3C website at,
`http://www.w3.org/TR/xquery/`

15.4.2.2 What You May Need to Know About XPath Expression Syntax

XQuery uses XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN.

The language is based on a tree-structured model of the information content of an XML document, containing seven kinds of node: document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.

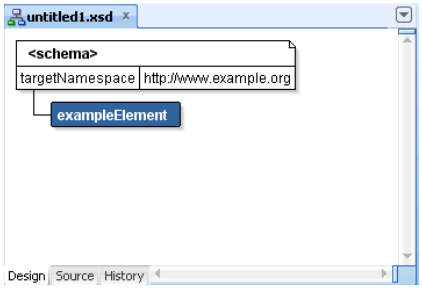
15.5 Working with XML Schemas

JDeveloper provides an XSD Visual Editor that gives a visual representation of the structure, content, and semantics of an XML document.

15.5.1 Working with Attributes in the XSD Visual Editor

You can create an XML schema's attributes and set properties and facets from using the XSD Visual Editor. Figure 15–1 contains an example XML schema in the Design tab of the XSD Visual Editor.

Figure 15–1 Schema in XSD Visual Editor



You can edit attributes in `attribute2` in the attribute editor, which is displayed in Figure 15–1 as the `union` element. In this editor, you can:

- Display all available attributes under an element. To hide or display details, click the plus and minus signs next to the attribute.
- Display all facets and type details of an attribute display in the attribute node
- Display the default "Insert Into" menu with the valid schema components (for example, `union`) when you right-click on an attribute node.
- Expand an attribute node within to display a subtree containing child nodes like list or union.

15.5.2 What Happens When You Create an XML Schema in the XSD Visual Editor

As you create an XML Schema in the XSD visual editor, JDeveloper automatically updates the XML source in the design tab, as well as updating the contents of the Structure window. Example 15–1 contains the source for the `example.xsd` file shown in Figure 15–1.

Example 15–1 XML Source

```
<?xml version="1.0" encoding="windows-1252" ?>
            xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.example.org"
            targetNamespace="http://www.example.org"
<elementFormDefault="qualified">
    <xsd:complexType name="UnionTest">
      <xsd:sequence>
      <xsd:element name="element1">
    <xsd:complexType>
       <xsd:attribute name="attribute1">
       </xsd:attribute>
    </xsd:complexType>
       </xsd:element>
       <xsd:element name="element2">
    <xsd:complexType>
       <xsd:attribute name="attribute2">
          <xsd:simpleType>
            <xsd:restriction>
              <xsd:simpleType>
                <xsd:union/>
              </xsd:simpleType>
              <xsd:pattern value="abcd"/>
            </xsd:restriction>
          </xsd:simpleType>
```

```
        </xsd:attribute>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

15.5.3 Understanding the XSD Component Display in the XSD Visual Editor

The JDeveloper XSD Visual Editor provides a visual representation of the structure, content, and semantics of an XML document. Use the XSD Visual Editor to author a new, or to edit an existing, XML Schema.

15.5.3.1 XSD Component Selection

The selection of any component or attribute in the editor is indicated by highlighting the selected item in blue. In Figure 15–2, the selected `simpleType` component defines a simple type and specifies the constraints and information about the values of attributes or text-only components, in this case restricting the string type.

Figure 15–2 simpleType Component

15.5.3.2 XML Schema Component

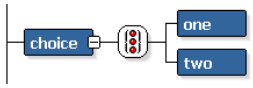
The XML Schema component is displayed at the top of an XSD file, as shown in Figure 15–3. Right-click the element and select **Properties** to display a dialog for configuring the schema namespaces.

Figure 15–3 XML Schema Component

15.5.3.3 Choice Component

The choice component allows only one of the components contained in the <choice> declaration to be present within the containing component, as shown in Figure 15–4. Set attribute `maxOccurs` to >1 to have more than one item from the choice in the parent.

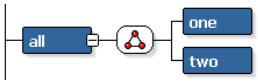
Figure 15–4 Choice Component



15.5.3.4 All Component

The all component shown in Figure 15–5 specifies that the child components can appear in any order and that each child component can occur zero or one time.

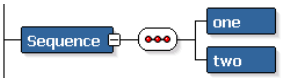
Figure 15–5 All Component



15.5.3.5 Sequence Component

The sequence component shown in Figure 15–6 specifies that the child components must appear in a sequence. Each child component can occur from 0 to any number of times.

Figure 15–6 Sequence Component

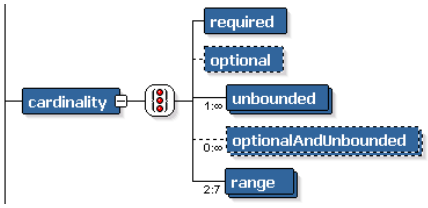


15.5.3.6 Cardinality and Ordinality

In the example of cardinality shown in Figure 15–7, components are displayed with the following attributes:

- Required components (minOccurs=">0") are displayed with a solid line.
- Optional components (minOccurs="0") are displayed with a dotted line.
- Unbounded components(maxOccurs="unbounded") display an infinity symbol in the component stack number. Any component that can appear more than once is displayed as a "stack" of components. In the numbers to the left of the component the number before the colon indicates the minimum number of times the component can occur (minOccurs) and the number after the colon indicates the maximum number of times the component can occur (maxOccurs). In the illustration the maximum is unbounded so an infinity symbol is displayed.
- Range of components is displayed in the component stack number. In the illustration the component must appear at least 2 times in the instance document, but no more than 7.

Figure 15–7 Cardinality Component



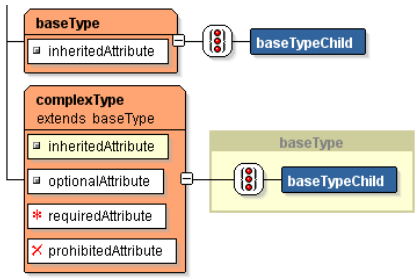
15.5.3.7 ComplexType Component

In Figure 15–8, the complexType component extends a base type, and inherits an attribute and children from that base type. The yellow background represents a reference to the baseType defined elsewhere in the schema and illustrated below the complexType component. The component attributes are displayed as:

- Inherited, marked with a square.
- Optional, marked with a square.
- Required, marked with an orange asterisk.

- Prohibited, marked with an orange X.

Figure 15–8 complexType Component

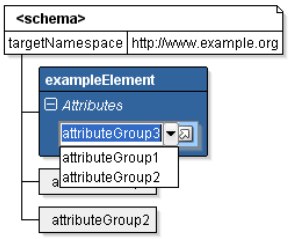


15.5.3.8 Attribute Group Component

The attribute group component groups a set of attribute declarations so that they can be incorporated as a group into complex type definition.

Figure 15–9 displays three attribute groups.

Figure 15–9 Attribute Group Component

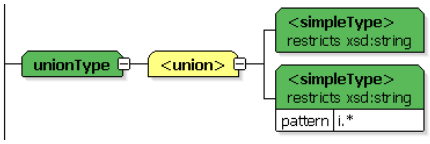


If you add an element to a schema that has multiple attributeGroups, you can add choose one or more attributeGroups for the element by clicking on the element's attribute and choosing from a drop-down list.

15.5.3.9 Union Component

The union component defines a simple type as a collection (union) of values from specified simple data types. In Figure 15–10, the union represents all strings that begin with the letter "i".

Figure 15–10 Union Component



15.5.3.10 List Component

The list component defines a simple type component as a space separated list of values of a specified data type. In Figure 15–11, the component represents a series of short value objects.

Figure 15–11 List Component

15.5.4 How to Generate an XML Schema from XML Documents

Use the XML wizard in the New Gallery to help you quickly generate industry standard W3C XML schema (`.xsd`) from your XML (`.xml`) documents. Conversely, you can also generate XML documents from your XML schema.

To generate an XML schema from an XML document:

1. Choose **File menu** > **New** > **General** > **XML** > **XML Schema from XML Document**.
2. Enter the information as directed.

To generate an XML document from an XML schema

1. Choose **File menu** > **New** > **General** > **XML** > **XML Documents from XML Schema**.
2. Enter the information as directed.

15.5.5 How to Generate an XSD File from a DTD File

You can generate an XML schema document (XSD) file from a document type definition (DTDs) file.

To generate an XSD file from a DTD:

1. On the main menu, click **Tools**.
2. In the **Tools** menu, click **Convert DTD to XSD**.

15.5.6 How to Display an XSD File for Editing

Open a schema (`.xsd`) file for editing in the XSD Visual Editor or XML Source Editor. By default new schema files are opened with the XSD Visual Editor in focus. Double-clicking a file in the Application Navigator opens or brings the default editor on the **Design** tab to the foreground. Clicking the **Source** tab opens the file in the XML Source Editor. Changes made in one editor are automatically updated in the other editor.

A schema file (`.xsd`) can be edited simultaneously with the visual and source editors by opening the page in one of the editors and using the splitter to open a second page view in the alternate editor.

To display a schema file in both editors:

- To split the file horizontally, grab the splitter just above the vertical scroll bar (on the upper right-hand side of the window) and drag it downward.
- To split the file vertically, grab the splitter just to the right of the horizontal scroll bar (on the lower right-hand side of the window) and drag it left.

15.5.7 How to Create an Image of the XSD Visual Editor Design Tab

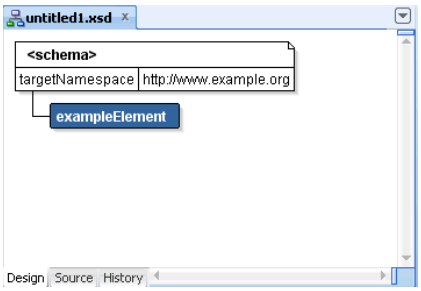
You can create the design tab of the XSD Visual Editor as an image. You can then share the image as a file or print out or image with others.

Supported image formats are `.svg`, `.svgz`, `.jpg`, and `.png`.

1. In the Application Navigator, double click the `.xsd` file you want to display in the XSD Visual Editor.
2. Click the Design tab in the XSD Visual Editor.

   A design view of the .xsd file displays, similar to Figure 15.6.

Figure 15–12 Design Tab in XSD Visual Editor



3. Right-click anywhere on the Design tab and choose **Publish Diagram**.
4. Enter a name, the path where you want to save the diagram, and the image type you want to use.

Notes: If the diagram you are attempting to save is too large, a message displays indicating that the image should be saved in `.svg` format.

If you right-click on a node in the XSD Visual Editor, only the current node and its child nodes are saved as an image.

15.5.8 How to Navigate with Grab Scroll in the XSD Visual Editor

In the XSD Visual Editor, you can quickly navigate an XML Schema that displays with scroll bars using a grab scroll operation. Use the grab scroll to invoke a small hand cursor to grab an XML Schema page and drag it inside the editor window.

To navigate using grab scroll in an XML Schema:

1. In the XSD Visual Editor press and hold down the spacebar.

   The pointer turns into an open hand cursor.
2. Press and hold down the left mouse button.

   The hand closes and grabs the XML Schema page.
3. Use your mouse to move the XML Schema page inside the editor window.
4. Release the XML Schema page by releasing the left mouse button.
5. Close grab scroll by releasing the spacebar.

15.5.9 How to Expand and Collapse the XSD Component Display

While working in the XSD Visual Editor or Design structure window, you can expand or collapse XSD components to display children components or collapse container components to create a higher level view of the schema.

To expand one level beyond the parent component:

Click the + (plus) sign of the parent component.

To collapse all levels below the parent component:

Click the - (minus) sign of the parent component.

To expand all parent components in the schema:

Press **Ctrl + ***, using the * on the numeric keypad of the keyboard.

> **Note:** This view can be big.

15.5.10 How to Zoom In and Out in the XSD Visual Editor

Zooming enables you to magnify (zoom in) or shrink (zoom out) on the display of an XML Schema in the XSD Visual Editor.

To zoom in:

1. Place your cursor in the area of the XML Schema you wish to magnify.
2. Press **Ctrl+Plus**.

> **Note:** Use the **Plus** on the numeric keypad of the keyboard.

To zoom out:

1. Place your cursor on the area of the XML Schema you wish to shrink.
2. Press **Ctrl+Minus**.

> **Note:** Use the **Minus** on the numeric keypad of the keyboard.

15.5.11 How to Select XSD Components

One of the most common actions you perform in the XSD Visual Editor or Structure window (Design or Source view) is to select components in order to do something with them. There are several reasons for selecting components:

- Edit the properties of the component(s)
- Move the component(s)
- Delete the component(s)
- Select a target position in which to insert another component

You can select a single component without children, a component along with its children, and multiple components.

To select a component:

- Click the component.

  If the selected component contains children, selecting the component also selects all its children. If you copy, move, or delete the parent, all its children are also copied, moved, or deleted.

Tip: Double-clicking an XSD component in the XSD Visual Editor displays a property editor for the component.

To select multiple components:

1. Click the first component.
2. Press and hold down the **Ctrl** key.
3. Click any additional components. If you want to deselect one without losing the other selections, continue to hold down the **Ctrl** key and click the component again.

   Selecting multiple, non-adjacent components for any reason other than deleting them can lead to unexpected results. For example, if the components exist at different levels in the schema hierarchy, they can lose their relative hierarchical positions if you move or copy them to another position in the schema page.

   In the XSD Visual Editor it is possible to select a container component (and thereby select its children) and also explicitly select one or more of the children. That means that any explicitly selected child is selected twice. If you do this and then copy and past the selection, the double-selected child will be pasted twice, once as a child to the copied parent and once as a peer to the copied parent.

15.5.11.1 What Happens When You Select a Component in the XSD Visual Editor

When a component is selected in the XSD Visual Editor, the component displays in blue. When a container component is selected and any of its children are also explicitly selected, all are displayed in blue.

When selected in the Structure window (Design or Source view), the component is highlighted. However, when you select any components with children, the children are also selected with it, even if their names are not selected. If you delete or move the parent, all the children are deleted or moved with it.

Whenever you select an component, you are also selecting a position in which another component can be inserted. For more information, see Section 15.5.12, "How to Select Target Positions for XSD Components".

Tips: When you pass the mouse pointer over a component, a tooltip with the component's name is displayed. That makes it easier to know where to click to select a component.

When you select a component in the XSD Visual Editor, it is also selected in the Design and Source view of Structure window, and vice versa. That means that you can look at the selection in both tools to clarify what is selected and where the insertion position is.

The JDeveloper status bar explicitly states the insertion point for a selected component.

15.5.12 How to Select Target Positions for XSD Components

While inserting, copying, or moving XSD components in the XSD Visual Editor or Structure window (Design or Source view), you need to select a target position in relation to the node on which you are performing the activity. The possible target positions on a node are before, after, and inside.

To select a target position:

Choose from one of the following options:

- Select the target position by clicking the node on which you are performing the action.
- When dropping a component at a target position, do one of the following:
  - To insert a component before a target node, drag it towards the top of the node until you see a solid horizontal line (visual editor) or horizontal line with an embedded up arrow (structure), then release the mouse button.

  - To insert a component after a target node, drag it towards the bottom of the node until you see a solid horizontal line (visual editor) or a horizontal line with an embedded down arrow (structure), then release the mouse button.

  - To insert a component inside a target node, drag it over the node until it is surrounded by a box outline, then release the mouse button. This target position is available only on nodes that can contain child nodes.
- When using the context menu to select a target position, right-click the target node, choose an option, and then select a component. The options are:
  - I**nsert before <component>** - inserts a component before the selected node.
  - **Insert inside <component>** - inserts a component inside (under) the selected node.
  - **Insert after <component>** - inserts a component after the selected node.

  Not all options are always available. Choosing an option displays a submenu from which you can choose a component list and then select the component you desire. Depending on the node you select, the submenu may also contain one or more components that are eligible for insertion inside the selected node.

> **Note:** When you select a target position in the Design or Source views in the Structure window, the selection is also reflected in the XSD Visual Editor, and vice versa. This enables you to verify the insertion position visually as well as hierarchically. The selection is also explicitly stated in the status bar at the bottom of the JDeveloper window.

15.5.13 How to Insert XSD Components

In the XSD Visual Editor and Structure window (Design and Source view), you can insert XSD components by dragging from the Component Palette or by using a context menu. You can also insert XSD components by copying or by cutting and pasting. If you are cutting and pasting, you can insert multiple components at a time. For more information, see Section 15.5.16, "How to Cut, Copy, and Paste XSD Components".

> **Note:** Pasting multiple components that were copied from different places in the XML schema hierarchy can lead to unexpected results.

To insert XSD components using the Component Palette:

1. In the XSD Visual Editor or Structure window, locate the desired position where you wish to insert a component. You may have to expand nodes in the Structure window to uncover the node you want.
2. In the Component Palette, select an XSD component list from the dropdown list box, and then drag the desired component from the list and drop into the desired target position in the XSD Visual Editor or Structure window.

You can also select the target position in the visual editor or Structure window and then click the desired component in the Component Palette.

> **Tip:** A brief description of a component appears when the cursor is placed over a component name in a list. For detailed help, right-click a components in the list and choose **Help**.

To insert XSD components using the context menu:

1. In the XSD Visual Editor or Structure window, right-click the desired node to display a context menu. You may have to expand nodes to uncover the node you want.
2. Choose an option in the context menu, and then select a component. The options are:
   - **Insert before <component>** - inserts a component before the selected node.
   - **Insert inside <component>** - inserts a component inside (under) the selected node.
   - **Insert after <component>** - inserts a component after the selected node.

   Not all options in the context menu are always available. Choosing an option displays a submenu from which you can choose a component list and then select the component you desire. Depending on the node you select, the submenu may also contain one or more components that are eligible for insertion inside the selected node.

15.5.14 How to Set and Modify XSD Component Properties

The Property Inspector displays the properties of XSD components selected in the XSD Visual Editor or the Structure (Design or Source view) window. Use the Property Inspector to set or modify the property values for any component in your XML Schema. Set property values are marked with a green square.

To undo changes, from the main menu select **Edit**, > **Undo *action***. Use the **Set to Default** button to reset a property that has been set to its default value (if any).

To set a component's properties:

1. With an XML Schema open, select a component in the visual editor or Structure window.

   The Property Inspector displays the property values for the selected component. If the Property Inspector is not in view choose **View** > **Property Inspector** or use the shortcut **Ctrl+Shift+I**.
2. Scroll until the property you want is visible, then select it with the mouse or the arrow keys.

   A brief description of the property is displayed at the bottom of the Property Inspector.

Tip: To quickly locate a property in a long list, click the search button in the Property Inspector toolbar. In the **Find** text field, type the name of the property, then press **Enter**.

3. Enter the property value in the right column in one of the following ways:
   - In a text field, type the string value for that property, for example a text value or a number value, then press **Enter**.
   - In a value field with a down arrow, click the down arrow and choose a value from the list, then press **Enter**.
   - In a value field with an ellipsis (...), click the ellipsis to display an editor for that property. Set the values in the property editor, then press **OK**.

     Tips: Double-click an XSD component or right-click the component and choose Properties to display a property editor for the component.

     In the property editor select an attribute and view a brief description in the status area below the editor.

     Click **Help** in the property editor for a link to a component reference topic.

15.5.15 How to Set Properties for Multiple Components

If you have multiple components selected, by default the Property Inspector displays all the properties of the selected components. Click the Union button in the Property Inspector toolbar to toggle between displaying all the properties of the selected components (union) and displaying only the properties that the selected components have in common (intersection). Values represented in italic font indicate common properties that have differing values.

To set properties for multiple components:

1. Hold down the **Ctrl** key and select each of the components.
2. To change the list of properties displayed by the Property Inspector, click the **Union** button in the Property Inspector toolbar.
   - **Selected state** displays all the properties of the selected components.
   - **Unselected state** displays only the properties the selected components have in common.
3. Select and edit the desired property in the Property Inspector.

   If the value is shown in italic font, the selected components have differing values. Editing the value of a shared property will cause all selected components to have the same value.

15.5.16 How to Cut, Copy, and Paste XSD Components

You can cut, copy, and paste XSD components in the XSD Visual Editor or Structure (Design or Source) window. You can perform these operations between files of the same project or different projects.

15.5.16.1 Cutting Components

When you cut a component, it is removed from the editor and placed into a local clipboard only accessible by JDeveloper, not to the system clipboard. If you quit

JDeveloper without pasting the component, the cut version of the component will be lost.

Deleting a component removes it without changing the contents. If you get in the habit of using the cut command to remove items permanently, there is a chance that one day you will inadvertently replace something in the clipboard that you would rather have kept. For more information, see Section 15.5.18, "How to Delete XSD Components".

To cut one or more components:

1. Select the XSD component you wish to cut in the visual editor or the Structure window.
2. Do one of the following:
   - Press **Ctrl+X**.
   - Right-click and select **Cut**.
   - Choose **Edit** > **Cut** from the main menu.

15.5.16.2 Copying Components

You can copy XSD components in the visual editor or the Structure window.

To copy one or more components:

1. Select the XSD component you wish to copy in the visual editor or the Structure window.
2. Do one of the following:
   - Press **Ctrl+C**.
   - Right-click and select **Copy**.
   - Choose **Edit** > **Copy** from the main menu.
   - Hold down **Ctrl** and drag a copy of the selected component to a target position.

15.5.16.3 Pasting Elements

The elements you cut or copy from the XSD Visual Editor or Structure window can be pasted into any other XSD file in JDeveloper. For more information, see Section 15.5.12, "How to Select Target Positions for XSD Components".

To paste an element:

1. Open the file in which you want to paste a XSD element in the visual editor or Structure window.
2. Select the insertion point where you want to paste the element.
3. Do one of the following:
   - Press **Ctrl+V**.
   - Right-click and select **Paste**.
   - Choose **Edit** > **Paste**.

15.5.17 How to Move XSD Components

You can move an XSD component to a new insertion point in the XSD Visual Editor or Structure (Design or Source view) window by dragging or by cutting and pasting. You

can work in the visual editor or the Structure window to move components or work in both at once, moving components between the editors. Move an XSD component to a valid insertion point in another file in the same project or a different project by cutting and pasting. For more information, see Section 15.5.11, "How to Select XSD Components".

You can move one or multiple components at a time. However, you should be aware that selecting and moving multiple, non-adjacent components or multiple components from different levels in the schema hierarchy can lead to unexpected results.

To move components by dragging

In the visual editor or Structure window do either of the following:

- Drag the component(s) from the original position to a target position in the visual editor or Structure window. For more information, see Section 15.5.12, "How to Select Target Positions for XSD Components".
- Right-click drag the component(s) from the original position to an insertion point in the visual editor or Structure window, and then choose **Move Nodes Here** from the context menu.

To move components by cutting and pasting:

In the visual editor or Structure window do either of the following:

- Cut the component(s). Then, paste into some other position in the visual editor or Schema structure window.
- Cut the component(s). Then, paste into another file in the same project or a different project.

Note: The selected components and all of its child components are moved to the new target position.

15.5.18 How to Delete XSD Components

You can remove components from your XML Schema in the XSD Visual Editor or Structure (Design or Source view) window. When you delete a component, JDeveloper deletes the associated lines from the source code.

To delete one or more XSD components:

1. Select one or more XSD components you wish to delete in the visual editor or Structure window. For more information, see Section 15.5.11, "How to Select XSD Components".
2. Do one of the following:
   - Press the **Delete** key.
   - Press **Ctrl+X**.
   - Right-click and select **Delete**.
   - Choose **Edit** > **Delete** from the main menu.

15.6 Developing Databound XML Pages with XSQL Servlet

JDeveloper provides a complete development environment to simplify the task of developing databound XML pages with XSQL servlet.

15.6.1 Supporting XSQL Servlet Clients

JDeveloper provides support for XSQL Servlet with these features:

- Provides XSQL tags on the Component Palette
- Lets you automatically create XSQL pages
- Includes XSQL libraries
- Provides XSQLConfig.xml on the classpath; you can modify it as needed
- Provides business component action handler tags so XSQL pages can use a business logic tier to access data

15.6.1.1 What is XSQL Servlet?

XSQL servlet lets you create and use XSQL pages as clients. These pages are written in XML with embedded SQL queries and other data manipulation language (DML) statements. In addition, you can use action handlers to provide more functionality than SQL, such as writing the XML data to a file.

An action handler is an application that allows you to call a Java class from within an XSQL page. There are predefined action handlers that can talk directly to the database or to Business Components for Java (BC4J), and you can create your own.

An XSQL Servlet application has these logical layers:

- **Client** - XSQL pages take care of querying and getting data by using XML with embedded SQL. To present the data, you need to convert the XML data to another form, such as HTML, wireless markup language (WML), and so on. You can write XSL style sheets to convert XML to any of these languages.
- **XSQL Servlet in a Web Server** - The servlet uses the XML SQL Utility to talk to a database.
- **Business Logic Tier** - You can optionally use a Business Components for Java tier to access and modify data.
- **Database** - You can use any database supporting JDBC 2.0 drivers.

15.6.1.2 How Can You Use XSQL Servlet?

XSQL servlets offer a simple and productive way to get XML in and out of the database. Using simple scripts developers can:

- Generate simple or complex XML documents
- Apply XSL style sheets to generate any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

For example, a file such as `emp.xsql` in Example 15–2:

Example 15–2 emp.sql File

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scottDS">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
     select e.ename, e.sal, d.dname as department
     from dept d, emp e
     where d.deptno = e.deptno
```

```
  </xsql:query>
</FAQ>
```

Generates the XML in Example 15–3:

Example 15–3 XML File

```
<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
...
  </EMP>
</EMPLOYEES>
```

With JDeveloper, you can easily develop and execute XSQL files. The built-in web server and your default web browser will be used to display the resulting pages.

For more information on XSQL Servlet, see your Oracle database documentation.

15.6.2 How to Create an XSQL File

With JDeveloper, you can easily develop and execute XSQL files. The built-in web server and your default web browser will be used to display the resulting pages.

To create an XSQL file:

1. In the Navigator, select the project in which you want to create the new XSQL page.
2. Choose **File** > **New** to open the New Gallery.
3. In the **Categories** tree, expand **General** and select **XML**.
4. In the Items list, double-click **XSQL Page**.

This will add a skeleton XSQL file named `untitled#.xsql` to your project, which opens in the XML Editor. You can type code in this editor, add tags by selecting them from the Component Palette, and modify the file with your own style sheet information.

15.6.3 How to Edit XML Files with XSQL Tags

JDeveloper's XML Editor supports syntax highlighting, Structure window view, and the Property Inspector. You can also select tags from the Component Palette to insert in your pages while you are editing.

To use the XML Editor to edit an XSQL file:

1. In the Navigator, right-click an XSQL file and choose **XML Editor**.
2. Choose **View** > **Componenet Palette** to open the Component Palette and select the **XSQL** tag page from the dropdown list in the Palette. You can then select XSQL tags from the Palette.
3. While you are typing, you can invoke Code Insight by pausing after typing the < (opening bracket) or by pressing **Ctrl+Space** (if you are using the default keymapping). Code Insight opens a list with valid tags.

4. After selecting a tag, enter a space and then either pause or press Ctrl+Space to open a list of valid attributes from which you can select. After you enter the opening quote for the attribute value, Tip Insight displays the type of value that is required.
5. While you are editing, or after you finish, you can right-click in the file and choose **Auto Indent XML** to properly indent the file.
6. You can also right-click in any tag and choose **Locate in Structure** to highlight that tag in the Structure window.

15.6.4 How to Add XSQL Tags

All XSQL tags can be inserted by selecting them from the Component Palette, as described below. You can also insert XSQL tags by typing them in the file. Code Insight is available for XSQL tags.

To add XSQL tags to a file:

1. In the Navigator, select the XSQL file to which you want to add tags, right-click and choose **XML Editor** to open the source file.
2. Place your cursor in the blank line after the `<page xmlns:xsql="urn:oracle-xsql">` tag.
3. Choose **View** > **Component Palette** to open the Palette if it is not displayed.
4. Select X**SQL Tags** from the dropdown list in the Palette if it is not displayed.
5. Select the appropriate tag from the Palette.

   If the tag has no attributes, it appears in the XSQL page immediately. If the tag has one or more attributes, a dialog displays.
6. In the dialog that displays, enter the required and any optional attributes. Press **F1** or click **Help** in the dialog to get help for an XSQL tag and its attributes.
7. After entering attributes, click **Next** to display the next dialog or click **Finish** if it is enabled.

   The button you see and the number of dialogs depend on which tag you select. Notice that the tag and attributes you entered appear in the XSQL page.
8. Add another line in the source file and select another tag from the Component Palette if necessary.
9. When you have finished adding tags, choose **File** > **Save All** to save all your work thus far.

After adding tags, you can view the raw XML data or format the XML data with a style sheet.

15.6.5 How to Check the Syntax in XSQL Files

You can check your XSQL file to determine if it is a well-formed XML document and if not, to find any errors.

To check the syntax in an XSQL file:

In the Navigator, or in an open XML Editor window, right-click an XSQL file and choose **Check XML Syntax**.

The results display in the Log window.

Note: The **Validate XML** command on this context menu is disabled whenever an XML file does not have an XML namespace defined.

15.6.6 How to Create XSQL Servlet Clients that Access the Database

You can create XML based clients for XSQL servlets using XSQL tags. XSQL servlets allow you to easily get data in and out of the database in XML format. The following procedure shows how to use the XSQL Query tag to display data

To create an XSQL servlet client that directly accesses the database:

1. Create a new project in the workspace that contains the Business Components project by selecting the workspace in the Navigator and choosing **File** > **New** to open the New Gallery.
2. In the **Categories** tree, expand **General** and select **Projects**.
3. In the **Items** list, double-click **Empty Project** to open the New Project dialog.
4. Complete the New Project dialog and click **OK** to add the empty project to your workspace.
5. Select the new project in the Navigator and choose **File** > **New**.
6. In the **Categories** list, select **General** and select **XML**.
7. In the **Items** list, double-click XSQL Page.

   This adds a skeleton XSQL file named `untitled#.xsql` to your project.
8. In the Navigator, right-click the new XSQL file, and choose X**ML Editor** to open the source file.
9. Place your cursor in the blank line after the `<page xmlns:xsql="urn:oracle-xsql">` tag.
10. Choose **View** > **Component Palette** to open the Palette if it is not displayed.
11. Select **XSQL Tags** from the dropdown list in the Palette if it is not displayed.
12. Drag the **Query (XSQL)** tag from the Palette onto the XSQL file.

    The **Query** tag executes a SQL statement and includes its result set in XML format.
13. In the dialog that displays, you can enter values and change default values for the attributes. Press **F1** or click **Help** in the dialog to get help on the tag and its attributes.
14. After entering attributes, click **Next**.
15. In the Connection Selection dialog, select your connection or create a new database connection, then click **Next**.
16. In the Query dialog, type the SQL statement that you want to execute, then click **Next**.

    For example, you might type `select * from customer` to display all the records in the `customer` database, based on the attributes you entered.
17. Click **Finish**.

    Notice that the **Query** tag and attributes you entered appear in the XSQL page.
18. Choose **File** > **Save All** to save your work.

19. Right-click the XSQL file in the Navigator, and choose **Run** ***filename*.xsql** to view the raw XML data in your web browser.

You can format the XML data with a style sheet. The XML data also can be passed on to another application through a messaging service.

15.6.7 Creating XSQL Servlet Clients for Business Components

You can create XML based clients for business components using XSQL servlet. XSQL servlet allows you to easily get data in and out of the database in XML format. The following procedure shows how to bind an XSQL client to a business components project you have already created, using the `ViewObject Show` tag to display the view object's data in XML format. You could also use the `ViewObject Update` tag to process inserts, updates, and deletes to a view object.

To create an XSQL servlet client for business components:

1. Create a new project in the workspace that contains the business components project by selecting the workspace in the Navigator and choosing **File** > **New** to open the New Gallery.
2. In the Categories tree, expand **General** and select **Projects**.
3. In the Items list, double-click **Empty Project** to open the New Project dialog.
4. Complete the New Project dialog and click OK to add the empty project to your workspace.
5. Select the new project in the Navigator and choose **File** > **New**.
6. In the Categories list, select **General** and select **XML**.
7. In the **Items** list, double-click **XSQL Page.**

   This adds a skeleton XSQL file named untitled#.xsql to your project.
8. In the Navigator, right-click the new XSQL file, and choose XML Editor to open the source file if it is not open.
9. Place your cursor in the blank line after the `<page xmlns:xsql="urn:oracle-xsql">` tag.
10. Choose **View** > **Component Palette** to open the Palette if it is not displayed.
11. Select **XSQL tags** from the dropdown list in the Palette if it is not displayed.
12. Select the `ViewObject Show` tag from the Palette.

    The `ViewObject Show` tag shows the view object's data in XML format. The `ViewObject Update` processes inserts, updates, and deletes to a view object based on an optionally transformed XML document.
13. In the View Object Selection dialog, select the appropriate view object > click **Next.**
14. Change or accept the default values for the attributes. Press **F1** or click **Help** in the dialog to get help on the tag and its attributes. After entering attributes, click **Next.**
15. Click **Finish**.

    Notice that the tag and attributes you entered appear in the XSQL page.
16. Choose **File** > **Save All** to save all your work thus far.
17. Right-click the XSQL file in the Navigator, and choose **Run** ***filename*.xsql** to view the raw XML data in your web browser.

You can format the XML data with a style sheet. The XML data also can be passed on to another application through a messaging service.

Note: Please refer to the section titled "Caveats while Querying View Objects with Circular ViewLink Accessors" in the ViewObject Show F1 help topic if you get the XSQL error JBO-27122.

15.6.7.1 What You May Need to Know About Business Components XSQL Action Handlers

To use XSQL pages with the Business Components XSQL action handlers, the XSQL Runtime and the JBO HTML libraries need to be in your project's classpath, in addition to any JBO libraries that are needed based on your intended connection mode. JDeveloper includes them in the classpath automatically.

15.6.8 How to Creating a Custom Action Handler for XSQL

An action handler in an XSQL page is a Java class that gets invoked to perform a specific task. There are prebuilt action handlers for various tasks such as setting cookies, applying style sheets, performing queries against databases, etc. However, if you choose to perform some operation which is not provided by the built-in action handlers, then you can write what is called a custom action handler. A custom action handler is a Java class that can be invoked from an XSQL page just as easily as a predefined action handler.

To create an action handler:

1. Add the XSQL configuration file to your project.
2. In the XSQL configuration file, register the new action handler by specifying the element name and handler class.
3. In the XSQL file, add the new element and its attributes.
4. In the XSQL file, add connection information to the `<page>` tag.
5. Add a Java file to the project.
6. In the Java file, create a class that extends the `XSQLActionHandlerImpl` class.

The XSQL action handlers for BC4J are packaged as part of the JBO HTML library in JDeveloper, which includes the relevant: `<JdevHome>/BC4J/jlib/bc4jhtml.jar` archive in the build.

Example 15–4 Action Handler For XSQL

```
// Copyright (c) 2000, 2009, Oracle and/or its affiliates. All
 rights reserved. import oracle.xml.xsql.*;
import org.w3c.dom.Node;
import java.util.Date;
/**
 * A Class class.
 * <P>
 * @author Pas Apicella
public class JavaDate extends XSQLActionHandlerImpl
{
 public void handleAction (Node root)
   {
    addResultElement(root, "CURRENTDATE", (new Date()).toString());
    }
```

```
}
```

15.6.9 How to Run and Deploy XSQL Servlet Clients

After you have completed your XSQL file, you can test the XSQL query by running it in Integrated WebLogic Server, which provides everything you need to develop, test and debug web applications from within the IDE. For more information, see Section 9.2, "Running Java EE Applications in the Integrated Application Server."

To run an XSQL servlet file:

1. In the Navigator, right-click the XSQL file and select **Check XML Syntax**.

   JDeveloper will scan the XSQL file looking for XML syntax errors and display the results in the Log window.

2. If there are no XML syntax errors, right-click the XSQL file and select **Run *filename*.xsql**.

   JDeveloper compiles the servlet. It then starts the Integrated WebLogic Server. The first time you start Integrated WebLogic Server, a dialog is displayed where you have to enter a password for the default user `weblogic` on the default domain. You only need to do this once.

JDeveloper launches your default web browser, and displays the output of the servlet in the browser.

After the XSQL file has run successfully in Integrated WebLogic Server, you can deploy the application containing it to an external application server, such as Oracle WebLogic Server.

To deploy an XSQL application:

1. The syntax used by JDeveloper and Oracle WebLogic Server to run XSQL is different, so in your XSQL source file you have to change the connection information as follows. Replace:

   ```
   connection="java:comp/env/jdbc/database-connection-nameDS"
   ```

   with

   ```
   connection="jdbc/database-connection-nameDS"
   ```

   Note: If you want to run the application in the Integrated WebLogic Server, you need to change the connection information back again.

2. In order to deploy the application, you first have to create a deployment profile and deploy the application to it. In the navigator, right-click the project containing your XSQL servlet, then choose **New**. In the New Gallery, expand **General** and select **Deployment Profiles**.

3. Choose a profile, for example, a WAR deployment profile and click **OK** and continue to create the deployment profile. For more information, see Section 9.3.2, "How to Create and Edit Deployment Profiles."

4. To deploy the application to the deployment profile, right-click on the project containing your XSQL servlet files and choose **Deploy** > ***profile*** where ***profile*** is the name of the deployment profile you just created.

In the Deployment dialog, choose **Deploy to WAR** (or the appropriate option if you have chosen a different type of deployment profile) and click **Finish**.

5. The application is now ready to deploy to an application server, for example, Oracle WebLogic Server. The steps you need to perform are:

   - Create a data source on the target application server using the connection information in the XSQL file. For more information, see Section 9.3.6.4, "Setting Up JDBC Data Sources on Oracle WebLogic Server."
   - Create a connection to the application server. For more information, see Section 9.3.1, "How to Create a Connection to the Target Application Server."
   - Deploy the application by right-clicking on the project containing your XSQL servlet files and choosing **Deploy** > ***profile*** where ***profile*** is the name of the deployment profile.

     In the Deployment dialog, choose **Deploy to application server** and on the next page choose the application server connection and click **Finish**.

   Once the application is deployed, you can view the results of the query in a browser window by navigating to `http://targethost:port/web-context-root/filename.xsql`.

15.6.10 How to View Output from Running XSQL Files as Raw XML Data

After creating an XSQL file and adding tags, you can view the raw XML data or format the XML data with a style sheet.

To view an XSQL file as raw XML data:

Select the XSQL file in the Navigator, right-click and choose **Run *filename*.xsql** to open the source file in your web browser.

JDeveloper starts the Integrated WebLogic Server, launches your default web browser, and displays the raw XML data that is produced after the XSQL servlet processes the XSQL page.

15.6.11 How to Format XML Data with a Style Sheet

After creating an XSQL file and adding tags, you can format the XML data with an XSL style sheet or view the raw XML data. You can use a style sheet you previously created or create a new one in JDeveloper and apply it. By applying a style sheet, you can convert the XML data into HTML or another markup language, such as wireless markup language (WML).

To format the XML data with a style sheet:

1. In the Navigator, select the XSQL file to which you want to add a style sheet, right-click and choose **XML Editor** to open the source file.
2. Locate the xml-stylesheet line and comment, which looks like this:

```
<!--
Uncomment the following processing instruction and replace
the stylesheet name to transform output of your XSQL Page using XSLT
<?xml-stylesheet type="text/xsl" href="YourStylesheet.xsl" ?>
-->
```

3. Uncomment the `<?xml-stylesheet?>` line by moving it below the `-->` closing comment bracket.

4. In this line, replace `YourStyleSheet.xsl` with the name of your style sheet; for example, your style sheet could be named `stylesheet1.xsl`.

   Next, add the file that you just specified to your project, if you used one created outside of this project.

5. In the Navigator, select the project and choose **Project** > **Add to Project** project name. In the Add to Project dialog, navigate to the directory and select the style sheet file you specified.

6. Click **Open**.

7. Choose **File** > **Save All** to save all your changes.

   The file you added displays in the Navigator and opens in the XML Editor. You can close the open files.

8. Select the XSQL file in the Navigator, right-click and choose **Run** ***filename*.xsql** to open the file in your web browser.

   You can see the formatted XML data in the browser.

15.6.12 How to Create an XSL Style Sheet for XSQL Files

In JDeveloper, you can create an XSL style sheet that you can apply to your XSQL files in order to format the data for HTML, WML or another output. When you create an XSL style sheet, it is added to the selected XSQL project.

To create an XSL style sheet:

1. In the Navigator, select the project in which you want to create the new XSL file.

2. Choose **File** > **New** to open the New Gallery.

3. In the **Categories** tree, expand **General** and select **XML**.

4. In the **Items** list, double-click **XSL Style Sheet** to open the New XSL File dialog.

5. Leave the **Directory Name** field unchanged to save your work in the directory where JDeveloper expects to find web application files. In the **File Name** field, enter the name of the file you want to generate.

   A skeleton XSL file is generated and appears in your active project.

You can edit it in the XML Editor to create your own custom style sheet. An example of an XSL style sheet that transforms XML data into wireless markup language (WML) is provided below. When you are finished, you can specify the style sheet name in your XSQL file to format the raw XML data.

XSL Style Sheet Example

The style sheet in Example 15–5 demonstrates the conversion of XML to WML. It uses the default DeptView in a BC4J application.

Example 15–5 Conversion of XML to WML

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Root template -->
<xsl:output type="wml" media-type="text/x-wap.wml"
doctype-public="-//WAPFORUM//DTD WML 1.1//EN"
doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"
indent="yes" />
```

```
<xsl:template match="* >/"><xsl:apply-templates/></xsl:template>
<xsl:template match="text()>@*"><xsl:value-of select="."/></xsl:template>
<xsl:template match="/">

<wml>
   <card id="C1">
   <p mode="nowrap">
   <big>DEPTLIST</big>
   </p>
   <xsl:for-each select="page/DeptView/DeptViewRow">
   <p>
   <strong><xsl:value-of select="Deptno"/> </strong>
   xsl:value-of select="Dname"/> 
   <xsl:value-of select="Loc"/>
   </p>
   </xsl:for-each>
   </card>
</wml>

</xsl:template>
</xsl:stylesheet>
```

15.6.13 How to Modify the XSQL Configuration File

The XSQL configuration file, XSQLConfig.xml, is on the classpath, so your XSQL pages always have access to it. The connection information is added to the XSQLConfig.xml file when you create a new connection in JDeveloper. XSQLConfig.xml is located in the system directory and gets copied to the WEB-INF directory when a project containing an XSQL file is compiled. You can add the file to your project if you need to modify it; for example, to register custom action handlers.

Note: When you migrate an XSQL project in JDeveloper, the XSQLConfig.xml file is not updated for you. You can update your connections after migrating the project by recreating the connection or editing an existing connection in JDeveloper.

To modify the XSQL configuration file for your project:

1. With the project selected in the Navigator, choose **Project** > **Add to Project** ***project name***.
2. Navigate to the system directory in your JDeveloper installation directory, select `XSQLConfig.xml` and click **Open.**
3. Make any changes or additions in the XML Editor.
4. Choose **File** > **Save** to save your revised file.

15.6.14 Using XML Metadata Properties in XSQL Files

The custom properties shown in Table 15–4 affect XML generation when using the writeXML method of a view object or row.

Table 15–4 Metadata Properties

Property Name	Value	Valid For
XML_ELEMENT	a legal element name	view objects and view attributes
XML_ROW_ELEMENT	a legal element name	view objects
XML_CDATA	any value (not empty)	view attributes
XML_EXPLICIT_NULL	any value (not empty)	view objects and view attributes

15.6.14.1 Using XML_ELEMENT

If the XML_ELEMENT custom property is present for a view object, its value is used as the XML element name for the view object in XML, when it is generated using the writeXML method and "consumed" by the readXML method.

If the XML_ELEMENT custom property is present for a view attribute, its value is used as the XML element name for the attribute in XML, when it is generated using the writeXML method and "consumed" by the readXML method.

For example, for a view object named DeptView with an attribute named Sal, setting:

- XML_ELEMENT="Departments" in the view object properties
- XML_ELEMENT="Salary" in the view attribute properties for Sal

would produce XML like:

```
<Departments>
  <DeptViewRow>
    <Empno>1010</Empno>
    <Ename>Steve</Ename>
    <Salary>1234</Salary>
  </DeptViewRow>
</Departments>
```

Instead of the default:

```
<DeptView>
  <DeptViewRow>
    <Empno>1010</Empno>
    <Ename>Steve</Ename>
    <Sal>1234</Sal>
  </DeptViewRow>
</DeptView>
```

15.6.14.2 Using XML_ROW_ELEMENT

If the XML_ROW_ELEMENT custom property is present for a view object, its value is used as the XML element name for each row of query results produced by the view object in XML, when it is generated using the writeXML method and "consumed" by the readXML method.

For example, for a view object named DeptView with an attribute named Sal, setting:

- XML_ELEMENT="Departments" in the view object properties
- XML_ROW_ELEMENT="Department" in the view object properties
- XML_ELEMENT="Salary" in the view attribute properties for Sal

would produce XML like:

```
<Departments>
  <Department>
    <Empno>1010</Empno>
    <Ename>Steve</Ename>
    <Salary>1234</Salary>
  </Department>
</Departments>
```

instead of the default:

```
<DeptView>
  <DeptViewRow>
    <Empno>1010</Empno>
    <Ename>Steve</Ename>
    <Sal>1234</Sal>
  </DeptViewRow>
</DeptView>
```

15.6.14.3 Using XML_CDATA

If the XML_CDATA custom property is set to a not empty value for a view attribute, then its value will be output as a CDATA section instead of as plain text.

15.6.14.4 Using XML_EXPLICIT_NULL

If the XML_EXPLICIT_NULL custom property is set to a not empty value for a view object, then any attribute with a null value will generate an XML element that looks like:

```
<AttributeName null="true"/>
```

instead of omitting the <AttributeName> element from the XML result, which is the default.

If the XML_EXPLICIT_NULL custom property is set to a not empty value for a view attribute, then in the case that the indicated attribute has a null value, the system will generate an XML element that looks like:

```
<AttributeName null="true"/>
```

instead of omitting the `<AttributeName>` element from the XML result, which is the default.

16

Developing Applications Using Web Services

This chapter describes how JDeveloper provides powerful tools that help you discover and use existing web services, and develop and deploy new web services.

This chapter includes the following sections:

16.1 About Developing Applications using Web Services

Web services consist of a set of messaging protocols and programming standards that expose business functions over the Internet using open standards. A web service is a discrete, reusable software component that is accessed programmatically over the Internet to return a response.

You can create web service clients to access existing web services. If you use web services in your application, you can create bottom-up (starting from Java) and top-down (starting from WSDL) web services as follows:

- Configure JDeveloper to develop and run web services
- Create web service clients by performing one or more of the following tasks:
  - Find web services in a UDDI registry

  - Create a client and proxy classes to access an existing web service to incorporate it into an application
- Create web services by performing one or more of the following tasks:
  - Create SOAP web services from the underlying Java implementation (bottom up)
  - Create web services from the WSDL (top down).
  - Create RESTful web services.
- Secure web services using policies
- Test and debug web services
- Deploy web services to the Integrated WebLogic Server or Oracle WebLogic Server
- Publish web services to a UDDI registry

Once deployed, your web services can then be accessed and used in other applications.

16.1.1 Discovering and Using Web Services

You can quickly create a client to an existing web service in order to use it in your application. You can view all web services in the application under the Web Services folder in the Application Navigator.

In addition, JDeveloper incorporates a UDDI browser and you can define connections to UDDI registries, for example, to one within your organization. For more information, see Section 16.3, "Working with Web Services in a UDDI Registry".

16.1.2 Developing and Deploying Web Services

You can create web services from Java classes, the remote interface of EJBs, and an ADF Business Components service session bean wrapped as an EJB. The Web service creation wizards create the deployment files for you, so once you have created your web service the final step is to deploy it to application servers. For more information, see Section 16.5, "Creating SOAP Web Services (Bottom-Up)".

Alternatively, you can create a web service starting with a WSDL, as a top-down web service. For more information, see Section 16.6, "Creating SOAP Web Services from WSDL (Top Down)".

Finally, you can develop web services that are based on Representational State Transfer (REST). A RESTful web service is a simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. For more information, see Section 16.7, "Creating RESTful Web Services"

JDeveloper also supports a set of standard Java-to-XML type mappings. You can also create custom serializers for types of objects that are not automatically supported. For more information, see Section 16.2, "Using JDeveloper to Create and Use Web Services".

16.2 Using JDeveloper to Create and Use Web Services

This following information will help you understand more about web services, and how you can use JDeveloper to create, configure, and use them.

- Section 16.2.1, "How to Use Proxy Settings and JDeveloper"

16.2.1 How to Use Proxy Settings and JDeveloper

By default, JDeveloper uses the proxy settings from the default browser on the same machine. If you have problems making connections from JDeveloper, for example, connecting to an application server that is on the same machine as JDeveloper, you may need to change the proxy server settings you use.

For example, if you are connecting to an IP address behind a proxy server, and your machine is also behind the same proxy server, then make sure that the web proxy preferences exclude the IP address you are trying to connect to.

When you use the HTTP Analyzer, the analyzer itself is a proxy and any traffic to be monitored by it is routed through it, just as though it was a normal proxy server. If you already have a proxy set in JDeveloper, the analyzer will make sure that the traffic goes through the original proxy after it has been passed through the analyzer.

To exclude an IP address:

1. Choose **Tool > Preferences**, and select **Web Browser and Proxy**.

   For more information at any time, click F1 or **Help** from the Web Browser and Proxy dialog.

2. Add the IP address to the **Exceptions** list.

To turn off use of the browser proxy server:

1. Choose **Tool > Preferences**, and select **Web Browser and Proxy**.

   For more information at any time, click F1 or **Help** from the Web Browser and Proxy dialog.

2. Deselect **Use HTTP Proxy Server**.

16.2.2 How to Set the Context Root for Web Services

The context root appears as part of the web service endpoint for a generated web service, so it is important that it is set to an appropriate value. You set the context root at the project level.

The web service context root is the string that comes after the `host:port` portion of the web service URL. For example, if the deployed WSDL of a WebLogic web service is as follows: `http://hostname:7001/financial/GetQuote?WSDL`

The context path for this web service is `financial`.

To set the context root:

1. In the Application Navigator, right-click the project and choose **Project Properties** to open the Project Properties dialog.

For more information at any time, click F1 or **Help** from the Project Properties dialog.

2. Expand **Project Source Paths** and select **Web Application**.
3. Either accept the default **HTML Root Directory** or enter a new value.

   Click **Browse** to browse the local directory.

16.2.3 How to Configure Connections to Use with Web Services

You can develop simple web services that you can test using the Integrated WebLogic Server. However, to develop more complex web services, and to deploy web services, you will need the appropriate connections.

- To deploy a web service to Oracle WebLogic Server, you need an application server connection as described in Section 16.12, "Deploying Web Services".
- To find web services using a Universal Description, Discovery and Integration (UDDI) registry, you need to create a connection to the registry. For more information, see Section 16.3.1, "How to Define UDDI Registry Connections".

16.2.4 How to Work with Type Mappings

Objects that can be passed to and from web services have to be able to be serialized to an XML type, and then deserialized back to their original type. Objects that are automatically handled are Java primitive types and certain Java standard types. If you want to create a web service using objects that are not automatically serialized, you can write your own custom serializer.

The objects that can be passed to and from web services are ones that conform to the JavaBean conventions For the purposes of web services, a JavaBean is any Java class that conforms to the following restrictions:

- Must have a public default (zero argument) constructor.
- Must expose all attributes of interest as accessors.
- Order of the accessors for the properties (`setMethod()` and `getMethod()`) must not matter.
- Accessors must be written in mixed case with a lower case first letter. For example, if an attribute is called `name` the accessors must be called `getName` and `setName`.

For more information, refer to the JavaBean spec at `http://java.sun.com/javase/technologies/desktop/javabeans/api/index.html`.

For web services, each property of the object must be of one of the Java types that maps to an XML schema simple type. These are listed in the table below, which shows the primitive XML Schema types and arrays of primitive XML Schema types that are supported as parameters, and the return values for web services. In addition, a service method can accept and return a single piece of XML element data, passed as an `org.w3c.dom.Element`.

Table 16–1 XML schema type mapping to Java types

XML Schema type	Java type
`string`	`java.lang.String`
`boolean`	`java.lang.Boolean`

Table 16–1 (Cont.) XML schema type mapping to Java types

XML Schema type	Java type
decima	java.lang.Double
float	java.lang.Float
double	java.lang.Double
dateTime	java.util.Date
time	java.util.Date
date	date java.util.GregorianCalendar
base64Binary	java.lang.Byte[]
normalizedString	java.lang.String
integer	java.lang.Integer
long	java.lang.Long
int	java.lang.Integer
short	java.lang.Short
byte	java.lang.Byte

JAX-WS web services use Java Architecture for XML Binding (JAXB), described at http://jcp.org/en/jsr/detail?id=222, to manage all of the data binding tasks. Specifically, JAXB binds Java method signatures and WSDL messages and operations and allows you to customize the mapping while automatically handling the runtime conversion. This makes it easy for you to incorporate XML data and processing functions in applications based on Java technology without having to know much about XML.

16.2.5 How to Work with PL/SQL Web Services and Types

This section describes the limitations for which PL/SQL web services cannot be created for a particular circumstance:

- Overloaded Program Units
- BFILE Type
- BCLOB Type
- OUT and IN-OUT Parameters
- Creating PL/SQL web services from PL/SQL records
- Stored procedures of the same name which are accessible in more than one schema
- Ref Cursors Return Types
- SYS Schema
- Types Declared Within a Package Spec
- PL/SQL nested tables

Overloaded Program Units

A program unit that shares its name with another program unit in the same package is an overloaded program unit. At runtime the WSDL processor cannot determine which program unit to execute when there is more than one program unit with the same name. Therefore, the PL/SQL program units cannot be deployed as web services.

You can avoid the problem of overloaded program units that you can adapt to suit your requirements. Consider the following example of a PL/SQL package containing the following program:

```
-- promotes an employee to the specified rank
PROCEDURE promote_emp(empno IN NUMBER, rank IN NUMBER);

-- promotes an employee to the rank above their current rank
PROCEDURE promote_emp(empno IN NUMBER);
```

You can workaround the overloaded types in one of the following ways:

- If you are able to change the existing package, you can add the two procedures shown below to the package, and publish the web services from the new procedures.

  or

- You can add the two procedures shown below to a new package, and publish the web services from the new package.

The new procedures are:

```
-- promotes an employee to the specified rank
PROCEDURE promote_emp_to_rank(empno IN NUMBER, rank IN NUMBER)
IS
BEGIN
promote_emp(empno, rank);
END;

-- promotes an employee to the rank above their current rank
PROCEDURE promote_emp_to_next_rank(empno IN NUMBER) IS
BEGIN
promote_emp(empno);
END;
```

BFILE Type

The PL/SQL type BFILE can only be used as an OUT argument or as a function return value.

BCLOB Type

The PL/SQL type BCLOB is not supported.

OUT and IN-OUT Parameters

When you publish a program unit with OUT or IN-OUT parameters, these are transferred back to the caller in a return type structure with one attribute for each OUT or IN-OUT parameter. For example, a service with the following signature:

```
PROCEDURE a_proc(val1 IN VARCHAR2, val2 IN OUT NUMBER, val3 OUT INTEGER)
```

Returns the final values of `val2` and `val3` in a generated result class. You can use accessor methods on the generated class to access these values.

Creating PL/SQL web services from PL/SQL records

JDeveloper does not allow you to create web services directly from PL/SQL packages that use PL/SQL records. If your organization uses PL/SQL packages that have been migrated from earlier versions of the Oracle database, you may find that you want to

expose some functionality as web services and be unable to do so because the packages accept and return parameters that are record types, rather than object types.

You can use Oracle JPublisher on the packages that contain record types. For more information, see Section 28.3.1, "How to Use JPublisher".

A SQL file is produced that you run against your database to create equivalent packages that contain object types. You can then use the new packages to create your PL/SQL web services in the usual manner.

Stored procedures of the same name which are accessible in more than one schema

On Oracle9i Database release 2, when a stored procedure or function of the same name and the same package name is accessible in more than one schema, then the SQLJ translator invoked during publication of PL/SQL web services will fail.

In order to resolve this problem, ensure that packages to be published are visible only in one schema, and that no other packages in other schemas share the same name.

Ref Cursors Return Types

You cannot create a web service from a packages that uses ref cursor as a return type, for example:

```
PACKAGE TEST AS
type EmpCurType is ref cursor;
function EmpData return TEST.EmpCurType;
END;
```

SYS Schema

In order to prevent an arbitrary user from assuming SYS privilege, a connection cannot be specified from the middle tier as SYS. This means that you cannot create a web service from a package in the SYS schema.

If you need to access a PL/SQL package in the SYS schema from the middle tier, for example, to create a web service in JDeveloper, log on to the database as SYS, and grant package EXECUTE privileges to the user you then use to create the JDeveloper database connection.

Types Declared Within a Package Spec

PL/SQL packages can have types declared within the package spec, however these packages cannot be published as web services. To avoid this, create the types outside the scope of the package.

PL/SQL nested tables

JDeveloper does not allow you to create web services directly from PL/SQL packages that use PL/SQL nested tables.

16.2.6 How to Choose Your Deployment Platform

When you create a web service using the web services wizards, you are offered a choice of deployment platforms, as defined in Table 16–2. The platform you choose determines the options available to you in the wizard, and the libraries that are added to the WAR/EAR file for deployment.

Table 16–2 Deployment Platforms

Deployment Platform	Description
J2EE 1.4 JAX-RPC with support for WebLogic Server 10.3	Generates a JAX-RPC web service that is configured for deploying to Oracle WebLogic Server 10.3.
Java EE 1.5 with support for JAX-WS Annotations	Generates a web service that takes advantage of the JAX-WS web services API, released as part of Java EE 1. 5. This option provides support for deploying to WebLogic Server 10.3 with Java annotations using the JAX-WS annotation specification.
Java EE 1.5 with support for JAX-WS RI	Generates a JAX-WS web service for deploying to any container that supports the Sun JAX-WS Reference Implementation.

16.2.7 How to Work with Web Services Code Insight

The web services Code Insight completes annotations when typing in a Java class, and is available for WSDL documents in the XML editor (that is, when typing in the Source tab). You can configure how fast Code Insight responds. You can access the Code Insight page in JDeveloper from **Tools menu > Preferences > Code Editor > Code Insight**.

> **Note:** Code Insight does not work with Java classes for JAX-RPC web services, only with Java classes for JAX-WS web services.

When you create a JAX-WS web service from a Java class by adding annotations in the source editor, the Code Insight features of Quick Fixes and Code Assists are available to help you.

For example, when you create a web service from a Java class by manually adding the @WebService annotation, a ragged line appears under the annotation. Click the Audit Fix icon and choose **Configure project for web services**.

From the Select Deployment Platform dialog, select one of the following JAX-WS platforms for your service:

- Java EE 1.5, with support for JAX-WS Annotations. In this case, JDeveloper adds:
  - `import javax.jws.WebService;` statement to the class
  - web.xml file to the project
- Java EE 1.5, with support for JAX-WS RI. In this case, JDevleoper adds:
  - `import javax.jws.WebService;` statement to the class
  - sun-jaxws.xml and web.xml files to the project

Other examples include:

- You can add policy annotations to a JAX-WS web service and use JDeveloper to complete the policy you want. For example, if you enter `@Pol`, then click **Alt+Enter** you can choose whether to use `@Policy`, for a single policy, or `@Policies` for multiple policies. The appropriate import statement is also added to the class.
- If you are working on a WSDL document in the source editor, you can use code completion to help you enter schema elements. For example, if you enter < and wait a second, a popup appears from which you can select the entry you want.
- If the WSDL and web service source files get out-of-sync, you can regenerate the web service from source.

- If you rename a Java class in either the Java class or WSDL, click the Audit fix icon and select how you would like to reconcile the discrepancy.

16.2.8 How to Migrate JAX-RPC 10.1.3 Web Services

You can migrate web services created as J2EE 1.4 JAX-RPC web services in JDeveloper 10.1.3.*n*.

> **Note:** You cannot migrate EJB 2.1 web services because they are deprecated in this version of JDeveloper. You cannot migrate JMS web services because they are Oracle-proprietary and no longer supported.

The following actions are performed during the upgrade:

- Remove the JAX-RPC 10.1.3 Web Services library and replace it with the JAX-RPC 11g Web Services library.
- Regenerate the web service as an Oracle WebLogic Server compatible JAX-RPC web service. For more information, see Section 16.5.6, "How to Regenerate Web Services from Source".

The WSDL is not changed. After performing the upgrade, you can use the Edit Web Service dialog to make any additional changes.

The limitations of the migrated service are:

- JAX-RPC web services in this version of JDeveloper only support SOAP 1.1, so support for SOAP 1.2 is removed.
- Any attached policies are removed. You can attach new policies by editing the web service and applying the appropriate Oracle WebLogic Server policies. For more information, see Section 16.9.3, "How to Attach Policies to Web Services".

> **Note:** JAX-RPC web services can only use Oracle WebLogic Server security policies.

- Support for the previous version of stateful services is removed. You can configure support for the current version of stateful services in the Edit Web Services dialog. For more information, see Section 16.10, "Editing and Deleting Web Services".
- Support for REST is removed.
- Methods with collection return type, which are not supported, are disabled in the implementation.

To migrate a web service:

1. In the Application Navigator, select **Open Application** from the dropdown list. Open the application that contains the web service.

   The migration wizard is automatically invoked enabling you to migrate the application and projects. For more information at any time, press F1 or click **Help** from within the wizard.

2. On the Webapp 2.5 Migration page of the Migration Wizard, deselect **Migrate to Webapp 2.5**. JAX-RPC is compatible with version 2.4 of web.xml, not version 2.5.

3. To upgrade the web service, in the Application Navigator, right-click the web service container and choose **Upgrade Web Service to WLS JAX-RPC Configuration**.
4. Read the Confirm Upgrade message and click **Yes**. The web service is upgraded to use the JAX-RPC 11*g* Web Services library and regenerated.
5. If necessary, edit the web service to make any additional changes.

16.3 Working with Web Services in a UDDI Registry

Universal Description, Discovery and Integration (UDDI) is one of the standards and protocols that underpin web services. It provides a common standard for publishing and discovering information about web services. It contains a UDDI browser that searches a UDDI registry using search criteria that you specify to find web services that are described by Web Services Description Language (WSDL). For more information about UDDI including the specification, see the UDDI OASIS standards at `http://uddi.xml.org/`.

The following sections describe how to work with web services in a UDDI registry:

- Section 16.3.1, "How to Define UDDI Registry Connections"
- Section 16.3.2, "How to Configure the View of UDDI Registry Connections"
- Section 16.3.3, "How to Search for Web Services in a UDDI Registry"
- Section 16.3.4, "How to Generate Proxies to Use Web Services Located in a UDDI Registry"
- Section 16.3.5, "How to Display Reports of Web Services Located in a UDDI Registry"
- Section 16.3.6, "How to Publish Web Services to a UDDI Registry"

16.3.1 How to Define UDDI Registry Connections

You can define connections to UDDI registries, for example, to browse your organization's internal UDDI registry. In addition, all defined UDDI registry connections are accessible to any workspace or project.

For more information about UDDI including the specification, see the UDDI OASIS standards at `http://uddi.xml.org/`.

The following sections describe how to define UDDI registry connections.

- Section 16.3.1.1, "Creating UDDI Registry Connections"
- Section 16.3.1.2, "Editing the Name of UDDI Registry Connections"
- Section 16.3.1.3, "Changing the View of UDDI Registry Connections"
- Section 16.3.1.4, "Refreshing UDDI Registry Connections"
- Section 16.3.1.5, "Deleting UDDI Registry Connections"

16.3.1.1 Creating UDDI Registry Connections

You can create a new connection to a UDDI registry that is public or private (within your organization). The UDDI registry connection is listed in the Resource Palette, in the Connections panel.

To create a new connection:

1. In the Application Navigator, select the project.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Business Tier** and select **Web Services**.
4. In the **Items** list, double-click **UDDI Registry Connection** to launch the Create UDDI Registry Connection wizard.

   For more information at any time, press F1 or click **Help** from within the Create UDDI Registry Connection wizard.

Alternatively, you can create the connection directly in the Resource Palette as described in Section 6.9, "Using WebDAV with JDeveloper".

16.3.1.2 Editing the Name of UDDI Registry Connections

You can edit an existing UDDI registry connection to change the name of the connection, or to change the URL of the inquiry endpoint.

To change the inquiry endpoint of a registry:

1. In the main menu, choose **View > Resource Palette**. By default, the Resource Palette is displayed to the right of the JDeveloper window.
2. In the Resource Palette, expand **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to edit, choose **Properties**.

   The reentrant UDDI Registry Connection wizard is launched.

   For more information at any time, press F1 or click **Help** from within the Create UDDI Registry Connection wizard.

16.3.1.3 Changing the View of UDDI Registry Connections

You can change the order that web services are listed in the UDDI registry from Category view to Business view, or from Business View to Category view. For more information, see Section 16.3.2, "How to Configure the View of UDDI Registry Connections".

To change the view of a registry:

1. In the main menu, choose **View > Resource Palette**. By default, the Resource Palette is displayed to the right of the JDeveloper window.
2. In the Resource Palette, expand the **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to edit, choose **Render Business Perspective** or **Render Category Perspective**.

16.3.1.4 Refreshing UDDI Registry Connections

You can refresh a UDDI registry connection to ensure that information stored under the connection is up to date.

To refresh a connection:

1. In the main menu, choose **View > Resource Palette**. By default, the Resource Palette is displayed to the right of the JDeveloper window.
2. In the Resource Palette, expand the **UDDI Registry**.

3. From the context menu of the UDDI registry connection you want, choose **Refresh**.

16.3.1.5 Deleting UDDI Registry Connections

When no longer needed, you can delete a UDDI registry connection from the Resource Palette.

To delete a connection:

1. In the main menu, choose **View > Resource Palette**. By default, the Resource Palette is displayed to the right of the JDeveloper window.
2. In the Resource Palette, expand the **UDDI Registry**.
3. From the context menu of the UDDI registry connection you want to delete, choose **Delete**.
4. A message is displayed asking whether you want to delete the connection. Click **Yes**.

16.3.2 How to Configure the View of UDDI Registry Connections

When you create the connection, as described in Section 16.3.1, "How to Define UDDI Registry Connections", you are prompted whether the web services in the registry are displayed in Business View or Category View. The view you choose will determine how you search for services in the registry.

16.3.2.1 Choosing Business View

A UDDI registry contains four data structure types that group information about web services:

- **businessEntity**: Defines the top-level data structure that contains information about the business providing the web service. When you find a web service, the business is added to the UDDI browser in the Resource Palette.
- **businessService**: Contains descriptive information for a family of services, including the name and brief description, and category information.
- **bindingTemplate**: Contains information about a web service entry point and references to interface specification.
- **tModel**: Represents the technical specification of the web service. When the Find Web Services wizard finds a web service, it also displays other web services that are compatible with the same tModel.

If you choose Business View, services are listed under Business Entities and Business Services.

16.3.2.2 Choosing Category View

If you choose Category View, you can search for web services based on one or more of the following categories:

- **UDDI Types**: Search by UDDI type.
- **NAICS**: Specify the type of industry.
- **ISO 3166**: Search by location.
- **UNSPSC**: Search by type of service.

When you search by name, you can enter all or part of a name and you can use wildcards. The results are tModels where the name of the tModel matches the search criteria. When a number of web services have the same tModel, they are listed in the wizard so that you can choose the one that best fits your requirements.

16.3.3 How to Search for Web Services in a UDDI Registry

You can search a UDDI registry connection in the Resource Palette for a web service.

> **Note:** If you are creating a top-down web service, you can use the Find Web Service Wizard to search a UDDI registry connection from within the Create Java Web Service from WSDL wizard.

To search for a web service in a UDDI Registry:

1. Create a UDDI registry connection, if required. For more information, see Section 16.3.1, "How to Define UDDI Registry Connections".
2. In the Resource Palette, search for the web service. For more information, see Section 3.7, "Working with the Resource Palette".

16.3.4 How to Generate Proxies to Use Web Services Located in a UDDI Registry

You can create a proxy to a web service in a UDDI registry connection in the Resource Palette.

> **Note:** You can only generate a proxy to a web service if the service uses a WSDL link. To determine this, open the web service report, and check that the Overview Description in the tModel Instances section of the report is `wsdl link`.

To generate a proxy:

1. Open the Resource Palette.

   In the main menu, choose **View > Resource Palette**. By default, the Resource Palette is displayed to the right of the JDeveloper window.
2. Navigate to the web service you want, or search for it.
3. Navigate to the service
4. Right-click the service, and choose **Generate Web Service Proxy** to launch the Web Service Proxy wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.

16.3.5 How to Display Reports of Web Services Located in a UDDI Registry

You can display a report of a web service in a UDDI registry.

To display a report of the service:

1. In the Resource Palette, expand the UDDI registry connection, and navigate to the endpoint for the service.
2. Right-click the service, and choose **View Report**.

   A report of the web service is displayed in the source editor.

16.3.6 How to Publish Web Services to a UDDI Registry

You can publish a web service to a UDDI registry through a connection to the registry in the Application Server navigator. Before you can publish a service to a UDDI registry, you must already have a connection to the registry in the Resource Catalog. For more information, see Section 16.3.1.1, "Creating UDDI Registry Connections".

To publish a web service to a UDDI registry:

1. Deploy the web service to Oracle WebLogic Server.

   Note: If you deploy the web service to the Integrated WebLogic Server, then the UDDI registry to which you are publishing must be local to the Integrated WebLogic Server.

2. In Application Server navigator, expand the application server node.
3. Expand the web services node and locate the node (which represents the WSDL) of the web service you want to publish.
4. Right-click the WSDL node and choose Publish WSDL to UDDI to launch the **Publish WSDL to UDDI Registry** dialog.

   For more information at any time, press F1 or click **Help** in the Publish WSDL to UDDI Registry dialog.

16.4 Creating Web Service Clients

JDeveloper makes it easy to use a web service in your application by allowing you to create client and proxy classes to access the service using the Create Web Service Client and Proxy wizard. You can launch the wizard when you locate or create a web service. Alternatively, you can launch the wizard directly and enter the URL for the web service or use the Find Web Service wizard to locate a web service in a UDDI registry.

JDeveloper automatically generates the correct type of proxy for an RPC or document style web service.

Note: JAX-WS web services do not support RPC style.

For more information about:

- Developing web service clients, see "Invoking Web Services" in the *Oracle Fusion Middleware Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.
- Securing and administering web services and clients, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

The following sections describe how to create and use web service clients:

- Section 16.4.1, "How to Create the Client and Proxy Classes"
- Section 16.4.2, "How to Use Web Service Client and Proxy Classes"
- Section 16.4.3, "How to View the WSDL Used to Create the Web Service Client"
- Section 16.4.4, "How to Update the Web Service WSDL at Run Time"
- Section 16.4.5, "How to Regenerate Web Service Client and Proxy Classes"
- Section 16.4.6, "How to Manage the Web Service Clients"

- Section 16.4.7, "How to Reference Web Services Using the @WebServiceRef Annotation"

16.4.1 How to Create the Client and Proxy Classes

Use JDeveloper to automatically create a client and proxy classes to access a web service and call its methods in your application. Using the wizard, you can also generate asynchronous methods and attach policies, as required.

You can create a client and proxy classes to access a web service using the Create Web Service Client and Proxy wizard. The wizard generates a new service class (JAX-WS) or stub (JAX-RPC) and service interface for each exposed port and lists them in the Application Navigator. It opens the generated client file `port-nameClient.java` in the source editor. Once generated, you can call the methods in your application.

> **Note:** In some cases, you may encounter errors when you run a web service client that you have created for a web service accessed on the Internet or using a UDDI registry. Because web services standards are still evolving, it is possible that the web services that you locate may not conform to the latest standards, or the standards to which they conform may not be compatible with those supported by the server on which the client is running. If a web service client that you have created in JDeveloper returns an error, examine the error message and consider creating a client to another web service that provides a similar service, but that is compatible with the server and will run without problems.

You can access the Create Web Service Client and Proxy wizard using one of the following methods. For help in completing the wizard, press F1 or click **Help** from within the wizard.

To create a client and proxy classes to access a web service:

1. In the Application Navigator, select the project you want to use.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, select **Web Services**.
4. In the **Items** list, double-click **Web Services** to launch t the Create Web Service Client and Proxy wizard.

   For more information at any time, press F1 or click **Help** from within the Create Web Service Client and Proxy wizard.

To create a client and proxy classes to access a web service defined in JDeveloper:

1. Right-click the web service container in the Application Navigator, and choose:
   - For a JAX-WS web service, **Create Client for Web Service Annotations**.
   - For a JAX-RPC web service, **Generate Web Service Proxy**.
2. The Create Web Service Client and Proxy wizard opens and is prepopulated with the selected web service project.

Note: When you create the client and proxy classes for an EJB web service that uses JavaBean parameters, the JavaBean must implement the `java.io.Serializable` interface.

16.4.2 How to Use Web Service Client and Proxy Classes

JDeveloper generates a number of files that define a proxy to the web service. Using the generated files, you can develop the following types of web service client applications:

- Stand-alone client application
- Java Standard Edition (SE) client application
- Java EE component deployed to Oracle WebLogic Server

Note: In addition to the procedures described below, you can use web service injection (using the `@WebServiceRef` method) to define a reference to a web service and identify an injection target in your web service client. For more information see Section 16.4.7, "How to Reference Web Services Using the @WebServiceRef Annotation"

16.4.2.1 How to Use a Stand-Alone Client Application

A stand-alone client application, in its simplest form, is a Java program that has the Main public class that you invoke with the java command. It runs completely separate from WebLogic Server.

To use the generated client proxy classes in a stand-alone client:

1. Open the client proxy class, called `port_nameClient.java`, in the source editor.

   This file opens automatically when you create the web service client proxy initially. To re-open the class, right-click on the client proxy container and select **Go to Client Class** or simply double-click on the file in the Application Navigator.

2. Locate the comment `// Add your own code here`, which is in a try-catch block in the main method, and add the appropriate code to invoke the web service.

3. Run the client.

16.4.2.2 How to Use the Java Standard Edition (SE) Client Application

Include the generated proxy classes as part of a Java Standard Edition (SE) application and reference them to access the remote web service.

To use the generated client proxy classes in a JSE component:

1. Copy the generated client proxy classes to your JSE application source directory.
2. Using the main client proxy class, called `port_nameClient.java`, as your guide, add appropriate methods to access the web service from your application.
3. Run the application.

16.4.2.3 How to Use the Java EE Component Client Application Deployed to WebLogic Server

In this case, the web service runs inside a Java Platform, Enterprise Edition (Java EE) Version 5 component deployed to WebLogic Server, such as an EJB, servlet, or another web service. This type of client application, therefore, runs inside a WebLogic Server container.

To use the generated client proxy classes in a Java EE component:

1. Open the main client proxy class, called `port_nameClient.java`, in the source editor.

   This file opens automatically when you create the web service client proxy initially. To re-open the class, right-click on the client proxy container and select **Go to Client Class** or simply double-click on the file in the Application Navigator.

2. Replace the main method with your own method(s) to access the web service and perform required operations. You can use the code generated in the main method as a guide.

3. Deploy the full set of client module classes that JDeveloper has generated.

4. Reference the client proxy class in your Java EE application.

16.4.3 How to View the WSDL Used to Create the Web Service Client

You can view the WSDL that was used to generate the web service client under the following circumstances:

- If available, the local copy of the WSDL file is displayed. When generating the web service client, you have the option to copy the WSDL of the source web service to your local directory. See Section 16.4.1, "How to Create the Client and Proxy Classes".

  Note: In most cases, the local copy of the WSDL will match the WSDL of the remote web service. If the remote web service is modified, the local WSDL may become out-of-sync with the remote WSDL. To ensure the web service client will be able to access the remote web service, you can regenerate the local WSDL using the remote WSDL, as needed. See Section 16.4.5, "How to Regenerate Web Service Client and Proxy Classes".

- If the local version is not available, the remote WSDL is displayed.

To view the client WSDL:

1. Right-click on the web service client within the Application Navigator.

2. Select **Go To WSDL** from the pop-up menu.

   The WSDL is displayed.

16.4.4 How to Update the Web Service WSDL at Run Time

In some cases, you may need to update your application to reference imported XML resources, such as WSDLs and XSDs, from a source that is different from that which is part of the description of the web service. Redirecting the XML resources in this way

may be required to improve performance or to ensure your application runs properly in your local environment.

For example, a WSDL may be accessible during client generation, but may no longer be accessible when the client is run. You may need to reference a resource that is local to or bundled with your application rather than a resource that is available over the network.

You can modify the location of the WSDL that will be used by the web service at runtime using one of the following methods:

- XML Catalog File
- Web Service Injection (`@WebServiceRef`) and a Deployment Plan

16.4.4.1 How to Use an XML Catalog File

When you create or regenerate a web service client, a `jax-ws-catalog.xml` file is created automatically in the `META-INF` directory. The file complies with the OASIS XML schema, as described in the Oasis XML Catalogs specification at http://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html.

You can update the web service WSDL by modifying the uri attribute of the <system> element in the jax-ws-catalog.xml file. The specified value will be used at run time.

The following provides a sample XML catalog (jax-ws-catalog.xml) file for a remote WSDL:

```
<catalog xmln="urn:oasis:names:tc:entity:xmlns:xml:catalog"
   prefer="system">
   <system systemId="http://foo.org/hello?wsdl"
      uri="http://foo.org/hello?wsdl" />
</catalog>
```

The following provides a sample XML catalog (jax-ws-catalog.xml) file for a local WSDL:

```
<catalog xmln="urn:oasis:names:tc:entity:xmlns:xml:catalog"
   prefer="system">
   <system systemId="http://foo.org/hello?wsdl"
      uri="../org/foo/HelloService.wsdl" />
</catalog>
```

In the preceding examples:

- The `<catalog>` root element defines the XML catalog namespace and sets the prefer attribute to system to specify that system matches are preferred.
- The `<system>` element associates a URI reference with a system identifier.

Note: When creating the client and proxy classes for multiple web services on a local system that share the same endpoint, to ensure that URL is unique for each web service in the jaxws-catalog.xml file, the service QName is appended as anchor text. For example:

http://foo.org/helloworld?wsdl

Might become:

http://foo.org/helloworld#%7Bhttp%3A%2F%2Fexample.com%2F%7DHelloService?wsdl.

16.4.4.2 How to Use Web Service Injection (@WebServiceRef) and a Deployment Plan

This method involves the following steps:

1. Using the `@WebServiceRef` annotation to define a reference to a web service and identify an injection target.
2. Updating the deployment plan and modifying the value of the web service WSDL that is referenced at run time.

Step 1: Using the @WebServiceRef Annotation

The @WebServiceRef annotation injects an endpoint for the Web service interface that is defined in the web.xml file. The following example demonstrates how to use the @WebServiceRef annotation to define a reference to a web service and identify an injection target.

```
@WebService
public class LoansApprover {
   /**
    ** Credit rating service injected from web.xml
    **/
   @WebServiceRef(name = "CreditRatingService")
   CreditRating creditRating;

   /**
    ** @return Loan application with approval code if approved.
    **/
   public LoanApprovalReponse approveLoan(LoanApplication la) {
      ...
   }
}
```

The web service class for the CreditRatingService is hard-coded in the web.xml file, as shown in the following example:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      version="2.5"
      xmlns="http://java.sun.com/xml/ns/javaee">
   ...
   <service-ref>
      <service-ref-name>CreditRatingService</service-ref-name>
      <service-interface>
        com.somecreditrating.xmlns.rating.CreditRating_Service
```

```
      </service-interface>
   </service-ref>
</web-app>
```

Step 2: Updating the Deployment Plan

To modify the value of the WSDL that is used at run time, you can generate and update a deployment plan.

A deployment plan is an optional XML document that you use to configure an application for deployment to a specific WebLogic Server environment. A deployment plan defines or overrides deployment property values that would normally be defined in an application's WebLogic Server deployment descriptors. To update the configuration for your application, you add or update variables in the deployment plan, defining both the location of the WebLogic Server descriptor properties and the value to assign to the properties. For more information, see the *Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server*.

The following example illustrates a deployment plan that overrides the value of the CreditRatingService web service WSDL, where:

- The `variable-definition` element defines the `CreditRatingService` variable and the value to assign to it.
- As part of the `module-override` element for the `LoanApplication-LoanApprover-context-root.war`, a variable-assignment element defines the `CreditRating Service` variable and the exact location within the descriptor where the property is overridden.

```
<?xml version='1.0' encoding='UTF-8'?>
<deployment-plan xmlns="http://www.bea.com/ns/weblogic/deployment-plan"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/ns/weblogic/deployment-plan
   http://www.bea.com/ns/weblogic/deployment-plan/1.0/deployment-plan.xsd"
   global-variables="false">
   <application-name>production</application-name>
   <variable-definition>
      <variable>
         <name>CreditRatingService</name>
         <value>http://www.somecreditrating.com/xmlns/rating?WSDL</value>
      </variable>
   </variable-definition>
   <module-override>
      <module-name>production.ear</module-name>
      <module-type>ear</module-type>
      <module-descriptor external="false">
         <root-element>weblogic-application</root-element>
         <uri>META-INF/weblogic-application.xml</uri>
      </module-descriptor>
      <module-descriptor external="false">
         <root-element>application</root-element>
         <uri>META-INF/application.xml</uri>
      </module-descriptor>
      <module-descriptor external="true">
         <root-element>wldf-resource</root-element>
         <uri>META-INF/weblogic-diagnostics.xml</uri>
      </module-descriptor>
   </module-override>
   <module-override>
      <module-name>
```

```
        LoanApplication-LoanApprover-context-root.war
      </module-name>
      <module-type>war</module-type>
      <module-descriptor external="false">
         <root-element>weblogic-web-app</root-element>
         <uri>WEB-INF/weblogic.xml</uri>
      </module-descriptor>
      <module-descriptor external="false">
         <root-element>web-app</root-element>
         <uri>WEB-INF/web.xml</uri>
         <variable-assignment>
            <name>CreditRatingService</name>
            <xpath>
       /web-app/service-ref/[service-ref-name="CreditRatingService"]/wsdl-file
            </xpath>
            <operation>add</operation>
         </variable-assignment>
      </module-descriptor>
      <module-descriptor external="true">
         <root-element>weblogic-webservices</root-element>
         <uri>WEB-INF/weblogic-webservices.xml</uri>
      </module-descriptor>
      <module-descriptor external="false">
         <root-element>webservices</root-element>
         <uri>WEB-INF/webservices.xml</uri>
      </module-descriptor>
      <module-descriptor external="true">
         <root-element>webservice-policy-ref</root-element>
         <uri>WEB-INF/weblogic-webservices-policy.xml</uri>
      </module-descriptor>
   </module-override>
   <config-root>
      D:\prom-demo\jdeveloper\mywork\LoanApplication\deploy\production\.\plan
   </config-root>
</deployment-plan>
```

16.4.5 How to Regenerate Web Service Client and Proxy Classes

There are times that you may need to regenerate the web service client and proxy classes.

> **Note:** When you regenerate the web service client and proxy classes, JDeveloper discards any changes that you have made to the class, WSDL, or supporting files since the client was last generated.

To regenerate the web service client and proxy classes:

1. In the Application Navigator, right-click the web service client node that you want to regenerate and choose **Properties** from the context menu.

   The Web Service Client and Proxy Editor wizard is displayed.

2. Select **Web Service Description**. (It should be selected by default.)
3. Select **Refresh Copied WSDL from Original WSDL Location** if you wish to refresh the local WSDL using the WSDL at the original location.
4. Click **OK**.

The local copy of the WSDL is refreshed and the web service client and proxy classes are regenerated.

To regenerate the web service client and proxy classes:

You can regenerate the web service client and proxy classes quickly and easily using the set of properties last defined in the Web Service Client and Proxy Editor wizard and the current locally stored WSDL as follows:

- In the Application Navigator, right-click the web service client node that you want to regenerate and choose **Regenerate Web Service Proxy** from the context menu.

  The web service client class, WSDL, and supporting proxy files are regenerated.

16.4.6 How to Manage the Web Service Clients

JDeveloper provides the ability to both edit and delete web service clients.

To edit web service clients:

You can edit a web service client using the Web Service Client and Proxy editor. To access the Web Service Client and Proxy editor:

1. Double-click on the client within the Application Navigator.
2. Right-click on the client within the Application Navigator, and select **Properties...**

   For help in completing the wizard, press F1 or click **Help** from within the wizard.

To delete web service clients:

1. In the Application Navigator, expand the node that contain the web service client proxy files, `package.proxy`, and select the files.
2. Choose **File > Erase from Disk**. You can ignore any usages JDeveloper finds.
3. Expand the node that contains the web service proxy runtime files, `package.proxy.runtime`, and select the files.
4. Choose **File > Erase from Disk**.

   The files are permanently erased.

16.4.7 How to Reference Web Services Using the @WebServiceRef Annotation

When you use the `javax.xml.ws.WebServiceRef` annotation, you can inject a reference to a web service into any container-managed Java class.

To add a `@WebServiceRef` annotation to your Java class quickly and easily, right-click within the Java class editor at the location you want to inject the web service reference, and select one of the following options:

- Select **Create Proxy and Insert Reference** from the context menu.

  This command invokes the Create Web Service Client and Proxy wizard, enabling you to generate a web service client and proxy classes. Then, the `javax.xml.ws.WebServiceRef` and web service proxy classes are imported automatically and a reference to the selected web service is injected at the specified location.

- Select **Insert Proxy Reference** from the context menu, then select an existing Web service proxy from the drop-down list.

The `javax.xml.ws.WebServiceRef` and web service proxy classes are imported automatically and a reference to the selected web service is injected at the specified location. If no web service proxy classes are currently available, then this option is greyed out.

The following excerpt provides an example of the code that is automatically added to the Java class:

```
import java.xml.ws.WebServiceRef;
import ratingservice.CreditRatingService;
...
/**
 ** Injectable field for service WebServiceClient
**/
@WebServiceRef
CreditRatingService creditRatingService1;
...
```

For more information, see "Defining a Web Service Reference Using @WebServiceRef Annotation" in *GD*.

16.5 Creating SOAP Web Services (Bottom-Up)

Web services can be created using two development methods: top-down or bottom-up. Bottom-up development refers to the process of developing a web service from the underlying Java implementation using SOAP. For information about using top-down development—starting from the WSDL—see Section 16.6, "Creating SOAP Web Services from WSDL (Top Down)".

The following sections describe how to generate different types of web services from the bottom up:

- Section 16.5.1, "How to Create Java Web Services"
- Section 16.5.2, "How to Use JSR-181 Annotations"
- Section 16.5.3, "How to Create PL/SQL Web Services"
- Section 16.5.4, "How to Create TopLink Database Web Service Providers"
- Section 16.5.5, "How to Use Web Service Atomic Transactions"
- Section 16.5.6, "How to Regenerate Web Services from Source"
- Section 16.5.7, "How to Use Handlers"
- Section 16.5.8, "How to Expose Superclass Methods for JAX-RPC"
- Section 16.5.9, "How to Handle Overloaded Methods"
- Section 16.5.10, "How to Set Mappings between Java Methods and WSDL Operations Using the JAX-RPC Mapping File Editor"

16.5.1 How to Create Java Web Services

You can create web services from:

- Java classes
- Remote interface of EJBs
- ADF Business Components service session bean wrapped as an EJB

The web service creation wizards create the deployment files for you, so once you have created your web service the final step is to deploy it.

Before you begin:

If you have not already done so, set an appropriate context root for your web service. For more information, see Section 16.2.2, "How to Set the Context Root for Web Services".

To create the web service:

1. In the Application Navigator, select the project containing the Java class or EJB from which you want to create a web service.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**. In the **Items** list, double-click **Java Web Service** to launch the Create Java Web Service wizard.

   For detailed help about completing the wizard, press F1 or click **Help** from within the wizard.

> **Note:** The Select Deployment Platform page is only displayed the first time a web service is created in a project. Thereafter, all additional web services in the same project will use the same version.

16.5.2 How to Use JSR-181 Annotations

JSR-181 specifies web services meta data, which allows you to use annotations to declaratively to make creating and managing web services easier. You use the annotations for methods and classes in order to expose these methods as web service end-points.

You can add JSR-181 annotations to a class manually, choose to have JDeveloper add them to the class when creating the web service, or add them when editing the web service using the Edit Web Services dialog.

> **Note:** If you delete the annotations using the Edit Web Services dialog, any annotations that you entered manually are also deleted.

To add annotations:

1. Open the Java class open in the source editor.
2. On correct line, type `@` and pause for a couple of seconds.

   Code Insight displays possible values. For more information, see Section 16.2.7, "How to Work with Web Services Code Insight".

For more information, see the following references:

- JSR-181 specification at `http://jcp.org/en/jsr/detail?id=18`
- JAX-WS specification at: `http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html`
- For JWS annotations available with WebLogic Server see "JWS Annotation Reference" in *Oracle Fusion Middleware WebLogic Web Services Reference for Oracle WebLogic Server*.

16.5.3 How to Create PL/SQL Web Services

The Create PL/SQL Web Service wizard makes it easy to generate a web service from a PL/SQL package or a Java stored procedure that uses object types. A Java stored procedure is defined by a SQL specification that invokes it, and the PL/SQL Web Service wizard treats these in the same way as packages.

> **Note:** You can only create JAX-RPC PL/SQL Web Services. For more information, see Section 16.2.6, "How to Choose Your Deployment Platform".

PL/SQL web services can be deployed to Oracle WebLogic Server. The Create PL/SQL Web Service wizard uses the functionality of Oracle JPublisher to wrap the PL/SQL in Java so that the service can be published. For more information see Section 28.3.1, "How to Use JPublisher".

You can either:

- Create the web service starting from a project in the Application Navigator. In this case, you select the database connection and the PL/SQL package to generate the web service from.
- Create the web service from the PL/SQL package under the database connection node in the Database Navigator or the Resource Palette. In this case, you have to select the project into which the generated files for the web service are deployed.

It should be noted that:

- If you edit a PL/SQL web service, make sure that the database connection still exists otherwise you will see an error message. If you have deleted the database connection, create a new one with the same name as the original connection.
- There are some cases where a web service cannot be created. For more information on the limitations, see Section 16.2.5, "How to Work with PL/SQL Web Services and Types".
- Deploying PL/SQL web services is similar to deploying other J2EE Web Applications. For more information, see Section 16.12, "Deploying Web Services".

To create the PL/SQL web service from a project:

1. In the Application Navigator, select the project.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**. In the **Items** list, double-click **PL/SQL Web Service** to launch the Create PL/SQL Web Service wizard.

   For detailed help about completing the wizard, press F1 or click **Help** from within the wizard.

To create the web service from a PL/SQL package:

1. In the Database Navigator or the Resource Palette, expand the database connection node, the schema node, the **Packages** node, then the node of the package.
2. Right-click the PL/SQL package body, and choose **Publish as Web Service** to launch the Create PL/SQL Service wizard.

16.5.4 How to Create TopLink Database Web Service Providers

The Create TopLink DB Web Service Provider wizard enables you to build a JAX-WS web service provider for a TopLink database to perform one of the following tasks:

- Access stored procedures and functions
- Execute an SQL query
- Perform CRUD operations on a table

Based on the type of service selected, the wizard generates a web service provider and WSDL document that can be deployed to an application server, such as Oracle WebLogic Server. Deploying TopLink web service providers is similar to deploying other J2EE Web Applications. For more information, see Section 16.12, "Deploying Web Services".

It should be noted that:

- The wizard generates a JAX-WS web service provider.
- If you edit a TopLink web service provider, ensure that the database connection still exists; otherwise an error message is returned. If you have deleted the database connection, create a new one with the same name as the original connection.
- In some cases, you may not be able to generate a TopLink web service provider. For more information on the limitations, see Section 16.2.5, "How to Work with PL/SQL Web Services and Types".

To create the TopLink web service provider from a project:

1. In the Application Navigator, select the project.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**. In the **Items** list, double-click **TopLink DB Web Service Provider** to launch the Create TopLink Web Service Provider wizard.

   For detailed help about completing the wizard, press F1 or click **Help** from within the wizard.

16.5.5 How to Use Web Service Atomic Transactions

WebLogic web services enable interoperability with other external transaction processing systems, such as Websphere, JBoss, Microsoft .NET, and so on, through the support of the following specifications:

- WS-AtomicTransaction (Versions 1.0, 1.1, and 1.2) at `http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html`
- WS-Coordination (Versions 1.0, 1.1, and 1.2) at `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html`

These specifications define an extensible framework for coordinating distributed activities among a set of participants. The coordinator is the central component, managing the transactional state (coordination context) and enabling web services and clients to register as participants. For more information about web service atomic transactions, see "Using Web Service Atomic Transactions" in *Oracle Fusion Middleware Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*.

To enable atomic transactions for a web service implementation at the class level or synchronous method level (for two-way methods only) use one of the following methods:

- Adding `@weblogic.wsee.wstx.wsat.Transactional` annotation directly in the Java class; the JDeveloper Code Insight feature can help you. For more information, see Section 16.2.7, "How to Work with Web Services Code Insight".
- Using the Property Inspector, as described below.

To enable atomic transactions for web service clients use one of the following methods:

- Right click on the `@WebServiceRef` annotation or web service injectable target, and select **Add Transactional** from the menu to add the `@Transactional` annotation.
- Pass the `weblogic.wsee.wstx.wsat.TransactionalFeature` as a parameter when creating the web service proxy or dispatch. For more information, see "Using Web Service Atomic Transactions" in *Oracle Fusion Middleware Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*.

When enabling web service atomic transactions, configure the following information:

- **Version**: Version of the web service atomic transaction coordination context that is used for web services and clients. For clients, it specifies the version used for outbound messages only. The value specified must be consistent across the entire transaction. Valid values include `WSAT10`, `WSAT11`, and `WSAT12`, and `DEFAULT`. The `DEFAULT` value for web services is all three versions (driven by the inbound request); the `DEFAULT` value for web services clients is `WSAT10`.
- **Flow type**: Flag that specifies whether the coordination context is passed with the transaction flow. The following table summarizes the valid values and their meaning on the web service and client. The table also summarizes the valid value combinations when configuring web service atomic transactions for an EJB-style web service that uses the `@TransacationAttribute` annotation.

Table 16–3 Transaction Configurations

Value	Web Service Client	Web Service	Valid EJB `@TransactionAttribute` Values
`NEVER`	**JTA transaction**: Do not export transaction coordination context. **No JTA transaction**: Do not export transaction coordination context.	**Transaction flow exists**: Do not import transaction coordination context. If the CoordinationContext header contains mustunderstand="true", a SOAP fault is thrown. **No transaction flow**: Do not import transaction coordination context.	`NEVER, NOT_SUPPORTED, REQUIRED, REQUIRES_NEW, SUPPORTS`
`SUPPORTS` (Default)	**JTA transaction**: Export transaction coordination context. **No JTA transaction**: Do not export transaction coordination context.	**Transaction flow exists**: Import transaction context. **No transaction flow**: Do not import transaction coordination context.	`SUPPORTS, REQUIRED`

Table 16–3 (Cont.) Transaction Configurations

Value	Web Service Client	Web Service	Valid EJB `@TransactionAttribute` Values
`MANDATORY`	**JTA transaction**: Export transaction coordination context. **No JTA transaction**: An exception is thrown.	**Transaction flow exists**: Import transaction context. **No transaction flow**: Service-side exception is thrown.	`MANDATORY, REQUIRED, SUPPORTS`

To enable web service atomic transactions in the Java class:

1. Open the web service class in the source editor.
2. You can use the JDeveloper Code Insight to help you.

   Start typing the annotation, for example, `@Transactional`. When you pause, or click **Ctrl+Shift+Space**, a popup appears from which you can choose the correct entry to complete the statement.
3. You can specify the version and flow type values as follows:

```
@Transactional(version=Transactional.Version.[WSAT10|WSAT11|WSAT12|DEFAULT],
        value=Transactional.TransactionFowType.[MANDATORY|SUPPORTS|NEVER])
```

To enable web service atomic transactions in the Property Inspector:

1. With the web service class open in the source editor, choose **View Property Inspector** to open the Property Inspector.

   For more information at any time, press F1 or click **Help** from within the Property Inspector.
2. With the cursor in the public class, `@WebService`, or two-way method line of the class, navigate to the Web Services Extensions node in the Property Inspector.
3. Select **Add Transactional**.

   The Property Inspector is refreshed to display options to set the flow type and version. For more information about the configuration options, see Table 16–3.
4. Select a flow type from the Flow Type drop-down list. Valid values include: `Supports`, `Never`, and `Mandatory`. This field defaults to `Supports`.
5. Select a version from the Version drop-down list. Valid values include: `WS-AT 1.0`, `WS-AT 1.1`, `WS-AT 1.2`, and `Default`. The `Default` value for web services is all three versions (driven by the inbound request); the `Default` value for web services clients is `WS-AT 1.0`.

To enable web service atomic transactions in a web service client's injectable target:

1. Open the web service client in the source editor.
2. Right-click on the `@WebServiceRef` annotation or injectable target and select **Add Transactional** from the menu.

   The `@Transactional` annotation is added to the web service client.
3. You can specify the version and flow type values as follows:

```
@Transactional(version=Transactional.Version.[WSAT10|WSAT11|WSAT12|DEFAULT],
          value=Transactional.TransactionFowType.[MANDATORY|SUPPORTS|NEVER])
```

For more information about the configuration options, see Table 16–3.

16.5.6 How to Regenerate Web Services from Source

There are times that you may need to regenerate your web service. For example, if the source from which the service was originally generated has changed.

> **Note:** When you regenerate the web service, JDeveloper discards any changes that you have made to the WSDL since it was last generated.

After you regenerate the web service, you may need to regenerate the client to the web service. Otherwise, you may get compilation errors (when the client is in the same project as the web service), or run-time errors (when the client is in a different project to the web service).

If you are not using annotations and change the name of the method in the underlying class, when you regenerate the service you will receive an error message indicating that no methods were selected. Because methods are tracked using namespaces, if you modify the namespace JDeveloper is not able to determine what needs to be regenerated. To correct this error, double-click the web service container to open the Web Services Editor, go to the Methods page, and select the methods on which to base the web service.

To regenerate a web service from source:

1. In the Application Navigator, right-click the web service container you want to regenerate.
2. Choose **Regenerate Web Service from Source** from the context menu.

   The service is automatically regenerated, and any changes you made to the WSDL since it was last generated are lost.

16.5.7 How to Use Handlers

JDeveloper allows you to specify the handler classes to deal with the web service message. The handlers can use initialized parameters, SOAP roles or SOAP headers.

To define handlers:

1. Create a web service. For more information, see Section 16.5.1, "How to Create Java Web Services".

   or

   Open the web service editor. For more information, see Section 16.10, "Editing and Deleting Web Services".
2. In the Handler Details page, enter the values you want to use.

   For more information at any time, press F1 or click **Help** from within the dialog.

16.5.8 How to Expose Superclass Methods for JAX-RPC

> **Note:** For JAX-WS web services, superclass methods are always exposed.

To expose superclass methods for JAX-RPC, consider the following two examples:

```
package mypackage;
public class Shape {
    public void area() {
    }
}
```

and

```
package mypackage;
public class Circle extends Shape {
    public Circle() {
    }
    public void callParentMethod() {
        super.area();
    }
}
```

In class `Circle`, which extends `Shape`, there is a public method `callParentMethod()` which is responsible for calling the parent class method `area`. To call the superclass method `area()`, create a J2EE Java web service on the `Circle` class using the public method `callParentMethod()`.

16.5.9 How to Handle Overloaded Methods

If the Java class on which you base a web service has overloaded methods, JDeveloper handles them automatically. However if you create a J2EE 1.4 web service, and then change the class on which it is based so that an existing method becomes an overloaded method you have to take action to update the mapping file.

The procedure to handle overloaded methods depends on the type of web service that you are developing, JAX-WS or JAX-RPC.

Handling Overloaded Methods for JAX-WS Web Services

For JAX-WS web services, you can use the `@WebMethod` annotation to change the name of an overloaded method. For example:

```
public class SimpleImpl {
   @WebMethod(operationName="sayHelloOperation")
   public String sayHello(String message) {
      System.out.println("sayHello:" + message);
      return "Here is the message: '" + message + "'";
   }
...
}
```

In the example, the `sayHello()` method of the SimpleImpl JWS file is exposed as a public operation of the web service. The `operationName` attribute specifies, however, that the public name of the operation in the WSDL file is `sayHelloOperation`.

For more information about `@WebMethod`, see "Specifying that a JWS Method Be Exposed as a Public Operation (@WebMethod and @OneWay Annotations)" in *Oracle Fusion Middleware Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.

Handling Overloaded Methods for JAX-RPC Web Services

For JAX-RPC web services, there are two ways that you can handle overloaded methods:

- Manually modify the mapping file

- Delete the mapping file and recreate the web service

To manually modify the mapping file

After you create the web service based on the Java class, add the overloaded method to the class.

To delete the mapping file and recreate the web service

1. In the Application Navigator, expand **Web Content** and **WEB-INF**.
2. Right-click **<web_service>-java-wsdl-mapping.xml** and choose **Delete**.
3. Open the web service editor. For more information, see Section 16.10, "Editing and Deleting Web Services".
4. Click **OK** to close it and regenerate the service.

16.5.10 How to Set Mappings between Java Methods and WSDL Operations Using the JAX-RPC Mapping File Editor

JAX-RPC maps Java types to WSDL definitions. However when the types you want to support are not covered by the JSR-109 specification, or when you want to use different mappings to provide the functionality your web service requires, you can use the JAX-RPC Mapping File Editor to amend an existing mapping file, or to create your own.

The JAX-RPC Mapping File Editor is a specialized schema-driven editor which helps you to create a JSR-109 compliant mapping file for a J2EE 1.4 web service. The mapping file standardizes the Java- WSDL mappings, and in general you have to provide a full mapping only when the default mapping rules in JSR-109 are not satisfied.

These features are available while you are using the JAX-RPC Mapping File Editor:

- While you are typing, you can invoke Code Insight by pausing after typing the < (open bracket), or by pressing Ctrl+Space (if you are using the default key mapping). Code Insight opens a list with valid elements based on the schema.
- You can choose **View > Component Palette** to open the Palette and select one of the available pages from the dropdown list.
- A mapping file's elements are displayed hierarchically in the Structure window, which also displays any XML syntax errors found as you type and edit. You can double-click on an element or error to edit it in the JAX-RPC Mapping File Editor.
- You can right-click on an XML element in the editor and choose Locate in Structure to expand the Structure window to both show the element and select it. While you are editing, you can right-click in the open file and choose Auto Indent XML to properly indent the elements.
- In an open JAX-RPC Mapping File Editor window, or in the Structure window with the web service selected in the Application Navigator, right-click a mapping file and choose Validate WSDL. The Validate WSDL command will validate the XML against the registered schemas.

16.6 Creating SOAP Web Services from WSDL (Top Down)

JDeveloper allows you to develop top-down web services, that is, starting with the WSDL. JDeveloper will generate a service implementation and its deployment

descriptors. You can browse to a WSDL in the file system, or use the Find Web Service Wizard to locate a web service in a UDDI registry connection in the Resource Palette.

To create a SOAP web service from WSDL (top down):

1. In the Application Navigator, select the project in which you want to create the web service.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**. In the **Items** list, double-click **Java Web Service From WSDL** to launch the Create Java Web Service from WSDL wizard.

   For detailed help about completing the wizard, press F1 or click **Help** from within the wizard.

   The SOAP web service is created and the Java implementation class is opened automatically in the editor

16.7 Creating RESTful Web Services

Representational State Transfer (REST) describes any simple interface that transmits data over a standardized interface (such as HTTP) without an additional messaging layer, such as SOAP. REST provides a set of design rules for creating stateless services that are viewed as resources, or sources of specific information, and can be identified by their unique URIs. A client accesses the resource using the URI, a standardized fixed set of methods, and a representation of the resource is returned. The client is said to transfer state with each new resource representation.

When using the HTTP protocol to access RESTful resources, the resource identifier is the URL of the resource and the standard operation to be performed on that resource is one of the HTTP methods: `GET`, `PUT`, `DELETE`, `POST`, or `HEAD`.

The following sections describe how to create RESTful web service and clients:

- Section 16.7.1, "How to Add the Jersey JAX-RS Reference Implementation to Your Project"
- Section 16.7.2, "How to Create JAX-RS Web Services and Clients"

16.7.1 How to Add the Jersey JAX-RS Reference Implementation to Your Project

Before you can create RESTful web services in JDeveloper, you need to download and add to your project the Jersey JAX-RS Reference Implementation (RI). The Jersey JAX-RS RI is available at `http://jersey.java.net`. Click **Download** for more information about the Jersey RI and to download the ZIP file that contains the relevant library JAR files. Once downloaded, you need to add the Jersey RI to your project.

Note: The Jersey RI 1.1.5.1 (`jersey-archive-1.1.5.1`) version or above is compatible with this release of JDeveloper.

To add the Jersey JAX-RS RI to your project:

1. With the project selected in the Application Navigator, open the Project Properties dialog.

   To display the dialog, double-click the Project folder or select **Edit > Properties**.
2. Select the **Libraries and Classpath** node.

3. On the Libraries and Classpath page, click **Add Library**.
4. In the Add Library dialog, click **New**.
5. In the Create Library dialog, enter a name for the new library (for example, JAX-RS) and select its location.
6. Enable **Deployed by Default**.

> **Note:** If you do not select this check box, you will experience errors during deployment of your RESTful web services and clients.

7. Select **Class Path** and click **Add Entry**.
8. In the Select Path Entry dialog, navigate to the lib directory of the Jersey archive. For example, c:\mylibraries\jersey-archive-1.1.5.1\lib.
9. Select all of the JAR files in the lib directory and click **Select**.

> **Note:** If you are developing RESTful web services only (that is, you are not developing RESTful clients), you do not have to include the jersey-client-1.1.5.1.jar. Similarly, if you are developing RESTful web service clients only, you do not have to include the jersey-server-1.1.5.1.jar.

10. If you downloaded the source files, you can set the **Source Path** to point to the source files (similar to the way that you defined the Class Path in the previous steps).
11. In the Create Library dialog, click **OK**.
12. In the Add Library dialog, click **OK**.
13. On the Libraries and Classpath page, if finished click **OK**.

Once you have added the Jersey JAX-RS RI to your project, you can then create JAX-RS web services and clients.

16.7.2 How to Create JAX-RS Web Services and Clients

After you have added the Jersey JAX-RS RI to your project, you can start creating JAX-RS web services and clients using JDeveloper. All of the standard Java source editor features will work with the JAX-RS calls, such as code insight, import assistance, and so on.

For more information about JAX-RS and samples, you might find it helpful to review the Jersey RI documentation at: `http://wikis.sun.com/display/Jersey/Main`.

Once you create your RESTful web services, you can test them using the HTTP Analyzer. For more information, see Section 16.13.4, "How to Examine Web Services using the HTTP Analyzer".

> **Note:** If you experience errors during the deployment of your RESTful web services and clients, ensure that you have selected the **Deployed by Default** check box when adding the Jersey JAX-RS RI to your project.

A Simple Hello World Example

Example 16–1 provides a very simple example of a RESTful web services:

Example 16–1 RESTful web services

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

// Specifies the path to the RESTful service
@Path("/helloworld")
public class helloWorld {

   // Specifies that the method processes HTTP GET requests
   @GET
   @Path("sayHello")
   @Produces("text/plain")
   public String sayHello() {
      return "Hello World!";
   }
}
```

Example 16–2 provides a simple RESTful client that calls the RESTful web service defined previously. This sample uses classes that are provided by the Jersey JAX-RS RI specifically; they are not part of the JAX-RS standard.

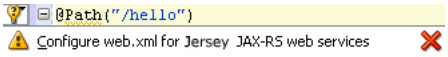
Example 16–2 RESTful client

```
package samples.helloworld.client;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;

public class helloWorldClient {
   public helloWorldClient() {
      super();
   }

   public static void main(String[] args) {
      Client c = Client.create();
      WebResource resource = c.resource(
            "http://localhost:7101/RESTfulService-Project1-context-root/
            jersey/helloWorld");
      String response = resource.get(String.class);
   }
}
```

About the web.xml File

JDeveloper does not automatically add the servlet class to the web.xml file. Instead, you are prompted to confirm whether you want to add it when you call a JAX-RS method from your code. For example, see Figure 16–1.

Figure 16–1 Prompt to confirm update



> **Note:** Why is the web.xml file not updated automatically? In the future, when you deploy to a Java EE 6.0 container, an update to the web.xml will not be required. Therefore, this is set up as an optional activity.

To update the web.xml:

To update the web.xml, select **Configure web.xml for Jersey JAX-RS web services** as follows:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5" xmlns="http://java.sun.com/xml/ns/javaee">
   <servlet>
      <servlet-name>jersey</servlet-name>
      <servlet-class>
          com.sun.jersey.spi.container.servlet.ServletContainer
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>jersey</servlet-name>
      <url-pattern>/jersey/*</url-pattern>
   </servlet-mapping>
</web-app>
```

16.8 Managing WSDLs

JDeveloper provides a number of ways that you can manage WSDLs for a web service, as described in the following sections:

- Section 16.8.1, "How to Create WSDL Documents"
- Section 16.8.2, "How to Add a WSDL to a Web Service Project"
- Section 16.8.3, "How to Display the WSDL for a Web Service"
- Section 16.8.4, "How to Save a WSDL to Your Local Directory"

16.8.1 How to Create WSDL Documents

You can create a WSDL document, for example, to create a top-down web service.

To create a WSDL:

1. In the Application Navigator, select the project containing the Java class or EJB from which you want to create a web service.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** list, expand **Business Tier** and select **Web Services**. In the **Items** list, double-click **WSDL Document** to open the Create WSDL Document dialog.

For detailed help about completing the wizard, press F1 or click **Help** from within the dialog.

16.8.2 How to Add a WSDL to a Web Service Project

You can generate a WSDL file for a web service and add it to the project using the procedures described below. The WSDL file is generated automatically and added to the `WEB-INF/wsdl` directory for Web applications and to the `META-INF/wsdl` directory for EJB applications within the project. In addition, the `@WebService` annotation is updated with the wsdlLocation attribute to reference the location of the local WSDL. For example:

```
@WebService(wsdlLocation="WEB-INF/wsdl/CreditRatingService.wsdl")
```

> **Note:** If a WSDL file already exists in the `WEB-INF/wsdl` or `META-INF/wsdl` directory, you are prompted whether or not to overwrite the existing WSDL file.

To add a WSDL to a web service project:

In the Application Navigator, right-click the web service for which you want to add a WSDL and select **Generate WSDL and Add to Project** from the context menu. The WSDL is automatically generated and added to the project in the `WEB-INF/wsdl` directory.

16.8.3 How to Display the WSDL for a Web Service

You can display the WSDL for a web service. The WSDL file is generated based on the annotations defined in the web service to a temporary directory and displayed.

To display the WSDL to a web service project:

In the Application Navigator, right-click the web service for which you want to display the WSDL and select **Show WSDL for Web Service Annotations** from the context menu.

The WSDL is generated to a temporary directory and displayed.

16.8.4 How to Save a WSDL to Your Local Directory

When viewing a remote WSDL for a web service, you can save the WSDL to your local directory.

> **Note:** If you want to use the WSDL within a web service project, you need to copy it to a location that is accessible by the project directory (for example, `WEB-INF/wsdl` for Web applications and `META-INF/wsdl` for EJB applications) and update the `@WebService` annotation to reference the WSDL location.

To save a WSDL to your local directory:

1. Display the WSDL file for the web service.
2. Choose **Tools > Copy WSDL Locally**.

3. In the Select Destination for WSDL dialog, navigate to the location that you want to save the WSDL, or enter the location in the Directory name text box, and click **Select**.

   The WSDL is saved to the location specified.

16.9 Using Policies with Web Services

This section describes how to use policies with web services created in JDeveloper. You can use the following types of policies:

- Oracle Web Service Manager (Oracle WSM) policies—Attach security policies only to JAX-WS web services.
- Oracle WebLogic web service policies—Attach to JAX-WS or JAX-RPC web services.

You cannot mix the two types of policies in the same web service, so you should decide which to use at the planning stage. Once you have added policies of one type to your web service, you cannot switch to the other type without deleting the policies that are currently attached. For example, if you have configured Oracle WSM policies and later decide that you want to use Oracle WebLogic web service policies, you must delete the Oracle WSM policies before you can attach the Oracle WebLogic web service policies.

The following sections describe how to use policies with web services:

- Section 16.9.1, "What You May Need to Know About Oracle WSM Policies"
- Section 16.9.2, "What You May Need to Know About Oracle WebLogic Web Service Policies"
- Section 16.9.3, "How to Attach Policies to Web Services"
- Section 16.9.4, "How to Attach Oracle WSM Policies to Web Service Clients"
- Section 16.9.5, "How to Invoke Web Services Secured Using WebLogic Web Service Policies"
- Section 16.9.6, "How to Edit and Remove Policies from Web Services"
- Section 16.9.7, "How to Use Custom Web Service Policies"
- Section 16.9.8, "How to Use a Different Oracle WSM Policy Store"

Before you begin:

A detailed examination of all the tasks to be performed to use policies is outside the scope of this guide, but in general the steps you need to perform are:

1. Decide on the policies you intend to use. For more information, see "Determining Which Security Policies to Use" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.
2. Attach the policies to a class or service. For more information, see Section 16.9.3, "How to Attach Policies to Web Services".
3. Configure a server with the correct key stores or other information that the policies need to work, and deploy the web service to the server. For more information, see "Configuring Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.
4. Test the web service to ensure that the policies work as expected. For more information, see Section 16.11.1, "How to Test Web Services in a Browser".

16.9.1 What You May Need to Know About Oracle WSM Policies

Oracle WSM policies can be attached to JAX-WS web services at the port-level. JDeveloper currently supports Oracle WSM security policies only.

JDeveloper is preconfigured to use the policy store set at the default location in the WS Policy page of the Preferences dialog at:

- **Tools menu > Preferences > WS Policy Store**

  or

- **Application menu > Application Properties > WS Policy Store**

You can specify another policy store location to use your organization's custom Oracle WSM policies. For more information Section 16.9.8, "How to Use a Different Oracle WSM Policy Store".

For more information about Oracle WSM policies, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

Policy Annotations

You can attach a single policy using the `weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` annotation in the Java class, for example:

```
@SecurityPolicy(uri = "oracle/wss11_message_protection_service_policy")
```

You can attach multiple policies as a composite using @SecurityPolicies containing a number of `@SecurityPolicy` elements, for example:

```
@SecurityPolicies( {
   @SecurityPolicy(uri = "oracle/wss_http_token_service_policy"),
   @SecurityPolicy(uri = "oracle/wss_oam_token_service_policy")
} )
```

Note: To display a list of valid policies, click **Ctrl+Alt+Enter** to invoke the Code Assist feature.

16.9.2 What You May Need to Know About Oracle WebLogic Web Service Policies

Oracle WebLogic web service policies can be attached to JAX-WS and JAX-RPC web services at the port or operation level. With Oracle WebLogic web service policies it is possible to specify the usage direction of the policies, i.e., to be applied on the inbound (request) message or outbound (response) message, or both.

You can configure JDeveloper to use your organization's custom Oracle WebLogic web service policies. For more information, see Section 16.9.7, "How to Use Custom Web Service Policies".

For more information, see the *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

Policy Annotations

You can attach a single policy using the `weblogic.jws.Policy` annotation in the Java class, for example:

```
@Policy(uri = "policy:Wssp1.2-2007-Https-UsernameToken-Plain.xml")
```

You can attach multiple policies as a composite using `@Policies` containing a number of `@Policy` elements, for example:

```
@Policies( {
   @Policy(uri = "policy:Wssp1.2-2007-Https-BasicAuth.xml"),
   @Policy(uri = "policy:Wssp1.2-2007-Https-UsernameToken-Plain.xml")
} )
```

Note: To display a list of valid policies, click **Ctrl+Alt+Enter** to invoke the Code Assist feature.

16.9.3 How to Attach Policies to Web Services

JDeveloper allows you to attach Oracle Web Service Manager (Oracle WSM) policies or Oracle WebLogic web service policies to web services.

After you attach a policy to a web service, you need to configure the policies. For more information, see "Configuring Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

You can attach policies to web services by:

- Selecting the policies to attach in the web service wizard when creating a new web service or in the web service editor when updating a web service that already exists.
- Adding policy annotations directly in the Java class; the Code Insight feature can help you. For more information, see Section 16.2.7, "How to Work with Web Services Code Insight".
- Using the Property Inspector.

To attach policies in the web service wizard or editor:

In the Create Java Web Service wizard or web service editor, navigate to the Configure Policies page. For more information at any time, press F1 or click **Help** from within the dialog.

When attaching Oracle WSM policies, you can view more information about the policy and its assertions as follows:

- Click the **Show Descriptions** checkbox to display a description of each of the policies.
- Click **View** to review the policy assertions in the policy file.
- Click the **Show Selected Policies** checkbox to display only those policies that are currently selected.

To attach policy annotations in the Java class:

1. Open the web service class in the source editor.
2. You can use the Code Insight to help you.

   Start typing the annotation, for example, `@Policies`. When you pause, or click **Ctrl+Shift+Space**, a popup appears from which you can choose the correct entry to complete the statement.

For more information about using policy annotations, see "Updating the JWS File with @Policy and @Policies Annotations" and "SecurityPolicy and SecurityPolicies

Annotations" in *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

To attach policies in the Property Inspector:

1. With the web service class open in the source editor, choose **View > Property Inspector** to open the Property Inspector.

   For more information at any time, press F1 or click **Help** from within the Property Inspector.

2. With the cursor in the `public class` or `@WebService` line of the class, navigate to the Web Services Extensions node where you can choose to use Oracle WSM Policies or Oracle WebLogic web service policies.

3. Select **Secure with OWSM Policies** or **Secure with WLS Policies**.

   The Property Inspector is refreshed to display options to select single or multiple policies for the policy type selected (Oracle WSM or WLS).

4. Click **...** to attach multiple policies from the Edit Property: Multiple Policies dialog, or select a single policy from the **Single Policy** list.

   When using the Edit Property: Multiple Policies dialog box to attach multiple Oracle WSM policy files, click **View** to review the policy assertions in the policy file.

   You cannot use both types of policy in the same web service. If you choose the wrong type, delete the lines containing the policy statements from the JAX-WS class so that you can choose again.

16.9.4 How to Attach Oracle WSM Policies to Web Service Clients

JDeveloper allows you to attach Oracle Web Service Manager (Oracle WSM) to web service clients.

After you attach an Oracle WSM policy to a web service client, you need to configure the policies. For more information, see "Configuring Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

> **Note:** For information about updating client applications to invoke web services that use WebLogic web service policies, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

You can attach Oracle WSM policies to web service clients by:

- Selecting the Oracle WSM policies to attach in the Create Web Service Client and Proxy wizard when creating a new web service client or in the Web Service Client and Proxy editor when updating a web service client that already exists. In the Create Web Service Client and Proxy wizard or editor, navigate to the Policy page.
- When attaching Oracle WSM policies, you can view more information about the policy and its assertions as follows:
  - Click the **Show Descriptions** checkbox to display a description of each of the policies.
  - Click **View** to review the policy assertions in the policy file.

  - Click the **Show Selected Policies** checkbox to display only those policies that are currently selected.
  - Click the **Show only the compatible client policies for selection** checkbox to view the policies that are compatible with the associated web service.

    For more information at any time, press F1 or click **Help** from within the dialog.

- Manually using `weblogic.wsee.jws.jaxws.owsm.SecurityPolicyFeature` class to attach a single policy or `weblogic.wsee.jws.jaxws.owsm.SecurityPoliciesFeature` to attach multiple policies.

  For more information, see the *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

16.9.5 How to Invoke Web Services Secured Using WebLogic Web Service Policies

When creating or editing a web service client from a WSDL that advertises a WebLogic web service policy, you can configure credentials to invoke the web service.

To configure credentials for a web service client that invokes a web service secured using WebLogic web service policies:

1. Perform one of the following tasks:
   - Create a web service client. For more information, see Section 16.4.1, "How to Create the Client and Proxy Classes".
   - Edit a web service client. For more information, see Section 16.4.6, "How to Manage the Web Service Clients".
2. Navigate to the **Select Credential** page of the wizard.
3. Select an existing set of credentials from the dropdown list or click **New** to define a new set of credentials.

   For help in completing the wizard, press F1 or click **Help** from within the wizard.
4. Complete the wizard.

The client class is updated to include methods for setting the client credentials. Once added, you can modify the credential values, as required.

The following provides an example of the code that is generated and included in the client class:

```
@Generated("Oracle JDeveloper")
public static void setPortCredentialProviderList(
      Map<String, Object> requestContext) throws Exception
{
   // Values used from credential preference: TestCredential
   String username = "weblogic";
   String password = "weblogic1";
   String clientKeyStore = "/C:/temp/ClientIdentity.jks";
   String clientKeyStorePassword = "ClientKey";
   String clientKeyAlias = "identity";
   String clientKeyPassword = "ClientKey";
   String serverKeyStore = "/C:/temp/ServerIdentity.jks";
   String serverKeyStorePassword = "ServerKey";
   String serverKeyAlias = "identity";
   List<CredentialProvider> credList = new ArrayList<CredentialProvider>();
```

```
    // Add the necessary credential providers to the list
    credList.add(getUNTCredentialProvider(username, password));
    credList.add(getBSTCredentialProvider(clientKeyStore, clientKeyStorePassword,
       clientKeyAlias, clientKeyPassword, serverKeyStore,
       serverKeyStorePassword, serverKeyAlias, requestContext));
       credList.add(getSAMLTrustCredentialProvider());
       requestContext.put(WSSecurityContext.CREDENTIAL_PROVIDER_LIST, credList);
}
```

For information about how to program your web service client to invoke a web service that is secured using WebLogic web service policies, see "Updating a Client Application to Invoke a Message-Secured Web Service" in the *Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server*.

16.9.6 How to Edit and Remove Policies from Web Services

You can edit policies and remove them entirely from web services with either of the following:

- Web service editor
- Source editor
- Property Inspector

To change or remove policies using the web service editor:

1. Right-click the web service in the Application Navigator, and choose **Web Service Properties**.

   For more information at any time, press F1 or click **Help** from within the dialog.

2. Navigate to the Configure Policies page, where you can change the policies for the type of policies selected, change to using a different type of policies (for example, from Oracle WSM policies to Oracle WebLogic web service policies), or choose No Policies. The web services is changed when you navigate away from this page of the editor.

To change or remove policies using annotations in the Java class:

- Open the web service class in the source editor, where the Code Insight feature is available to help you. For more information, see Section 16.2.7, "How to Work with Web Services Code Insight".Add or remove the annotations, as required.

To change or remove policies using the Property Inspector:

1. With the JAX-WS web service class open in the source editor, choose **View > Property Inspector** to open the Property Inspector.

   For more information at any time, press F1 or click **Help** from within the Property Inspector.

2. With the cursor in the `public class` or `@WebService` line of the class, navigate to the Web Services Extensions node:

   - To change multiple policies, click **...** to open the Edit Property: Multiple Policies dialog.
   - To change a single policy, delete the name from the from the **Single Policy** list and choose another.

- To change from one type of policy to another, delete all the policies so that you can start again.

16.9.7 How to Use Custom Web Service Policies

You can use custom policies from within JDeveloper. The process is different based on whether you are using custom Oracle Web Service Manager (Oracle WSM) policies or Oracle WebLogic web service policies, as described in the following sections:

- Section 16.9.7.1, "Using Custom Oracle WSM Policies"
- Section 16.9.7.2, "Using Custom Oracle WebLogic Web Service Policies"

16.9.7.1 Using Custom Oracle WSM Policies

To use custom Oracle Web Service Manager (Oracle WSM) policies, perform one of the following steps:

- Add a custom policy in the default policy store location at:

  `JDEV_USER_HOME\system11.1.1.2.x.x.x\DefaultDomain\oracle\store\gmds`. If not set,

  `JDEV_USER_HOME` defaults to `C:\Documents and Settings\user-dir\Application Data\JDeveloper`.

  Within this directory, policies must be included using one of the following directory structures:

  – Predefined Oracle WSM policies: `owsm/policies/oracle/policy_file`

  – Custom user policies: `owsm/policies/policy_file`

  Note: When exporting policy files from the Oracle WSM repository for use in JDeveloper, this directory structure is not maintained. You must ensure that when adding the exported policy to the JDeveloper environment that you use the required directory structure noted above. Otherwise, the policies will not be available in the JDeveloper environment. For more information about exporting policies from the Oracle WSM repository, see "Understanding the Different Mechanisms for Importing and Exporting Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

- Specify a different policy store. Fro more information, see Section 16.9.7.2, "Using Custom Oracle WebLogic Web Service Policies".

  If you elect to use a policy store on a remote application server, you can import custom Oracle WSM policies to the MDS repository on the remote application server using Fusion Middleware Control or WLST.

  For more information about importing policies to the Oracle WSM MDS on the remote application server, see "Understanding the Different Mechanisms for Importing and Exporting Policies" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

For more information about creating custom policies, see "Creating Custom Assertions" in *Extensibility Guide for Oracle Web Services Manager*.

16.9.7.2 Using Custom Oracle WebLogic Web Service Policies

To use custom Oracle WebLogic web service policies, perform one of the following steps:

- Place the custom policy JAR in the classpath and enable the WebLogic Server `property weblogic.wsee.policy.LoadFromClassPathEnabled` to `true`.
- Place the custom policy JAR in `WEB-INF/policies` (Web application) or `META-INF/policies` (EJB).
- Place the custom policy XML file in `WEB-INF` (Web application) or `META-INF` (EJB).

To access the policies:

- When using the `@Policy` annotation, ensure that you add the `policy` prefix; for example, `policy:mypolicy.xml`.
- Click the **Add Custom Policies** button on the Configure Policies page of the Java Web Service Editor and select the custom policy files using the Select Custom Policy Files dialog box.

For more information about creating custom policies, see "Creating Custom Assertions" in *Extensibility Guide for Oracle Web Services Manager*.

16.9.8 How to Use a Different Oracle WSM Policy Store

The Oracle Web Service Manager (Oracle WSM) policy store is installed as part of JDeveloper. You can use a different policy store, for example, to use a shared policy store. You can use a policy store that is available on the local file store or on a remote application server.

To specify a different policy store location:

1. Choose **Tools > Preferences** to open the Preferences dialog, and navigate to the WS Policy Store page.

   For more information at any time, press F1 or click **Help** from within the Preferences dialog.

2. To specify a policy store that is in the local file store, click **File Store** and enter the location of the policy store in the Override Location text box, or click **Browse** to browse to its location.

3. To configure a policy store on a remote application server, click **App Server Connection** and select a remote application server connection from the drop-down list.

   To add a new remote application server connection, click **New**.

> **Note:** The remote application server that you select must be configured with the Oracle WSM Policy Manager. To verify that the Oracle WSM Policy Manager has been properly configured, use the following URL: http://<host>:<port>/wsm-pm/validator. Enter the username and password for the server when prompted. If the Oracle WSM Policy Manager is operational, then a list of the predefined policies is displayed with descriptions. For more information about troubleshooting the Oracle WSM Policy Manager, see "Diagnosing Problems" in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

16.10 Editing and Deleting Web Services

You can edit or delete a web service that you have created in JDeveloper, for example to change the exposed method or a file location.

To edit a web service:

1. In the Application Navigator, right-click the web service container and choose **Properties**. The reentrant web service wizard is displayed.
2. Make your changes to the web service. Click **OK**. The web service files are regenerated.

   For detailed help about completing the wizard, press F1 or click **Help** from within the wizard.

After editing the web service files, you must redeploy the web service. For more information, see Section 16.12, "Deploying Web Services".

When you edit a web service, the previously generated WSDL file is overwritten, and any changes you have made to it will be lost. If you have already deployed the web service and you edit it, you must redeploy it.

When you edit a PL/SQL web service, ensure that the database connection is present; otherwise, you will receive an error message. If you have deleted the database connection, create a new one with the same name as the original one.

To delete a web service:

- In the Application Navigator, right-click the web service container and choose **Delete Web Service**. The Delete Web Service dialog listing the files that will be deleted is displayed. Click **OK**.

  The files are deleted and any references to the service web.xml are removed.

When you delete a web service from JDeveloper, the web service container and the files it contains (a WSDL file and possibly some interfaces) are deleted. The entries for the web service in `web.xml` are removed, although the file is not deleted. The `WebServices.deploy` file is unchanged as it may be used for other web services.

16.11 Testing and Debugging Web Services

Developer provides a number of ways that you can test web services. You can use the debugger, which enables you to debug web services that you create locally, on the Integrated WebLogic Server, and remotely, on Oracle WebLogic Server. You can also run a web service deployed to Integrated WebLogic Server in a browser to check that it returns what you expect.

In addition to the topics described in this section, you can use HTTP Analyzer to examine the content of web services over HTTP, similar to examining other packet information. For more information, see Section 16.13.4, "How to Examine Web Services using the HTTP Analyzer".

The following sections describe how to test and debug web services:

- Section 16.11.1, "How to Test Web Services in a Browser"
- Section 16.11.2, "How to Debug Web Services"

16.11.1 How to Test Web Services in a Browser

Once you have created and deployed a web service, you can check that it returns what you expect by running it in the browser.

The process that you use to test web services depends on whether you are testing WebLogic web services or Oracle Infrastructure web services, such as ADF business components.

- Testing WebLogic Java EE web services in a browser
- Testing Oracle Infrastructure web services in a browser

To test WebLogic Java EE web services in a browser:

1. Open the following URL in a browser: `http://IP_address:port/wls_utc`
2. Enter the URL of the WSDL and click **Test.**

   For example: `http://IP_address:port/Project1-context-root/MyWebService1?WSDL`

   The browser shows a simple page which lists the operations available on the service.
3. Enter values for each of the parameters and click the **operation-name** button to review the request details.

Testing Oracle Infrastructure Web Services in a Browser

For Oracle Infrastructure web services, such as ADF business components, you can test web services in a browser deployed to:

- Integrated WebLogic Server
- Oracle WebLogic Server

To test a service deployed to Integrated WebLogic Server:

1. When you deploy the Oracle Infrastructure web service to Integrated WebLogic Server, examine the contents of the log window. Find the line containing the following:

   ```
   Use the following context root(s) to test your web
   application(s): http://IP_
   address:port/Project1-context-root/MyWebService1
   ```
2. Copy the URL and paste it into browser. The browser shows a simple page which lists the operations available on the service.
3. Enter a parameter, and click **Enter**. The result from the web service is displayed.

To test a service deployed to Oracle WebLogic Server:

1. When you deploy the Oracle Infrastructure web service to Oracle WebLogic Server, examine the contents of the log window. Find the line that says:

   ```
   The application can be accessed at location: http://IP_
   address:port/Project1-context-root
   ```

   This URL only shows the context root for the web service.
2. Copy the URL and paste it into browser, and add the name of the web service to the end to give the full location of the service:

   ```
   http://IP_address:port/Project1-context-root/MyWebService1
   ```

The browser shows a simple page which lists the operations available on the service.

3. Enter a parameter, and click **Enter**. The SOAP message containing the parameter you entered to the web service is displayed.
4. Click **Invoke**. The result from the web service is displayed.

16.11.2 How to Debug Web Services

The debugging tools allow you to debug web services created using the web service wizards. This is similar to debugging Java programs; you can debug a web service locally or remotely by running a client against the service in debug mode. You set breakpoints in the client, which is the proxy to the web service, to investigate the functionality of the service.

Although you can debug a PL/SQL web service, what you are debugging is the Java class generated by JDeveloper to wrap the PL/SQL for deployment as a web service. Therefore, the correct way to ensure that a PL/SQL web service runs as expected is to debug the PL/SQL before you create a web service from it. For more information, see Section 29.3, "Debugging PL/SQL Programs and Java Stored Procedures".

You can use the HTTP Analyzer to examine and monitor HTTP request and response packets. It acts as a proxy between code in JDeveloper and the HTTP resource that the code is communicating with, and helps you to debug your application in terms of the HTTP traffic sent and received. For more information, see Section 16.13.4, "How to Examine Web Services using the HTTP Analyzer".

JDeveloper lets you debug a web service that is running in the Integrated WebLogic Server, locally or a web service that is deployed remotely.

Debugging Web Services Locally

Once the web service is running in Integrated WebLogic Server, you can create a proxy client to the web service. This client contains methods to run against each exposed method in the web service, and you can add your own code and set breakpoints to examine how the web service runs.

You can quickly debug a web service created in JDeveloper by debugging it locally. There are two ways to do this:

- By putting breakpoints in the web service class, then running a proxy client against it. This allows you to debug the service class itself.
- By putting breakpoints in the client.

Before locally debugging a web service, you should turn off the proxy settings. Remember to turn the proxy settings back on when you have finished debugging.

To debug a web service locally:

1. First, turn off the proxy settings. Choose **Tools > Preferences**, and select **Web Browser and Proxy**.
2. Deselect **Use Http Proxy Server**.
3. Run the web service in debug mode. In the navigator, right-click the web service container, and choose **Debug**.

   The Integrated WebLogic Server is started in debug mode, and the web service is deployed to it. The results are displayed in the log window.

4. Create a web service client, as described in "Creating Web Service Clients" on page 16-14.

   A proxy container is generated and displayed in the navigator, with a Java class called `web_serviceSoapHttpPortClient.java` displayed in the source editor.

5. In the source editor, navigator to `// Add your own code here`, and enter some code.

6. If you are debugging the client to the web service, add one or more breakpoints, right-click and choose **Debug**.

   Alternatively, if you have set breakpoints in the web service class, choose either **Debug** or **Run** from the context menu.

   The debugger operates as for any Java class. For more information, see Chapter 19, "Running and Debugging Java Programs".

Debugging Web Services Remotely

JDeveloper lets you debug a web service that is deployed remotely.

The web service could be running on Oracle WebLogic Server on the local machine, or it could be running on a service located on a remote machine. In either case, you will need a connection to the server, and the server must be running in debug mode.

When you remotely debug a web service, you have to start the server in debug mode, deploy the web service to it. You can then create a client to the service and set breakpoints in it, and run the client in debug mode.

To debug a web service remotely:

1. Run the remote server in debug mode.
2. Deploy the web service. For more information, see Section 16.12, "Deploying Web Services".
3. Create a client to the web service.
4. In the source editor, navigate to `// Add your own code here`, and enter some code.
5. Add one or more breakpoints, right-click and choose **Debug**.

   The debugger operates as for any Java class. For more information, see Chapter 19, "Running and Debugging Java Programs".

16.12 Deploying Web Services

JDeveloper provides tools that help you create and deploy web services to Oracle WebLogic Server, where they run within a Java EE container. You can:

- Deploy web services to Integrated WebLogic Server. See Section 16.12.1, "How to Deploy Web Services to Integrated WebLogic Server".
- Deploy web services to Oracle WebLogic Server. See Section 16.12.2, "How to Deploy Web Services to Oracle WebLogic Server".
- Deploy web services to an archive file. For more information, see Section 9.4.1, "Deploying to a Java JAR".
- Undeploy a web service. See Section 16.12.3, "How to Undeploy Web Services".

In addition, you can define a different WebLogic Server domain to be the Integrated WebLogic Server on which to run web services. For more information, see Section 9.2, "Running Java EE Applications in the Integrated Application Server".

16.12.1 How to Deploy Web Services to Integrated WebLogic Server

You can deploy web service generated in JDeveloper to Integrated WebLogic Server.

To deploy a web service to Integrated WebLogic Server:

1. In the Application navigator, right-click the project containing the web service, and choose **Deploy > Web Services**.

   The first time you start Integrated WebLogic Server by running or debugging an application or web service, a dialog is displayed where you enter a password for the administrator ID on the default domain. When you click **OK**, the default domain is created. You only need to do this once.

2. In the Deploy Web Services dialog, on the Deployment Action page, select **Deploy to Application Server** and click **Next**.

3. On the Select Server page, select **IntegratedWebLogicServer** and click **Next** to view the Summary page or **Finish** to deploy the web services.

16.12.2 How to Deploy Web Services to Oracle WebLogic Server

You can deploy a web service generated in JDeveloper to Oracle WebLogic Server.

When you used one of the Create Web Services wizards to generate the files for your Java EE web service, the wizard automatically created all the files that you need, including a deployment profile named `WebServices.deploy`. The `WebServices.deploy` file is created at project level. The deployment profile contains a WAR file and an EAR file.

Note: If you are deploying a PL/SQL web service, you must create an EAR, which is a deployment profile at application level, and deploy the EAR file. The database connection information is contained in the EAR, although password indirection is used so you also have to set a JDBC data source on Oracle WebLogic Server.

To deploy a web service:

- In the navigator, right-click the project containing the web service and choose **Deploy to > connection**. From the list of available connections choose the application server connection that you specified when you created the web service.

To deploy a PL/SQL web service:

1. Create an application-level EAR deployment profile and add the web service to it.
2. Deploy the EAR to the application server connection to Oracle WebLogic Server.
3. Set the database connection details on Oracle WebLogic Server by following the information about deploying EARs to Oracle WebLogic Server.

For more information, see Section 9.4.1, "Deploying to a Java JAR".

To examine the contents of a web services deployment profile:

1. To examine the contents of a web services deployment profile:
2. Choose the **File Groups > WEB-INF/classes > Filters** node to display a listing of the `.java` and `.wsdl` files for the web service.

16.12.3 How to Undeploy Web Services

If you have deployed the web service to Integrated WebLogic Server you do not need to undeploy it as the integrated server resets itself to the new application and project whenever it is started.

If you have deployed the web service to a server using an application server connection, you can undeploy it from the Resource Palette.

To undeploy a web service:

1. In the Application Server Navigator, select the application server connection you have been using and expand **Web Services**.
2. Right-click `application-name_project-name_ws` and choose **Undeploy**.

16.13 Monitoring and Analyzing Web Services

You can analyze web services in a number of ways, for example to check whether they conform to WS-I Basic Profile 1.1, or to investigate the contents of SOAP packets.

The Web Services-Interoperability Organization (WS-I) was formed by Oracle and other industry leaders to promote the interoperability of web services technologies across a variety of platforms, operating systems, and programming languages. JDeveloper provides tools that allow you to test the interoperability of web services by checking that the services conform to the WS-I Basic Profile 1.1. For more information about WS-I, see the web site of The Web Services-Interoperability Organization (WS-I) at `http://www.ws-i.org`.

In order to monitor a web service against the WS-I Basic Profile, or analyze the log file resulting from monitoring a service, you need to have downloaded a WS-I compliant analyzer.

You can analyze a web service for conformity to WS-I standards. The service can either be one you have created that is listed in the Application Navigator, or it can be a web service that you have located using a UDDI registry that is listed in the Resource Palette. Alternatively, you can create a client and proxy classes to access a deployed web service and use the HTTP Analyzer to create a log file that you then use to analyze whether the web service conforms to WS-I standards.

In order to use a WS-I compliant analyzer to analyze a web service, you need to download one to your machine and register it with JDeveloper.

To download and register a WS-I analyzer:

1. Download and install a WS-I analyzer from `http://www.ws-i.org`.
2. In JDeveloper choose **Tools > Preferences** and select **WS-I Testing Tools**.
3. Enter details of where your WS-I compliant analyzer is installed.

   For detailed help in using this dialog, press F1 or click **Help** from within the dialog.

The following sections describe how to monitor and analyze web services:

16.13.1 How to Analyze Web Services in the Navigator

You can produce a report of a web service that is listed in the Application Navigator, or that you have located using a UDDI registry and that is listed in the Resource Palette to see whether it conforms with WS-I Basic Profile 1.1 standards. Before you can do this you must have downloaded a WS-I compliant analyzer to your machine and registered it with JDeveloper.

The parts of the WS-I Basic Profile that check the content of messages sent between a web service and a client cannot be used until the client is run against the service. When invoked from the navigator, the WS-I analyzer can only analyze the description of the service in its WSDL document.

To analyze a web service:

1. With the web service selected in the navigator, choose **WS-I Analyze WSDL** from the context menu.
2. The WS-I Analyze Web Service wizard is displayed.

   For detailed help in using the wizard, press F1 or lick **Help** from within the wizard.
3. Once the wizard has run, a report of the analysis called `wsi-report.html` is displayed in JDeveloper. The report may take a few moments to appear, depending on whether you are analyzing a local web service or one deployed elsewhere on the Web.

16.13.2 How to Create and Analyze Web Service Logs

You can use the HTTP Analyzer to produce a log from running a web service client. Then you can use a WS-I compliant analyzer that you have downloaded and registered with JDeveloper to check whether the web service complies with WS-I standards.

Because you are running the analyzer against a client to the web service, discovery, description and messages of the service are reported on.

Note: If you are working within a firewall, make sure that the proxy server exceptions do not include the IP address of the machine on which the web service is running.

To create and analyze a web service:

1. Create a client to the web service you want to analyze.
   - Either to an external web service.

     or
   - For a web service that you have created and deployed to Oracle WebLogic Server, create a client stub or proxy.

     or

- For a web service that you have just created in JDeveloper, ensure that the web service is running on the embedded server by selecting **Run** from the web service container's context menu. In the navigator, select **Generate Web Service Proxy** from the web service container's context menu. You need to make sure that the web service endpoint in the WSDL is exactly the same as the _endPoint variable in the generated proxy.

2. Start the Http Analyzer. Choose **View > HTTP Analyzer**, and in the monitor click the Start button.

3. Run the client. Either:
   - Select **Run** from the context menu of the client in the source editor.

     or

   - Select **Run** from the context menu of the client in the navigator.
4. Once you have received the response you expect from the web service, stop the Http Analyzer by clicking the Stop button.

5. Click the WS-I Analyzer button to launch the WS-I Analyze wizard, and follow the instructions in the wizard. The message log records the progress, and the results are displayed in the HTTP Analyzer.

16.13.2.1 What You May Need to Know About Performing an Analysis of a Web Service

There are a number of reasons why you may find you have problems when performing an analysis of a web service. Some of these are outside the scope of the JDeveloper documentation, but there are two issues you might come across:

- When the Message section of the wsi-report.html is missing all inputs
- When the Discovery section of the wsi-report.html is missing all inputs

When the Message section of the wsi-report.html is missing all inputs

This can happen when the WSDL for an external web service has an endpoint that contains the machine name in upper or mixed case, and the client generated by JDeveloper has the `_endPoint` variable with the machine name in lower case. This is similar to the case discussed in Section 16.13.3, "How to Analyze Web Services Running in the Integrated Server".

The workaround is to import the WSDL into JDeveloper so that it is listed in the navigator, then edit the WSDL so that the machine name is lower case. Then you can generate the client (and associated proxy classes) and run it with the Http Analyzer running.

To import the WSDL into the navigator:

1. Create a new WSDL document accepting the defaults.
2. Open the WSDL document in a browser. View the source of the document, and copy the XML source of the WSDL.

3. Replace the contents of the WSDL document you have just created with the source from the WSDL document of the web service you want to use.

When the Discovery section of the wsi-report.html is missing all inputs

The Discovery section of `wsi-report.html` reports on the `REGDATA` artifacts that are used by web services you locate in a UDDI registry. If you have created a report of a web service that you have not located using a UDDI registry, then it all the `Inputs` in this section of the report will be missing.

16.13.3 How to Analyze Web Services Running in the Integrated Server

The WS-I compliant analyzer correlates messages in the log file against a set of standard assertions, and in particular the `soap:address` subelement of the `service` element in the WSDL document must exactly match that specified in the wsi-log.xml messageEntry's `senderHostAndPort` or `receiverHostAndPort`, otherwise the messages will not be analyzed for WS-I compatibility.

16.13.3.1 Changing the Endpoint Address

When the web service is run in the Integrated Server (by choosing Run from the web service's context menu), and you create the log by running the Http Analyzer while running a generated client against the web service, you may need to change the web service endpoint in the WSDL or the `_endPoint` variable in the generated client before creating the log file of the client running.

To make sure the web service endpoint is the same as the _endPoint variable in the proxy:

1. Edit the WSDL document of the web service using one of the following methods:
   - Double-click the web service container in the navigator, go to the Endpoint page of the Edit Web Service dialog, and edit the **Web Service Endpoint**.
   - Select the web service container in the navigator, and double-click the WSDL document in the Structure window. Navigate to the `soap:address` subelement, and edit the endpoint.
2. Change the web service endpoint to one of the following:
   - `IP_address:integrated_port_no` (the default integrated port number is `8988`)
   - `hostname` (lower-case)
3. For JAX-RPC web services, open the `EmbeddedStub.java` file by double-clicking on it and navigate to the `_endPoint` variable. After ensuring the web service endpoint is the same as the `_endPoint` variable in the proxy you can create and analyze the web service logs. For more information, see Section 16.13.2, "How to Create and Analyze Web Service Logs".

16.13.3.2 Changing the Endpoint Address Without Modifying the WSDL (JAX-WS Only)

For JAX-WS web services, you can change the endpoint address without modifying the WSDL, as shown in the following example:

```
import java.net.URI;
import java.net.URL;
import java.util.Map;
import javax.xml.ws.BindingProvider;
```

```
import javax.xml.ws.WebServiceRef;
import project2.proxy.Hello;
import project2.proxy.HelloService;

public class HelloPortClient
{
   @WebServiceRef
   private static HelloService helloService;

   public static void main(String [] args) {
      helloService = new HelloService();
      Hello hello = helloService.getHelloPort();
      setEndpointAddress(hello, "http://some.new.addr/endpoint");
      hello.sayHello("Bob");
   }

   public static void setEndpointAddress(Object port, String newAddress) {
      assert port instanceof BindingProvider :
        "Doesn't appear to be a valid port";
      assert newAddress !=null :"Doesn't appear to be a valid address";

      //
      BindingProvider bp = (BindingProvider)port;
      Map <String, object> context = bp.getRequestContext();
      Object oldAddress = context.get(
         BindingProvider.ENDPOINT_ADDRESS_PROPERTY);
      context.put(
         BindingProvider.ENDPOINT_ADDRESS_PROPERTY, newAddress);
   }
}
```

16.13.4 How to Examine Web Services using the HTTP Analyzer

You can use the HTTP Analyzer to examine the network traffic of a client connecting to a web service. More information, see Section 8.3, "Monitoring HTTP Using the HTTP Analyzer". It allows you to:

- Observe the exact content of the request and response TCP packets of your web service.
- Edit a request packet, resend the packet, and see the contents of the response packet.

You can use the results to debug a locally or remotely deployed web service.

> **Note:** In order to use the HTTP Analyzer, you may need to amend the proxy settings. For more information, see Section 16.2.1, "How to Use Proxy Settings and JDeveloper".

To examine the packets sent and received by the client to a web service:

1. Create the web service.
2. Either run the web service in the Integrated WebLogic Server by right-clicking the web service container in the navigator and choose **Run web_service**.

   or

Deploy and run the web service on Oracle WebLogic Server. For more information, see Section 16.12, "Deploying Web Services".

3. Start the HTTP Analyzer by selecting **View > HTTP Analyzer**. It opens in its own window in JDeveloper.
4. Run the HTTP Analyzer by clicking **Start HTTP Analyzer**.

5. Run the client proxy to the web service. The request/response packet pairs are listed in the Http Analyzer.
6. To examine the content of a request/response pair highlight it in the **History** tab and then click the **Data** tab.
7. You can quickly move from one pair to the previous or the next by clicking the up Next message and Previous message buttons.

Part IV

Developing Java Applications

This part describes how to develop Java applications with Oracle JDeveloper. JDeveloper enables you to build and assemble Java applets and client applications using JavaBeans, and interactive, desktop-based GUI applications using Swing and AWT components. You can also create and run Java client applications with Java Web Start within the JDeveloper IDE.

- Chapter 17, "Getting Started with Developing Java Applications"

  This chapter describes the tools and features that JDeveloper provides to help you develop Java applications. These include the Java Source Editor, toolbar icons, and Code Insight.

- Chapter 18, "Programming in Java"

  This chapter describes how to build Java applications. It explains how to define classes on a diagram. Class members, inheritance, and composition relationships are all derived directly from the Java source code for those classes

- Chapter 19, "Running and Debugging Java Programs"

  This chapter describes how to run and debug Java programs. JDeveloper offers several techniques to monitor and control the way Java programs run. When running Java programs, JDeveloper keeps track of processes that are run and debugged, or profiled.

- Chapter 20, "Implementing Java Swing User Interfaces"

  This chapter describes how to develop Java Swing interfaces. It explains the fundamental tasks you perform as you work with components and the JDeveloper UI design tools to create a user interface.

17

Getting Started with Developing Java Applications

This chapter provides an overview of the tools and features that JDeveloper provides to help you develop Java applications.

This chapter includes the following sections:

17.1 About Developing Java Applications

JDeveloper enables you to build and assemble Java applets and client applications using JavaBeans, and interactive, desktop-based GUI applications using Swing and AWT components. You can also create and run Java client applications with Java Web Start within the JDeveloper IDE.

JDeveloper provides resources for editing, optimizing, running, and debugging Java code:

- **Editing Java Source Files** - The Source Editor supports several Java-aware editing features. For more information, see Section 18.3, "Editing Java Code".
- **Building Apache Ant -** Compile projects using the Make and Rebuild commands, or Apache Ant. For more information, see Section 18.6, "Building Java Projects".
- R**unning Java Programs** - Keep track of processes that are run, debugged, or profiled. For more information, see Chapter 19, "Running and Debugging Java Programs"
- **Debugging Java Programs -** Tools for local and remote debugging. For more information, see Section 19.6, "About the Debugger"
- **Java Beans Components** - JavaBeans Components technology lets you implement your own framework for data retrieval, persistence, and manipulation of Java objects. For more information, see Section 18.7, "Working with JavaBeans".

- **Refactoring Java Projects** - A collection of automated refactoring operations that modify code structure without altering program behavior. For more information, see Section 18.8, "Refactoring Java Projects".
- **Optimizing Application Performance -** Tools for analyzing the quality and performance of your Java code. For more information, see Section 18.9, "Optimizing Application Performance".
- **Modeling Java Classes** - Tools to visually create Java classes and interfaces, or to graphically view existing Java classes and interfaces. For more information, see Section 18.11, "Modeling Java Classes".
- **Unit Testing for JUnit** - Tools to write and run tests that verify Java code, using the open source JUnit framework. For more information, see Section 18.12, "Unit Testing with JUnit".

17.2 About the Java Source Editor

The Java Source Editor displays Java source files, and facilitates editing of Java code. The Java Source editor is a specialized form of the generic Source Editor that JDeveloper provides for editing source code across several technologies, including XML, JSP, and HTML.

In additional to the Java-specific features of the Java Source Editor, you can also use the common set of features that JDeveloper provides to enhance coding across all domains. These features are available through the context menu or the Source menu.

Double-clicking a node in the Application Navigator either opens or brings the default editor to the foreground. When a file is open in the Source Editor, its corresponding elements are displayed hierarchically in the Structure window. Double-clicking a node in the Structure window shifts the focus to the definition of that element in the Source Editor.

You can customize the behavior of the Java Source Editor by specifying preferences in the Preferences Dialog.

17.3 Understanding Java Source Editor Features

The Java Source Editor provides features to enable easier and quicker navigation through code.

17.3.1 Using Code Insight

With Java Code Insight, you can filter out information not likely to be as useful to you (such as top-level packages, imported classes, default Object methods, deprecated items) and emphasize the information that you'll want to focus on (local variables, locally declared members, overloaded methods).

You can configure member insight, the Java-specific implementation of Code Insight's completion insight, and you can choose to display deprecated members or not in Code Insight's parameter insight window.

Member insight provides you with a list of which instance and static members (fields, methods, inner classes) are accessible from a given statement context. For example, it tells you which methods you can call from any given method.

You can use Code Insight to speed up the process of writing code. Code Insight has two varieties: completion insight and parameter insight. You can enable or disable

each independently and set the delay in seconds for each to appear when the cursor is paused at an appropriate insertion point.

To invoke completion after typing the period separator or, in the default keymap, press **Ctrl+Space**. To invoke parameter insight, pause after typing an opening (the left) parenthesis or, in the default keymap, press **Ctrl+Shift+Space**. To exit either type of insight at any time, press **Esc**. Note that if you change your keymapping, these keyboard accelerators may change. You can click **QuickDoc**, located at the bottom right of the completion insight list, to display the Javadoc for the currently selected element

After a method has been completed by completion insight, the source editor automatically fills in the parameters based on the method code. You can tab between these parameters, and edit them manually or using parameter insight.he source editor will automatically add an import if it can find only one exact match for an unresolved reference to a class. You can set preferences for this feature in the Preferences Dialog.

To change Code Insight settings or to view or change accelerators, from the main menu choose **Tools** > **Preferences** to open the Preferences dialog and then navigate to the appropriate page. For more information, see Section 17.4.1, "How to Set Code Insight Options for the Java Source Editor"

17.3.1.1 Adding Annotations to Your Java Code

Use the Code Insight feature to quickly add annotations to your Java code. An annotation is used to associate information with a program element. Annotations can be used in classes, fields, methods, parameters, local variables, constructors, enumerations, and packages. To add annotations in your Java code: declare the annotation, create a function, and then add your annotations.

When you start adding an annotation, Member Insight (**Ctrl-Space**) displays a list of options (fields, members, classes) based on the statement context. Parameter Insight (**Ctrl-Shift-Space**) displays information about the annotation like the names of the elements of the annotation type, the default values, and the created values. It also highlights the element currently under the cursor in the annotation.

For more information see Section 18.5, "How to Customize Javadoc Options for the Java Source Editor."

17.3.2 Using Code Peek

You can hold down the Shift key and then hover over a variable or method to show its definition in a ghost window. This feature makes it convenient to quickly view code without moving cursor focus from your current code.

17.3.3 Using Scroll Tips

While dragging the vertical scroll bar, a small tip window appears next to the bar, revealing the methods that are visible or partially visible on screen. This enables you to more easily see what methods are in view while quickly scrolling. You can also see the name of the method whose beginning is not immediately in view.

17.3.4 Searching Incrementally

To search incrementally, from the main menu choose **Search** > **Incremental Find Forward** or **Search**, then I**ncremental Find Backward**. In the dialog that appears, begin typing. As you type, the cursor jumps to the next instance of that particular

letter combination, either forward or backward. The search does not support wildcards.

17.3.5 Using Shortcut Keys

Shortcut keys, or accelerators, are combinations of keys that you can use to navigate or to perform certain operations using the keyboard instead of the mouse. You can select from a variety of predefined keymaps or define your own accelerators.

To view or change existing accelerators, to define new accelerators, or to load preset keymaps, from the main menu choose **Tools,** then **Preferences t**o open the Preferences dialog and then navigate to the Shortcut Keys page. To view or change accelerators for the editor, select **Code Editor** from the Category list.

Note that block commenting is indicated by Toggle Line Comments. It is defined in the default keymap as **Ctrl-Shift-Slash** or **Ctrl-Slash**.

17.3.6 Bookmarking

While bookmarking code:

- You can see a list of all bookmarks you have created in a Bookmarks window. This window appears when you click the Go to Bookmark icon. This window also displays the line number and method name that contains the bookmark.
- You can create numbered bookmarks using the keyboard shortcut Ctrl-Shift-number. You can quickly navigate to that bookmark with Ctrl-number.

17.3.7 Browsing Java Source

To navigate to the source for any identifier in an open Java file, right-click on the identifier you would like to browse and choose Go to Declaration. Alternatively, you can hold down the Ctrl key and click on an identifier to navigate to its source. If the source is not available, JDeveloper will reverse-engineer the class file.

You may browse imported classes and interfaces, member fields and methods, and local variables. If you are browsing a method or constructor invocation, this declaration search will resolve the types in order to determine the correct method or constructor invocation.

For instance, the code in Example 17–1 revokes the declaration search at SetText. It brings up the source code for `javax.swing.JButton`, with the `SetText()` method displayed.

Example 17–1 Revoking the Declaration Search

```
import javax.swing.JButton
...
JButton b1 = new JButton();
...
b1.SetText: ('OK');
```

If the identifier cannot be browsed or if there is nothing at that cursor position, this search command on the context-sensitive menu will be disabled. If JDeveloper is unable to locate the appropriate location to jump to or if the identifier cannot be browsed due to access restrictions (for example, private members), the Java Source Editor's status bar will display a message indicating so.

17.3.8 Using Code Templates

Code templates are sections of pre-written code that can be conveniently inserted into source file to avoid typing it in manually. Templates can intelligently modify the inserted code to suit its surrounding code, and imports required by code templates are automatically imported. You can use shortcuts to speed up the selection of the required template.

Pressing **Ctrl+Enter** anywhere in the source file brings up a list of code templates that you can select. The templates provided in this list are contextual and only those suitable for the current location are offered. You can click **QuickDoc** on the bottom right corner of this list to see the structure of the selected code template.

If you were using the existing template for the for loop, for instance, you would type for and then (in the default keymapping) press **Ctrl+Enter**. The template would then be filled in as follows:

```
for (;;)
```

A complete list of all code templates is available in the Code Editor Help.

To edit or create code templates, or to view or change accelerators, from the main menu choose **Tools** > **Preferences** to open the Preferences dialog and then navigate to the appropriate page.

For more information, see Section 18.3.9, "How to Use Code Templates."

17.4 Setting Preferences for the Java Source Editor

You can customize the behavior of the Java Source Editor using the Preferences Dialog.

You can also use the Preferences Dialog to specify settings for the general source editing environment.

17.4.1 How to Set Code Insight Options for the Java Source Editor

You can set various Insight options to create the behavior you want.

To set the options for Code Insight as it applies to Java:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Expand the **Java** node and select **Code Insight**.
4. On the Java Insight page, set the options you want.
5. Click **OK**.

17.4.2 How to Set Comment and Brace-Matching Options for the Java Source Editor

JDeveloper enables you to set comment and brace-matching options for the Java source editor.

To set the options for Java comment and brace matching in the source editor:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Select the **Display** node.

4. On the Display page, enable or disable automatic brace matching and set the delay time.
5. Click **OK**.
6. Reopen the Preferences dialog, expand the **Code Editor** node, and select the **Java** node.
7. On the Java page, set the attributes for comments and brace matching to create the behavior that you want.
8. Click **OK**.

Note that block commenting is an accelerator function. In the default keymap, use **Ctrl+Shift+/** or **Ctrl+/** to block-comment Java code.

17.4.3 How to Enable Automatic Import Assistance for the Java Source Editor

You can view assistance that enables you to organize import statements in the Java Source Editor.

To enable assistance for automatically adding import statements:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Expand the **Java** node and select **Imports**.
4. On the Imports page, select **Enable Auto-Popup for Import Assistance**.
5. Click **OK**.

17.4.4 How to Set Import Statement Sorting Options for the Java Source Editor

You can set options to sort import statements in the Java Source Editor.

To set the options for sorting import statements:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, expand the **Code Editor** node.
3. Expand the **Java** node and select **Imports**.
4. On the Imports page, set the attributes to create the behavior that you want when sorting import statements in the editor.
5. Click **OK**.

17.5 Using Toolbar Options

The Java Source Editor displays Java source files, and facilitates editing of Java code. Icons that perform various features are located at the top of the Java Source Editor, as described in Table 17–1.

Table 17–1 Toolbar Options

Icon	Name	Description
	Quick Outline	Click to display a tree of the available methods and fields of the current class and its super classes. Clicking this icon brings up the Quick Outline window (for more information, see Section 17.6, "Using the Quick Outline Window"). This window floats just above the code and contains a tree of the available methods and fields of the current class and its super classes. You can instantly start typing in a filter field to reduce the visible items, allowing quick and easy selection for navigation to the desired place
	Code Highlight	Click to highlight all instances of the code component that the cursor is currently placed on.
	Clear All Highlighting	Click to clear all highlighting.
	Generate Accessors	Click to insert get and set methods into a class, using the Generate Accessors dialog.
	Override Methods	Click to override inherited methods for the class in focus.
	Implement Interfaces	Click to modify a target class to implement one or more interfaces, or to make a target interface extend one or more other interfaces, using the Implement Interface dialog.
	Reformat	Click to apply source formatting to your code.
{...}	Surround	Click to surround the currently selected block of text in the Java Source Editor with a coding construct, using the Surround With dialog.
	Toggle Bookmark	Click to insert or remove a bookmark on the line of code currently in focus.
	Go to Next Bookmark	Click to place the cursor at the next bookmark.
	Go to Previous Bookmark	Click to place the cursor at the previous bookmark.
	Show Selected Element Only	Click to view only one particular element in the editor. You can use this feature to tightly focus on a method, class, inner class, or field declaration. A message at the bottom of the file reminds you that the Show Selected Element mode is currently active.
	Block Coloring	Click to activate block coloring. You can use this feature to highlight blocks of code for better readability. Coloring preferences can be set using the Preferences Dialog.

17.6 Using the Quick Outline Window

Clicking the Quick Outline Toolbar icon directly to the right of the Find field brings up the Quick Outline window shown in Figure 17–1. This window floats just above the code and contains a tree of the available methods and fields of the current class and its super classes. You can instantly start typing in a filter field to reduce the visible items, allowing quick and easy selection for navigation to the desired place.

Figure 17–1 Quick Outline Window

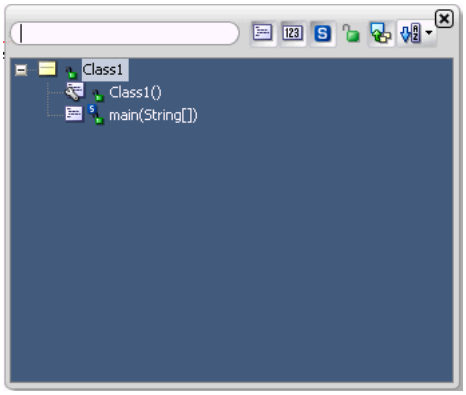


Table 17–2 lists the available icons and options.

Table 17–2 Quick Toolbar Icons

Icon	Name	Description
	Show Methods	Click to display methods and constructors. The default is methods, fields, and static members all displayed.
	Show Fields	Click to display fields. The default is methods, fields, and static members all displayed.
	Show Static Members	Click to display static members. The default is methods, fields, and static members all displayed.
	Show Public Members Only	Click to display only public members. The default value is deselected.
	Show Inherited Member	Click to display only inherited members.
	Sort Alphabetically	Click to sort class members alphabetically. The default value is deselected.
	Sort by Type	Click to sort class members first by type (in this order: constructors, methods, fields, inner classes), and then alphabetically within those categories. The default value is selected. Click the down arrow next to Sort Alphabetically to view the option.
	Sort by Access	Click to sort class members first by access modifier, and then alphabetically within those categories. The default value is deselected. Click the down arrow next to Sort Alphabetically to view the option

17.7 About the Java UI Visual Editor

The Java UI Visual Editor displays the visual components of a user interface in Editing mode. **Note**: You can use the Java Visual Editor for Swing/AWT Applications only.

When a Java Visual Editor is open, its corresponding elements are displayed hierarchically in the Structure window. If the Property Inspector is open, selecting elements in either the Structure window or the Java Visual Editor changes the selection in the Inspector as well.

The Java Visual Editor displays a GUI hierarchy. If these are menu items, the hierarchy is displayed in one fashion; if these are nonmenu items, it is displayed in another. The mode of presentation differs, as the sort of editing that you are engaged in differs. For more information, see Section 18.2.9, "How to View the Hierarchy of a Class or Interface."

When the node is selected in the Navigator, its GUI structure also displays in the Structure window. All nonmenu GUI items for this object appear under a node labeled **UI**. Any menu items appear under a node labeled **Menu**. Any non-GUI items appear under a node labeled **Other**. Once you have opened the Java Visual Editor for an object in the Navigator, to switch between the display of nonmenu GUI elements and menu elements you have only to click on a node below these UI or Menu nodes in the Structure window.

Because the display in the Java Visual Editor is rooted in the GUI hierarchy, when you click on a node (for all GUI objects) in the Structure window what loads into the editor is the visual representation of the root node and all its descendants: the entire hierarchy opens, regardless of which node in the hierarchy you selected. What displays in the editor reflects the complete GUI hierarchy; the specific element selected in the display reflects the specific node selected in the window. Selections in the Structure window and the Java Visual Editor are kept in synch.

If you have an orphan node in the Structure window, that node and its descendants comprise the entire hierarchy, with the orphan being the root of the hierarchy. The Java UI display will reflect this. If you had a control, for instance, that was not parented, and you selected the node for that control, the control and any descendants would now appear in the editor, without a container. Any changes you make to that control, however, will result in generated code.

Right-click anywhere within the Java Visual Editor to bring up a context-sensitive menu of commands. The context menus differ, depending upon whether you are editing nonmenu or menu items, and the commands available within the context menu depend on the selected object.

17.7.1 Java Swing and AWT Components

Use Swing and AWT JavaBeans components to assemble the user interface (UI) for a Java application or applet. You construct the UI in the Java Visual Editor by selecting JavaBeans from the Component Palette, such as buttons, text areas, lists, dialogs, and menus. Then, you set the values of the component properties and attach event-handler code to the component events. Tools to visually design and program Java classes to produce new compound or complex component.

For more information, see Section 20.1, "About Implementing Java Swing User Interfaces."

18

Programming in Java

This chapter describes how to use the tools and features provided by JDeveloper to build Java applications. It explains how to define classes on a diagram. Class members, inheritance, and composition relationships are all derived directly from the Java source code for those classes.

This chapter includes the following sections:

18.1 About Programming in Java

JDeveloper enables you to build and assemble Java applets and client applications using JavaBeans, and interactive, desktop-based GUI applications using Swing and AWT components. You can also create and run Java client applications with Java Web Start within the JDeveloper IDE.

JDeveloper provides resources for performing the following tasks in the Java domain:

- **Modeling Java Classes** - Tools to visually create Java classes and interfaces, or to graphically view existing Java classes and interfaces.
- **Editing Java Source Files** - The Source Editor supports several Java-aware editing features.
- **Refactoring Java Projects** - A collection of automated refactoring operations that modify code structure without altering program behavior.

- **Building Apache Ant -** Compile projects using the Make and Rebuild commands, or Apache Ant.
- **Running Java Programs** - Keep track of processes that are run, debugged, or profiled.
- **Debugging Java Programs -** Tools for local and remote debugging.
- **Optimizing Application Performance -** Tools for analyzing the quality and performance of your Java code.
- **Unit Testing with Junit** - Tools to write and run tests that verify Java code, using the open source JUnit framework.

18.2 Navigating in Java Code

JDeveloper supports Java-aware features for locating and moving to the source code for your projects' classes and interfaces and their members.

18.2.1 How to Browse Classes or Interfaces

While working in JDeveloper, you can browse Java elements directly from the UI or directly from a file open in the Java Source Editor.

To browse a Java element directly from the UI:

1. From the main menu, choose **Navigate** > **Go to Java Type**. Or, you can use the keyboard shortcut, **Ctrl-minus**.
2. In the Go to Java Type dialog, enter the name of the Java type that you want to locate. When you begin entering text in this field, a list of Java entities matching the text is displayed.
3. Double-click an entity in the list to open it in the source editor.

To browse a class or an interface for a file currently open in the editor:

1. With the file open in the editor, ensure that the editor has focus.
2. Select the class or interface name in the source file, right-click, and choose **Go to Declaration**.

   The source file opens in the Java Source Editor.

18.2.2 How to Locate the Declaration of a Variable, Class, or Method

When working in the Java Source Editor, you can quickly locate the declaration of any identifier.

To navigate to the declaration of a code element:

- Right-click on the code element and choose **Go to Declaration**, or
- Press the Control key and left-click on the code element.

The source code for that element opens, with the line on which it is declared highlighted.

18.2.3 How to Find the Usages of a Class or Interface

While working in the Java Source Editor, you can quickly locate references to a class or interface and its members. By default, usages in the current project and its dependency

projects will be reported. You can extend the search to libraries if the source files for the libraries are accessible.

The Find Usages command can also be applied to individual methods and fields, and to local variables and parameters.

To find the usages of a class:

1. Select the class or interface in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a navigator or the Structure window, select the class or interface.
2. Invoke the command using one of the following ways:
   - Choose **Search** > **Find Usages**.
   - Right-click and choose **Find Usages**.
   - Press **Ctrl+Alt-U**.

     The Usages of <*Object*> dialog will open
3. In the **Find** box select the types of references that the search will return.
4. In the **Where** box define the optional additional areas you want to search in.
5. Direct the output of the search: select **New Tab** to direct the output to a new Usages Log. If not selected, the result of the previous search for usages, if any, will be discarded.
6. Click **OK**.

   The search will commence, and the results will be displayed in the Usages Log window.

18.2.4 How to Find the Usages of a Method

While working in the Java Source Editor, you can quickly locate references to a method.

The search will show applications of the method to instances of the class or interface for which the method is defined and also for instances of its subclasses or subinterfaces, if any, that inherit the method.

The Find Usages command can also be applied to classes and interfaces, fields, and to local variables and parameters.

To find the usages of a method:

1. Select the method in one of the following ways:
   - In a Java Source Editor, select the name.
   - In the Structure pane, select the field.
2. Invoke the command in one of the following ways:
   - Choose **Search** > **Find Usages**.
   - Right-click and choose **Find Usages**.
   - Press **Ctrl+Alt-U**.

     The Usages of <*Method*> dialog displays. This dialog provides various options that you can specify to search for usages.

- Specify options in the dialog and click **OK** to begin the search.

18.2.5 How to Find the Usages of a Field

While working in the Java Source Editor, you can quickly locate references to a field.

The search will show references to the field in instances of the class or interface for which the field is defined and also for instances of its subclasses or subinterfaces, if any, that inherit the field.

The Find Usages command can also be applied to classes and interfaces, methods, and to local variables and parameters.

To find the usages of a field:

1. Select the field in one of the following ways:
   - In a Java Source Editor, select the name.
   - In the Structure pane, select the field.
2. Invoke the command in one of the following ways:
   - Choose **Search** > **Find Usages**.
   - Right-click and choose **Find Usages**.
   - Press **Ctrl+Alt-U**.
3. Complete the Usages of <*Object*> dialog and click OK. You can specify options to extend the search to other areas, define the scope of the search, and optionally specify that the results be displayed in a new tab in the Log window.

   The search will commence, and the results will be displayed in the Usages of <*Object*> Log window.

18.2.6 How to Find the Usages of a Local Variable or Parameter

While working in the Java Source Editor, you can quickly locate references to a local variable or a parameter in a method body.

The Find Usages command can also be applied to classes and interfaces, methods, and fields.

To find the usages of a local variable or parameter:

1. Select the variable or parameter name in the Java Source Editor.
2. Invoke the command in one of the following ways:
   - Choose **Search** > **Find Usages**.
   - Right-click and choose **Find Usages**.
   - Press **Ctrl+Alt-U**.

     The search will commence, and the results will be displayed in the Usages Log window.

18.2.7 How to Find Overridden Method Definitions

While working in the Java Source Editor, you can easily identify methods that override superclass definitions. Overriding definitions are marked with the **Overrides** up arrow icon in the Java Source Editor margin.

To view the overridden definition of a method, click the **Overrides** margin icon.

If you navigate to the override, then click the **Back** Main toolbar button to return to the previous view.

18.2.8 How to Find Implemented Method Declarations

While working in the Java Source Editor, you can easily identify methods that override superclass definitions. Overriding definitions are marked with the **Implements** down arrow icon in the Java Source Editor margin.

To view the overridden definition of a method, click the **Implements** margin icon.

If you navigate to the override, then click the **Back** Main toolbar button to return to the previous view.

18.2.9 How to View the Hierarchy of a Class or Interface

While working in the Java Source Editor, you can inspect the hierarchy of subtypes and supertypes of a class or interface. The Hierarchy window displays the hierarchy of the selected classes or interface.

To view the hierarchy of a class or interface in the Java Source Editor:

1. Select the class or interface, then either right-click and choose **Type Hierarchy** or choose **Navigate** > **Type Hierarchy**.

   The Hierarchy window will open (if it is not already open) and the tree of either subtypes or supertypes will be shown.

2. To toggle the display between subtypes and supertypes, click the **Subtype Hierarchy** or **Supertype Hierarchy** button.

To open a hierarchy initially in the Java Visual Editor:

1. Select a node in the Navigator.
2. Right-click and choose **Edit**, or use the **View** menu.

   The entire GUI hierarchy for the this node displays in the editor. The method of display depends upon whether this hierarchy consists of menu or nonmenu items.

18.2.10 Stepping Through the Members of a Class

You can use keyboard accelerators to step from member to member in a class definition in a Java Source Editor:

- To step to the next member definition or declaration in the current Java source view, press **Alt-Down**, or choose **Navigate** > **Go To Next Member**.
- To step to the previous member definition or declaration in the current Java source view, press **Alt-Up**, or choose **Navigate,** then **Go To Previous Member**.

These additional code-stepping commands are also defined, but are not assigned default accelerators:

- **Go to Next Class**
- **Go to Next Field**
- **Go to Next Method**
- **Go to Previous Class**

- **Go to Previous Field**
- **Go to Previous Method**

These commands are listed in the **Navigate** category of the Shortcut Keys page of the Preferences dialog. You can add or change accelerators.

18.3 Editing Java Code

JDeveloper provides many Java-aware editing features you can use to improve your productivity. As an alternative to text editing, you can also use the Java Visual Editor when developing graphical user interfaces. The Source Editor and Visual Editor are synchronized; a change in one is immediately reflected in the other. These Java editing features augment generic source editing features that support coding in any technology.

18.3.1 Editing Code with the Java Visual Editor

The Java Visual Editor displays the visual components of a user interface in Editing mode.

When a Java Visual Editor is open, its corresponding elements are displayed hierarchically in the Structure window. If the Property Inspector is open, selecting elements in either the Structure window or the Java Visual Editor changes the selection in the Inspector as well.

The Java Visual Editor displays a GUI hierarchy. If these are menu items, the hierarchy is displayed in one fashion; if these are nonmenu items, it is displayed in another. The mode of presentation differs, as the sort of editing that you are engaged in differs.

To open a hierarchy initially in the Java Visual Editor, you have only to select a node in the Navigator and then right-click and choose **Edit**, or use the View menu. The entire GUI hierarchy for the this node displays in the editor. The method of display depends upon whether this hierarchy consists of menu or nonmenu items.

When the node is selected in the Navigator, its GUI structure also displays in the Structure window. All nonmenu GUI items for this object appear under a node labeled UI. Any menu items appear under a node labeled **Menu**. Any non-GUI items appear under a node labeled Other. Once you have opened the Java Visual Editor for an object in the Navigator, to switch between the display of nonmenu GUI elements and menu elements you have only to click on a node below these UI or Menu nodes in the Structure window.

Because the display in the Java Visual Editor is rooted in the GUI hierarchy, when you click on a node (for all GUI objects) in the Structure window what loads into the editor is the visual representation of the root node and all its descendants: the entire hierarchy opens, regardless of which node in the hierarchy you selected. What displays in the editor reflects the complete GUI hierarchy; the specific element selected in the display reflects the specific node selected in the window. Selections in the Structure window and the Java Visual Editor are kept in synch.

If you have an orphan node in the Structure window, that node and its descendants comprise the entire hierarchy, with the orphan being the root of the hierarchy. The Java UI display will reflect this. If you had a control, for instance, that was not parented, and you selected the node for that control, the control and any descendants would now appear in the editor, without a container. Any changes you make to that control, however, will result in generated code.

Right-click anywhere within the Java Visual Editor to bring up a context-sensitive menu of commands. The context menus differ, depending upon whether you are editing nonmenu or menu items, and the commands available within the context menu depend on the selected object.

18.3.2 Opening the Java Visual Editor

Right-click the Java file in the Navigator that you want to modify and choose **Open** > click the **Design** tab.

The source code is accessible in the Code Editor (right-click the file in the Navigator and choose **Open** to view the source code) so you can view and edit your source code in parallel with designing your UI. Any changes made in the Java Visual Editor or Property Inspector are immediately reflected in the source code, and vice-versa.

The Java Visual Editor toolbar lets you easily work with components and duplicates commands that you can choose from the context sensitive menu displayed on a selected component. Among the component operations included are:

- Constraints to specify component weight, fill, anchor position, padding, and inset.
- Alignment to quickly position components relative to one another.
- Z-Order to change the sequence of stacked components.

18.3.3 Understanding Java Visual Editor Proxy Classes

JDeveloper is a lightweight (JFC) application. As such, using heavyweight (AWT) controls directly in the Java Visual Editor will not work as expected. Heavyweight components always obscure lightweight components, including the lightweight JDeveloper environment (including the Code Editor and UML Modelers).

The Java Visual Editor includes a proxy mechanism for registering lightweight proxies to represent heavyweight controls for instantiation in the Java Visual Editor. By default JDeveloper includes lightweight proxies for all the standard AWT controls.

18.3.4 Registering a Java Visual Editor Proxy for Custom Components

JDeveloper supports lightweight views of heavyweight components that you register for use by the Java Visual Editor.

To register the proxy class:

Add a key-value definition to `oracle.jdevimpl.uieditor.UIEditorAddin` section in the `JDeveloper\lib\addins.xml` file, as shown in Example 18–1.

Example 18–1 Key-value Definition

```
<property>
   <key>PREFIX.CLASS_NAME</key>
   <value>PROXY_CLASS_NAME</value>
</property>
```

where:

- PREFIX is `jdeveloper.concreteProxy`
- CLASS_NAME is the fully-qualified classname of the heavyweight component for which a proxy is being registered

- PROXY_CLASS_NAME is the fully-qualified classname of the proxy class to register

For example; if you were to register a hypothetical heavyweight component `implementation jdeveloper.concreteProxy.java.awt.Component` using the `oracle.jdevimpl.uieditor.proxy.Component proxy` class, the property to add would look like Example 18–2.

Example 18–2 Heavyweight Component

```
<property>
   <key>jdeveloper.concreteProxy.java.awt.Component</key>
   <value>oracle.jdevimpl.uieditor.proxy.Component</value>
</property>
```

In order for the Java Visual Editor proxy class to be available from within the IDE, so that it will appear can be added to the Component Palette, the proxy class must be added to the IDEClasspath as a directive in `JDeveloper\bin\jdev.conf` file. For example:

```
AddJavaLibFile <myUiProxies.jar>
```

where `myUiProxies.jar` contains the compiled class file for your Java Visual Editor proxy implementation.

18.3.5 How to Create a New Java Class

Before creating a new class, note that you must first create an application and a project. As soon as you create the class, it is added to the active project.

To create a new class and add it to a project:

1. In the Application Navigator, select the project you want to add the class to.
2. Right-click and choose **New**.
3. In the New Gallery, under the **General** category, select **Java**.
4. Under **Items**, select **Class**.
5. In the Create Java Class dialog, enter the class name, the package name, and the superclass that the new class will extend. Select attributes as needed.
6. Click **OK**.

   The new class appears in the active project.

18.3.6 How to Create a New Java Interface

Before creating a new interface, note that you must first create an application and a project. As soon as you create the interface, it is added to the active project.

To create a new interface and add it to a project:

1. In the Application Navigator, select the project you want to add the class to.
2. Right-click and choose **New**.
3. In the New Gallery, under the **General** category, select **Java**.
4. Under **Items**, select **Interface**.
5. In the Create Java Interface dialog, enter the interface name, the package name, and the superclass that the new class will extend. Select attributes as needed.

6. Click **OK**.

   The new class appears in the active project.

18.3.7 How to Implement a Java Interface

In the source editor, you can quickly add framework code to modify a target class to implement an interface or to make a target interface extend another interface.

An `implements` or `extends` clause is added to the declaration for the target class or interface, and an import statement is added to the file. If the target is a class, stub definitions for the implemented interface's methods are appended to the class or interface body.

To implement an interface:

1. Open a Java source file.
2. From the main menu, choose **Source** > **Implement Interface**.
3. On the Search or Hierarchy tab, locate the class that will implement the interface and select the names of the interfaces that are to be implemented.
4. If you want documentation comments from the overridden methods to be included, select **Copy Javadoc**.
5. Click **OK**.

18.3.8 How to Override Methods

In the source editor, you can quickly add stub definitions to a class to override methods inherited from superclasses.

To override methods:

1. Open a Java source file.
2. From the main menu, choose **Source** > **Override Methods**.
3. In the **Methods** list, select the methods that are to be overridden.

   The list displays methods inherited from all superclasses. Abstract methods are shown in bold type.
4. If you want documentation comments from the overridden methods to be included, select **Copy Javadoc**.
5. Click **OK**.

   The stub method definitions are added to the class.
6. Edit the stub definitions.

18.3.9 How to Use Code Templates

JDeveloper provides predefined code templates that you can use. Code templates assist you in writing code more quickly and efficiently by inserting text for commonly used statements. For example, the "Iterate over a list" (`itli`) template inserts the following code:

```
for (int i = 0; i < unknown.size(); i++) {
     Object object = (Object) unknown.get(i);
   }
```

> **Note:** If the template contains variables, the variables are highlighted. You can edit each variable to complete the template. Pressing Tab moves the caret to the next template variable.

In addition to the templates provided by JDeveloper, you can also define your own code templates in the Code Editor - Code Templates page of the Preferences dialog.

To evoke a defined code template:

1. In the file open in the editor, put the cursor at the point where the template is to be inserted.
2. Type the shortcut associated with the template and then press **Ctrl+Enter**.

   The code as defined in the template is inserted in the source file. Import statements needed for the template, if any, are inserted at the top of the file.

> **Note:** **Ctrl+Enter** is the accelerator assigned in the default keymap. You can assign an alternative.

18.3.10 Using Predefined Code Templates

This section lists the predefined code templates. The shortcut and the code that the template introduces are shown for each.

Array Iterator

```
ai
for (int $i$ = 0; $i$ < $array$.length; $i$++)
{
    $type$ $var$ = $array$[$i$];
    $end$
}
```

Data Action Event Handler

```
daev
public void on$end$(PageLifecycleContext ctx)
{
}
```

for loop

```
for
for ($end$ ; ; )
{
}
```

if statement

```
if
if ($end$)
{
}
```

if else statement

```
ife
if ($end$)
{

} else
{

}
```

integer based loop

```
fori
for (int $i$ = 0; $i$ < $lim$; $i$++)
{
   $end$
}
```

integer based loop

```
forn

int $n$ = $lim$;
for (int $i$ = 0; $i$ < $n$; $i$++)
{
   $end$
}
```

instanceof + cast

```
iofc
if ($var$ instanceof $type$)
{
     $type$ $casted$ = ($type$) $var$;
     $end$
}
```

Instantiate a BC4J application module

```
bc4jclient

String amDef = "test.TestModule";
String config = "TestModuleLocal";
ApplicationModule am =
Configuration.createRootApplicationModule(amDef,config);
ViewObject vo = am.findViewObject("TestView");
$end$// Work with your appmodule and view object here
Configuration.releaseRootApplicationModule(am,true);
```

Iterate over array

```
itar

for (int $i$ = 0; $i$ < $array$.length; $i$++)
{
     $type$ $var$ = $array$[$i$];
     $end$
}
```

Iterate over a collection

itco

```
for(Iterator $iter$ = $col$.iterator();$iter$.hasNext();)

{
     $type$ $var$ = ($type$) $iter$.next();
     $end$
}
```

Iterate over a list

itli

```
for (int $i$ = 0; $i$ < $list$.size(); $i$++)
{
     $type$ $var$ = ($type$) $list$.get($i$);
     $end$
}
```

Iterate over map keys

itmk

```
Iterator $iter$ = $map$.keySet().iterator();
while ($iter$.hasNext())
{
     $type$ $var$ = ($type$) $iter$.next();
     $end$
}
```

Iterate over map values

itmv

```
Iterator $iter$ = $map$.values().iterator();
while ($iter$.hasNext())
{
     $type$ $var$ = ($type$) $iter$.next();
     $end$
}
```

JDBC Connection

conn

```
public static Connection getConnection() throws SQLException
{
     String username = "$end$scott";
     String password = "tiger";
     String thinConn = "jdbc:oracle:thin:@localhost:1521:ORCL";
     Driver d = new OracleDriver();
     Connection conn =
DriverManager.getConnection(thinConn,username,password);
     conn.setAutoCommit(false);
     return conn;
}
```

List to array

```
ltoar

$type$ $var$ = new $typeelem$[$list$.size()];
$var$ = ($type$) $list$.toArray($var$);
$end$
```

main method

```
main

public static void main(String[] args)
{
     $end$
}
```

out.println()

```
outp

out.println($end$);
```

private ArrayList

```
pral

private ArrayList _$end$ = new ArrayList();
```

private boolean

```
prb

private boolean _$end$;
```

private HashMap

```
prhm

private HashMap _$end$ = new HashMap();
```

private int

```
pri

private int _$end$;
```

private String

```
prs

private String _$end$;
```

public static final

```
pusf
```

```
public static final $end$;
```

public static final boolean

```
pusfb

public static final boolean $end$;
```

public static final int

```
pusfi
public static final int $end$;
```

public static final String

```
pusfs

public static final String $end$;
```

Reverse array iterator

```
ritar

for (int $i$ = $array$.length; --$i$ >= 0 ;)
{
    $type$ $var$ = $array$[$i$];
    $end$
}
```

Reverse iteration over a list

```
ritli

for (int $i$ = $list$.size(); --$i$ >= 0 ; )
{
     $type$ $var$ = ($type$) $list$.get($i$);
     $end$
}
```

System.err.println

```
sep
System.err.println($end$);
```

System.out.println

```
sop

System.out.println($end$);
```

switch statement

```
sw

switch ($end$)
{
    case XXX:
       {
       }
```

```
    break;
    default;
      {
      }
    break;
}
```

try statement

```
try

try
{
   $end$
} catch (Exception ex)
{
  ex.printStackTrace();
} finally
{
}
```

Insert a tag

```
tag

<$tag$>
  $end$
</$tag$>
```

while statement

```
wh

while ($end$)
{

}
```

18.3.11 How to Expand or Narrow Selected Text

You can use the Expand/Narrow Selection option to successively expand or narrow a selected block of code, based on Java syntax.

To expand selected code:

1. With the file open in the editor, ensure that the editor has focus.
2. Put the cursor at the point where you want to expand the selection, or select a portion of the code.
3. From the main menu, choose **Source** > **Expand Selection**, or press **Ctrl+Shift+Equals**.

   The selection expands to include the smallest logical unit containing the element previously selected or within which the cursor previously resided.

With each successive application of the option, the selection expands to include the next logical step up in the Java hierarchy, based on the starting point, until the entire file is selected. For example: method name, qualified method call, assignment, definition, and so on.

Use the **Narrow Selection** option (or press **Ctrl+Shift+Minus**) to successively reduce selected code in the same fashion.

18.3.12 How to Surround Code with Coding Constructs

You can easily surround Java statements and blocks with coding constructs in the Java Source Editor.

To surround a block of code with a construct:

1. With the file open in the editor, right-click within a statement, or select a block of code, and choose **Surround**. Alternatively, you can click the **Surround** ({ }) icon the Source Editor toolbar.

   Note: This icon is only enabled when the selected code is a valid code block to which the Surround With feature can be applied.

2. In the Surround With dialog, select the coding construct.

18.3.13 Adding an Import Statement

You can add needed import statements while working in the Java Source Editor. If, as you are typing in the Source Editor, you introduce a reference to a class that has not yet been imported, a ragged line will appear below it. A popup will open showing that an import is needed, giving the fully-qualified name of the class. JDeveloper will automatically add an import if it can find only one exact match for an unresolved reference to a class.

JDeveloper will automatically add an import if it can find only one exact match for an unresolved reference to a class. If the import assistance matches more than one possible match, then a popup list appears that lists all possible matches from the class path. The user can then choose the appropriate import and the import statement is automatically added.

The import assistance popup can be triggered at any time by pressing **Alt+Enter**.

The gutter-based code assistance can be used to add an import statement. If the editor doesn't recognize a class, a lightbulb appears in the gutter when the line is highlighted and various import options are displayed.

To configure or disable Import assistance, you can set Import Statement Options in the Java Source Editor.

18.3.14 How to Organize Import Statements

You can organize import statements easily in the Java Source Editor. Set the options for organizing imports to your liking in the Preferences dialog. The following options are provided:

- Sort and group the import statements alphabetically by package and class name.
- Narrow the imports by replacing type-import-on-demand statements for packages with single-type-import statements for individual classes.
- Widen the imports by replacing two or more single-type-import statements for classes from a common package with a single type-import-on-demand statement for the package.
- Remove import statements for classes that are not referenced.

You can configure or disable import organizing options. For more information, see Section 17.4.3, "How to Enable Automatic Import Assistance for the Java Source Editor."

To organize import statements in a source file:

With the file open in the editor, right-click and choose **Organize Imports**.

18.4 Adding Documentation Comments

You can use JDeveloper's editing commands to create and maintain documentation comments.

18.4.1 How to Add Documentation Comments

You can add documentation comments to your source files in the Java Source Editor.

To add documentation comments to a source file:

- Place the cursor just above the declaration of the class, field, or method to be documented, type the start of a documentation comment (`/**`), and press **Enter**.
- With the code element selected in the Structure window, right-click and choose **Add Javadoc Comments**.

A template for the documentation comment will be inserted in the file. Add information to the template to describe the element.

18.4.2 How to Edit Documentation Comments

You can customize the use of documentation comment tags in the Java code editor. You can define custom tags, and choose which tags will be automatically included when a documentation comment is created. These choices apply to all projects.

To define a custom tag:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, choose the **Code Editor** > **Java** > **Javadoc** page.
3. Click **Add**. A tag with the default name `new` will be added to the list.
4. In the **Tag Properties** box change the name of the tag and set its other properties.
5. When finished, click **OK**.

18.4.3 How to Update Documentation Comments

You can update documentation comments in the Java Source Editor.

To update documentation comments in a source file:

1. In the Structure window, place the cursor on the element for which comments are to be updated.
2. Right-click and choose **Add Javadoc Comments**.

   Tags will be added to or removed from the documentation comment to reflect changes you have made to the element. Add descriptions for the new tags.

18.4.4 How to Audit Documentation Comments

You can validate documentation comments in your source files. The audit will report formatting errors and missing or extraneous tags.

To check documentation comments in a source file:

1. In the Application Navigator, select the file to be checked.
2. From the main menu, choose **Build** > **Audit <*filename*>**.
3. In the Audit dialog, select **Javadoc Rules** from the **Profile** dropdown list.
4. If you want configure the audit, to choose which types of errors to search for or to ignore, click **Edit**.

   The Audit Profile dialog opens.
5. Click **Run**.

   The results of the audit appear in the Log window.

18.5 How to Customize Javadoc Options for the Java Source Editor

You can add new Javadoc tags and customize the attributes of some existing tags. When creating custom tags, you can associate the tag with code elements, define it as required or not, assign it a default value, and give it an order in the tag list.

To customize the options for Javadoc in the Java Source Editor:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the **Preferences** dialog, expand the **Code Editor** node.
3. Expand the **Java** node and select **Javadoc**.
4. On the Javadoc page, select an item in the Tags list to see its attributes displayed to the right. Tags appearing in bold are customizable.
5. To add a tag, click **Add** and fill in the information.
6. To change a tag's position in the Tags list, select the tag and click the up or down button.

   To do this without using the mouse, tab to the button and press the spacebar.
7. To delete a tag, select it in the **Tags** list and click **Remove**.
8. When finished, click **OK**.

18.5.1 How to Add Documentation Comments

You can add documentation comments to your source files in the Java Source Editor.

To add documentation comments to a source file:

- Place the cursor just above the declaration of the class, field, or method to be documented, type the start of a documentation comment (`/**`), and press **Enter**.
- With the code element selected in the Structure window, right-click and choose **Add Javadoc Comments**.

A template for the documentation comment will be inserted in the file. Add information to the template to describe the element.

18.5.2 How to Set Javadoc Properties for a Project

Every project you create carries the JDeveloper project defaults or those you have supplied yourself for all the projects across workspaces. You can also replace these defaults on a project-by-project basis. Setting these properties is the same in either case: only the location, and application, of the information differs.

To set Javadoc properties for an individual project:

1. In the Application Navigator, select the project.
2. From the main menu, choose **Application** > **Project Properties**, or right-click and choose **Project Properties**.

   The Project Properties dialog opens.
3. Make changes to the project properties as required.
4. When finished, click **OK** to close the Project Properties dialog.

18.5.3 How to View Javadoc for a Code Element Using Quick Javadoc

When working in the Java Source Editor, you can quickly access Javadoc-generated documentation for the following code elements: a class, interface, or an individual member, using the Quick Javadoc feature.

The Quick Javadoc feature looks up the selected entity on the source path and displays the Javadoc comment entered in a popup window. If no Javadoc comment exists for that element, an empty Javadoc comment is displayed. The source code is available if one of the following is met:

To display Javadoc for a code element:

1. Select the code element.
2. From the main menu, choose **Source** > **Quick Javadoc**, or from within the editor, right-click and choose **Quick Javadoc**.

   A popup window displaying the documentation for the element appears. Click outside the window to close it.

The Quick Javadoc feature is available when the selected source code meets of the following criteria:

- It is on this project's source path.
- It is on the source path of a project that the current project depends on.
- It is available for a library assigned to this project
- It is a part of the JDK in use.

18.5.4 How to Preview Documentation Comments

You can preview documentation comments in your source files, in the same way that you view Javadoc for a single source element.

To display documentation comments for a given class, member, or method call:

1. Select the name of the code element.
2. Right-click and choose **Quick Javadoc**.

   A popup window showing the Javadoc for just that element now appears. From this window, you can link to other Javadoc as you would in a browser.

18.6 Building Java Projects

JDeveloper provides these facilities for building projects:

- Make /Rebuild option
- Ant
- Apache Maven

18.6.1 Building with Make and Rebuild Commands

The Make and Rebuild commands execute standard operations for compiling projects in JDeveloper.

- **Make project** makes all the projects the project depends on (recursively), and then makes the project.
- **Make project only** makes the project but not any of the projects it depends on.
- **Rebuild project** rebuilds all the projects the project depends on (recursively), and then rebuilds the project.
- **Rebuild project only** rebuilds the project but not any of the projects it depends on

18.6.1.1 Compiling with Make

Make operations compile source files that have changed since they were last compiled, or have dependencies that have changed. Rebuild operations, in contrast, compile source files unconditionally. You can invoke make on individual source files, on working sets, or on containers such as packages, projects, and workspaces.

If you wish to compile more selectively, you can add an Ant buildfile to a project, define additional targets, and run Ant to make those targets.

You cancel a compilation currently in progress by clicking the **Cancel Build** icon in the main toolbar. When you click this icon, an error message gets printed to the top row of the Compiler Log window.

Ways to make source file(s):

- In a file's source editor window, right-click and choose **Make**.
- Select one or more projects in the navigator, and click **Make** in the toolbar.
- Select one or more projects in the navigator, and choose a **Make** item from the **Build** menu.
- Select one or more projects in the navigator, right-click, and choose **Make**.

18.6.1.2 Compiling with Rebuild

Rebuild operations compile all the source files in a project or workspace. Unlike make operations, which recompile only those source files that have changed or have dependencies that have changed, rebuild operations are not conditional.

If you wish to compile more selectively, you can add an Ant buildfile to a project, define additional targets, and run Ant to make those targets.

You cancel a compilation currently in progress by clicking the **Cancel Build** icon in the main toolbar. When you click this icon, an error message gets printed to the top row of the Compiler Log window.

Ways to rebuild source files:

- Select one or more source files in the Navigator, right-click, and click **Rebuild** (for one file), or **Rebuild Selected** (for multiple files).
- Select one or more projects or workspaces in the navigator, and click **Rebuild** in the toolbar.
- Select one or more projects or workspaces in the navigator, and choose a **Rebuild** item from the **Build** menu.
- Select one or more projects or workspaces in the navigator, right-click, and choose **Rebuild**.

18.6.1.3 Understanding Dependency Checking

JDeveloper provides fast yet complete compiling by analyzing dependencies while building. Dependency checking results in fewer unnecessary compiles of interdependent source files, and thus accelerates the edit and compile cycle.

When you compile using JDeveloper, dependency checking is performed whenever you compile with **Make. Make** uses a dependency file that is automatically created within JDeveloper.

If you compile from the command line, you create or use a dependency file by specifying the following parameter:

```
javac -make <makedepfile>
```

18.6.1.4 How to Configure Your Project for Compiling

For each project, you can configure the Java compiler by setting options in the Project Properties. For example, you may not want the compiler may to display compiler messages such as:

Example 18–3 Compiler Messages

```
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

To configure project properties for compiling:

1. Right-click a project in the navigator and choose **Project Properties** from the context menu.

   You can also double-click a project node in the Application Navigator.

2. In the Project Properties dialog, expand the **Compiler** node.
3. Expand the **Javac** node.
4. Optionally expand the **Warnings** node.

   You can optionally check here first to see which options are turned on by default. For example, if **-Xlint:all** is turned on, all `-Xlint` warnings are turned on.

   If you don't want to display the **-Xlint:unchecked** message shown in Figure 18–3, go to the Turn Individual Message Off section of the Project Properties dialog. This allows you to turn off the display of specific Xlint messages, while continuing to display others by default.

5. Optionally expand the **Turn Individual Messages Off** node.
6. Check the **-Xlint Unchecked** checkbox.
7. Close all dialogs and recompile.

Note: If you want to have all your project files automatically saved before compiling, specify this in the Environment page of the Preferences dialog.

18.6.1.5 How to Specify a Native Encoding for Compiling

You can specify an encoding scheme to control how the compiler interprets multibyte characters. If no setting is specified, the default native-encoding converter for the platform is used.

Text characters are represented using different encoding schemes. In the Windows environment, these are code pages, whereas Java refers to them as native encodings. When moving data from one encoding scheme to another, conversion needs to be done. Since each scheme can have a different set of extended characters, conversion may be required to prevent loss of data.

Most text editors, including the JDeveloper source editor, use the native encoding of the platform on which they run. For example, Japanese Windows uses the Shift-JIS format. If the source code has been encoded with Shift-JIS and you are compiling it in a US Windows environment, you must specify the Shift-JIS encoding for the compiler to read the source correctly.

JDeveloper supports the character encoding schemes included with your currently installed J2SE.

To set the encoding option, do one of the following:

1. Within JDeveloper, select **Application** > **Project Properties**. In the Project Properties dialog, select the **Compiler** node.
2. On the command line, use the javac command with the `-encoding` option followed by the encoding name.
3. Choose an encoding name in one of the two following ways:
   - Select a name from the **Character Encoding** dropdown list.
   - Select "default" from the **Character Encoding** dropdown list to use the default encoding of your environment.

The Java SDK supported encodings are listed at `http://download.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html`

18.6.2 Compiling Applications and Projects

JDeveloper uses the Java Compiler (Javac) to compile Java source code (`.java` files) into Java bytecode (`.class` files). The resulting bytecode is the machine code for a Java Virtual Machine (JVM). Compiling a Java source file produces a separate class file for each class or interface declaration. When you run the resulting Java program on a particular platform, its JVM runs the bytecode contained in the class files.

Javac compiles the specified Java file and any imported files that do not have a corresponding class file. Unless dependency checking is specified (with the `-make` option), the compiler compiles all of the target Java files. For more information, see Section 18.6.1.3, "Understanding Dependency Checking."

When you work inside JDeveloper, the compiler used is Javac. You can adjust compiler options in the **Project Properties** > **Compiler** > **Option**.

The following command line options are supported:

-classpath *path*

The path used to find classes. It overrides the default CLASSPATH or the CLASSPATH environment variable. Directories are separated by semicolons. For example, to search for the class foo.java in the myclasses directory, you would enter the following:

```
javac -classpath c:\mydir;c:\jdeveloper\myclasses foo.java (Windows)
javac -classpath ~/mydir;/usr/jdeveloper/myclasses foo.java (UNIX)
```

If you are using the Java 2 platform (the default target is JDK 1.6), then the SYSTEM CLASSPATH is prepended to the CLASSPATH. For the above example, SYSTEMCLASSPATH would look similar to the following:

Windows

```
%JAVAHOME%\jre\lib\rt.jar; %JAVAHOME%\jre\lib\i18n.jar;
%JAVAHOME%\jre\lib\sunrsasign.jar;
%JAVAHOME%\jre\lib\jsse.jar; %JAVAHOME%\jre\lib\jce.jar;
%JAVAHOME%\jre\lib\charsets.jar;
%JAVAHOME%\jre\lib\classes; c:\mydir; c:\jdeveloper\myclasses
```

UNIX

```
$JAVA_HOME/lib/rt.jar;
$JAVA_HOME/lib/i18n.jar;
$JAVA_HOME/lib/sunrsasign.jar;
$JAVA_HOME/lib/jsse.jar;
$JAVA_HOME/lib/jce.jar;
$JAVA_HOME/lib/charsets.jar;
$JAVA_HOME/lib/classes;
~/mydir;
/usr/jdeveloper/myclasses
```

If JAVAHOME is not defined, then the JDK defined by the `SetJavaHome` in *`jdev.conf`* will be used. If there is no JAVAHOME, then the JDK in the <*`jdev_install`*>/jdeveloper/jdk will be used (if present).

If the target JDK is 1.6 (by using -target 1.6), the SYSTEM CLASSPATH is appended to the CLASSPATH. For the above example it would then look similar to the following:

Windows

```
c:\mydir;
c:\jdeveloper\myclasses
%JAVAHOME%\lib\classes.zip
%JAVAHOME%\classes
```

UNIX

```
~/mydir;
~/usr/jdeveloper/myclasses
$JAVAHOME/lib/classes.zip;
$JAVAHOME/classes
```

To change the SYSTEMCLASSPATH use option **-sysclasspath** or option **-bootclasspath**.

-sourcepath *pathlist*

A semicolon-separated list of paths used to locate required Java files.

-sysclasspath *pathlist*

A semicolon-separated list of paths used to find the system class files.

-bootclasspath *pathlist*

Equivalent to `-sysclasspath`.

-d *outdir*

The root directory of the class (destination) file hierarchy. For example:

`javac -d C:\JDeveloper\myclasses JavaBean.java` (Windows)

`javac -d ~/usr/jdeveloper/myclasses JavaBean.java` (Unix)

causes the class files for the classes defined in the `JavaBean.java` source file to be saved in the directory `C:\JDeveloper\myclasses\MyPackage`, assuming that `JavaBean.java` contains the package statement `MyPackage`.

Java files are read from the SOURCEPATH and class files are written to the CLASSPATH directory. The destination directory can be part of the CLASSPATH. The default destination matches the package structure in the source files and starts from the root directory of the source.

-deprecation:self

Detects usage of deprecated types, fields, and methods within the class they are defined in.

-encoding name

You can specify a native-encoding name (or code page name) to control how the compiler interprets characters beyond the ASCII character set. The default is to use the default native-encoding converter for the platform. For more information, see Section 18.6.1.5, "How to Specify a Native Encoding for Compiling."

For example,

`javac -encoding SJIS JavaBean.java`

compiles `JavaBean.java` and any directly imported Java files that do not have class files. Characters in all source files are interpreted as the **Shift-JIS** character set for Japanese.

-endorseddirs *pathlist*

Allows you to override the default value for java.endorsed.dirs, the default endorsed standards JDK classes provided by Sun. In pathlist, separate path names with semicolons.

-exclude *classname(s)*

This option allows you to specify class names to exclude from your build. The compiler will ignore all calls to public static void methods of the specified class(es). This is useful mainly for diagnostics where your non-production application build may contain code that will need to be compiled in the official production build. More than one class may be excluded by separating them with semicolons or specifying -exclude more than once. For example: `-exclude p1;p2;p3 -exclude p4` will exclude four classes, p1, p2, p3, and p4.

Note: This option is also supported in the JDeveloper IDE, on the **Compiler - Options** page of the Project Properties dialog.

Example 18–4 contains example code that uses the `-exclude` option.

Example 18–4 -exclude option

```
// beginning of excludeTest.java
public class excludeTest
{
 public static void main(String argv[])

{
 diag.Trace("Application is about to start");
 System.out.println("Test successful");
 diag.Trace("Application is about to end");
  }
}
class diag
{
 static void Trace(String msg)
  {
 System.out.println(msg);
 }
}
// end of excludeTest.java
```

When compiling the application in Example 18–4 without the **-exclude** option, the output is:

```
Application is about to start
Test Successful
Application is About to End
```

When compiling with the following **-exclude** option:

```
javac -exclude diag excludeTest.java
```

the output becomes:

```
Test Successful
```

and successfully ignores all calls to `diag.Trace`.

-extdirs *pathlist*

A semicolon-separated list of paths that overrides the location where the compiler looks for extensions.

-g

Generates debugging information in the class file. It is required to access local variables and other information while debugging the class.

-g:none

Forces the compiler not to generate debugging information in the class file.

-g:source,lines,vars

Generates selective debugging information in the class file.

-help
-?
Displays the options for the compiler.

-make *depfilename*
Uses the named dependency file for dependency checking. If the specified file is not found, it will be created.

-msglimit:#
Maximum number of errors and warnings written to output. Use -1 to represent no limit. The default is 1000.

-noquiet
Displays file names as they are compiled.

-nowarn:*<id>*
When specified with an argument, suppresses the warning associated with the number entered by the user.

-nowarn:486
Suppresses unused `import` statements.

-nowarn:487
Suppresses partially used `import` statements.

You can also use `-nowarn` in combination with `-warn:`

-nowarn -warn:487
to output warnings only for warning 487.

-nowrite
Compiles the program without outputting class files.

-p *packagename(s)*
Compiles all the source files found in the specified package(s)

-rebuild
Rebuilds specified files regardless of dependencies. Rebuild is assumed unless the **-make** option is used.

-recurse [*level*]
Instructs the compiler to recursively descend into directories when expanding file name specifications containing wildcards.

For example:

```
javac -recurse foo/*java
```

might be the equivalent to entering:

```
javac foo/bar/*java foo/lish/*java foo/lish/lee/*java
```

The option [level] takes an optional integer argument specifying the maximum recursive level.

For example:

```
javac -recurse 1 foo/*java
```

might be the equivalent to entering:

```
javac foo/bar/*java foo/lish/*java
```

Note that `foo/lish/lee/*java` would not be within the scope of the [level] variable.

-s *sourcefile*

Compiles the specified source file name(s).

-source {1.1|1.2|1.3|1.4|1.5|1.6}

By default, the source is compatible with J2SE 1.5, which enables J2SE assertions in the source code. You can enter -source 1.4 if you want the source compatibility to be J2SE 1.4.

-strictfp

Forces the compiler not to use extended precision for intermediate floating point calculations.

-target [1.1|1.2|1.3|1.4|1.5|1.6]

If the target is set to 1.1, the compiler compiles for JSDK 1.1. If the target is set to 1.2, the compiler compiles for Java 2 (JSDK 1.2). If the target is set to 1.3, the compiler compiles for Java 2 v1.3 (J2SE 1.3). If the target is set to 1.4, the compiler compiles for Java 2 v1.4 (J2SE 1.4).

The default target is JDK 6 (J2SE 1.6), consistent with Sun's `javac` defaults

-verbose

This option gives more information about compiling, such as which class files are loaded from where in the CLASSPATH. You get information about:

- Which source files are being compiled
- Which classes are being loaded
- Which classes are being loaded

-verbosepath

This option displays SOURCE PATH and CLASSPATH values used by the compiler.

-warn:<*id*>

This option allows you to specify warnings. You can have any number of warnings in combination with any suppressed warnings. When used with no arguments, all warnings are displayed. Two useful warnings:

-warn:486

Displays Unused Import Statement

-warn:487

Displays Partially used Import Statement.

-warningtag tag[, tag]

A list of javadoc comment tags. If listed tag occurs in source comment, a warning is output. Use a comma to separate tags names.

18.6.2.1 Compiling from the Command Line

You have two ways to compile applications (workspaces) and projects:

- Inside JDeveloper by using the various Build and Compile options on the application and project nodes
- From the command line by using `ojmake` and `ojdeploy`.

You can find both in the `jdeveloper/jdev/bin` directory.

- `ojmake` can be used for applications and projects that don't involve any deployment, for example, projects with no deployment profile defined.
- `ojdeploy` can handle the build of any application and project (including any that involve deployment). You can think of it as a super-set of `ojmake`.

You can view help for the tools simply by executing `ojmake` or `ojdeploy` on the command line. The help will display in the console.

Note: When you work from the command line, it is possible to use Javac to compile Java files, but it's not possible to build applications and projects by executing Javac manually. You must use `ojmake` or `ojdeploy`.

18.6.3 Cleaning Applications and Projects

You can clean your application or project using the Clean command. Running this command cleans the output and deploy directories in your project or application.

Running the Clean command on an application or project removes all class files, all copied resource files, and all deployed files. You can do this to ensure that there are no outdated files in the output and deploy directories. For instance, classes get renamed, moved, or deleted, and obsolete class files belonging to those classes need to be removed. Similarly, resources and deployments also get renamed, moved or deleted, and their obsolete copies in the output directory or deployment directory need to be removed. Cleaning enables you to remove previous build artifacts and start afresh.

You can run the Clean command on applications or projects.

When you clean an application:

- The content in the output and deploy directories of each of the constituent projects in the application are deleted.
- The content in the deploy directory of the application is deleted.

The content in the deploy directory of the application is deleted.

The following conditions must be satisfied for the Clean command to run successfully:

- The output directory of the project to be cleaned, or of each of the projects in the application to be cleaned, must be specified.
- The output location must be specified as a directory, and not a file.

18.6.3.1 How to Run the Clean Command

The Clean command enables you to remove artifacts left over from previous builds in order to begin a fresh build process.

To clean a project:

1. In the Application Navigator, select the project to be cleaned.
2. In the **Build** menu, select **Clean project**.
3. In the Cleaning *project* dialog, click **Yes**.

To clean an application and all its projects:

1. In the Application Navigator, select the application you want to clean.
2. In the **Build** menu, select **Clean All**.
3. In the Cleaning *application* dialog, click **Yes**.

18.6.4 How to Run Javadoc

You can generate API references and other documentation directly from the navigator, based upon the properties set for the project in the Javadoc page of the Preferences dialog. The documentation will be generated by the javadoc utility from the code and documentation comments in your files.

To run Javadoc on a package, file, or project:

1. Select the appropriate node in the navigator.
2. From the main menu choose **Build,** then **Javadoc**.

   The Javadoc is generated in the background. Information and results appear in the Log window. A link in the Log window allows you to add the `index.html` file to the project.

18.6.5 Building with Apache Ant

Apache Ant is a build tool similar in functionality to the Unix make utility. Ant uses XML formatted buildfiles to both describe and control the process used to build an application and its components. Ant supports cross-platform compilation and is easily extensible. Apache Ant is a product of the Apache Software Foundation. For more information, see the website `http://ant.apache.org/index.html`.

An Ant buildfile defines targets and dependencies between targets. A target is a sequence of programmatic tasks. When Ant is run to make a target, it first makes other targets on which it depends, and then executes the target's own tasks.

Ant is integrated into JDeveloper. Ant buildfiles can be added to or created for projects. Ant buildfiles can be edited with the XML Source Editor. Ant can be invoked from the user interface to make targets defined in buildfiles.

Ways to run Ant on buildfile targets:

- On targets in the project buildfile. A project can contain several Ant buildfiles, but one can be designated as the project buildfile. You can configure the **Run Ant on project** toolbar icon and dropdown menu to give easy access to the project buildfile's targets.

- From the Structure pane when editing an Ant buildfile. When an Ant buildfile is open in an XML source editor, its targets are listed in the structure pane. You can select these and run them.
- From external tools you define. Use the Create External Tool wizard to define menu items and toolbar buttons that make Ant targets.

18.6.5.1 Running Ant on Project Buildfile Targets

You can invoke Ant from JDeveloper's main menu and toolbar to build targets defined in the current project's project buildfile.

A project can contain several Ant buildfiles, one of which can be designated as the *project buildfile*. You can configure the **Run Ant on project** toolbar button and dropdown menu to give easy access to the project buildfile's targets.

To select and configure a project's project buildfile, go to the Ant project properties page (choose **Application** > **Project Properties**).

Ways to run Ant on targets in the project buildfile:

- From the toolbar, click **Run Ant on project**.

  Ant will make the project's designated default target.
- From the main menu, choose **Build** > **Run Ant on project.**

  Ant will make the project's designated default target.
- From the toolbar **Run Ant on project** dropdown menu, choose a target.

18.6.5.2 Using the Ant Tool in the IDE

The Ant Log window displays messages specific to the Ant build. Some features of the Ant Log window are:

- It displays messages generated by an Ant invocation to build one or more targets.
- In the Ant Log window, messages generated by Ant tasks are linked to the definitions of those tasks in the Ant buildfile, while compilation errors and warnings are linked to the source code that produced them.
- The color coding indicates the output level of messages.

18.6.6 Building and Running with Apache Maven

Apache Maven is a software project management and comprehension tool. Maven can manage a project's build, reporting and documentation from a central piece of information, the project object model (POM). You can build the project using its POM and a set of plugins that are shared by all projects using Maven, providing a uniform build system.

Maven can be extended by plugins to use a number of other development tools for reporting or the build process. For more information about Maven, see `http://maven.apache.org/index.html`.

18.6.6.1 Understanding the Project Object Model

The Project Object Model (POM) is an XML file that contains information about the project and configuration details used by Maven to build the project. The XML file contains most of the information required to build a project. Configuration information that can be specified in the POM includes the project dependencies, the plugins or goals that can be executed, and the build profiles.

For more information about the Maven Project Object Model, see:
http://maven.apache.org/index.html

18.6.6.2 How to Create a Project Object Model

Use options in the New Gallery to:

- Create a new Apache Maven POM
- Create a POM for the Application
- Generate a Maven POM from a Project

To create a POM:

1. Choose **File** > **New** to open the New Gallery.
2. In the **Categories** list, expand **General** and select **Maven**.
3. Select an option for creating a POM and click **OK**.

18.6.6.3 How to Create a Maven POM for a Project

You can create a Maven POM based on an existing project that you select in the Application Navigator. Build elements will be added for multiple source directories. Settings will be added for the Java compiler you have specified for the project.

To create a Maven POM from a project:

1. In the Application Navigator, select the project that you want to create the POM from.
2. Choose **File** > **New** to open the New Gallery.
3. In the **Categories** list, expand **General** and select **Maven**.
4. Select **Generate a Maven POM from a Project.**
5. Click **OK**.

18.6.6.4 How to Generate a Project Object Model from an Application

You generate a new Project Object Model (POM) for the selected application and optionally, a new POM for each project in the application. This generates a top level POM for the application, and a POM for each project.

To generate a POM from a application:

1. Choose **File,** then **New** to open the New Gallery.
2. In the **Categories** list, expand **General** and select **Maven**.
3. Select **Create a POM for the Application** and click **OK**.

18.6.6.5 Creating a Maven Template

You can create a:

- Maven Application template that is made of one or more Maven projects.

  This generates an application, a top level Project Object Model file (`pom.xml`) for the application, and a default `pom.xml file` for each project.

- Maven Project template.

This generates a Java project that includes a default Project Object Model file and Maven Configuration. The Project Object Model file is automatically created during project creation.

To create a JDeveloper Project with a default POM:

1. Choose **File,** then **New** to open the New Gallery.
2. Select **All Items**.
3. Select either **Maven Application** or **Maven Project**.
4. Click **OK**.

18.6.6.6 How to Run a Maven Project

Use the Application Navigator to locate and run a Maven project.

To run a Maven Project:

1. In the navigator, locate the project containing the `pom.xml` file you want to run.

   The `pom.xml` file is typically located in the Resources folder under the project.
2. Right-click on the `pom.xml` file.
3. In context menu, select **Run Goal(s)**, then select a goal.

   The list of goals that displays in the context menu is set using the Maven: Goals properties dialog.

18.6.6.7 How to Change the Maven Version

In the Maven project properties dialog, you can specify which version of Maven to use. Maven version 2.0 and above is supported.

To specify the Maven version:

1. In the Application Navigator, locate the project whose properties you want to set.
2. Right-click on the project name.
3. Select **Project Properties**.
4. Select **Maven**.
5. Enter the Maven version you want to use in the **Specify Maven Version** field.

18.6.6.8 How to Set Project Properties

Use the Maven Project Properties dialogs to specify:

- Which Maven version to use
- Location of the `settings.xml` file
- Location of the `pom.xml` file
- Additional classpath entries
- Dependencies
- Choices that appear in the **Run Goal(s)** context menu
- Java version
- Environment variables
- Maven repositories

- Command line options

To set Maven project properties:

1. In the Application Navigator, locate the project whose properties you want to set.
2. Right-click on the project name.
3. Select **Project Properties**.
4. Select **Maven**.
5. Select the name of a properties dialog.

18.6.6.9 How to Set Log Window Preferences

You can set the colors of various message text that displays in the messages log when you run a Maven project.

To set Maven preferences:

1. Select **Tools** > **Preferences**.
2. Select **Maven**.

18.7 Working with JavaBeans

JDeveloper comes with a set of ready-to-use JavaBeans on the Component Palette. You can also supplement these components by creating new JavaBeans yourself or by installing third-party ones.

JavaBeans Component technology lets you implement your own framework for data retrieval, persistence, and manipulation of Java objects. You can use JavaBeans technology to create reusable software components for building Java applets and Java client applications. In a Java EE application, applets and application clients can communicate with business-tier components directly or indirectly through web-tier components. For example, a client running in a browser would communicate with the business tier through JSP pages or servlets.

Although JavaBeans components are not considered Java EE web components according to the Java EE specification, JavaBeans components are often used to handle data flow between server components and application clients or applets on the client tier, or between server components and a database on the back end.

For more information on JavaBeans, for example, the basic notion of JavaBeans and what makes a bean, see http://download.oracle.com/javase/tutorial/javabeans/. The tutorial also contains lessons on writing a simple bean, bean properties, manipulating events and other topics.

18.7.1 Using JavaBeans in JDeveloper

JavaBeans are the Java building blocks used in the Java Visual Editor to build a program. Each JavaBean represents a program element, such as a user interface object, a data-aware control, or a system facility. You build your program by choosing and connecting these elements.

In order to speed up your UI design work in the future, create JavaBean components such as toolbars, status bars, checkbox groups, or dialog boxes that you can add to the Component Palette and reuse with no (or only minor) modifications

JDeveloper comes with a set of ready-to-use JavaBeans on the Component Palette. You can also supplement these components by creating new JavaBeans yourself or by installing third-party ones.

JavaBeans are objects in the true object-oriented programming (OOP) sense. Because they are true objects, JDeveloper components exhibit the following:

- *Encapsulation* of some set of data and data-access functions.
- *Inheritance* of data and behavior from a superclass.
- *Polymorphism*, allowing them to operate interchangeably with other objects derived from a common superclass.

Each component encapsulates some element of a program, such as a window or dialog box, a field in a database, or a system timer. Visual components must ultimately extend either `java.lang.Object` or extend some other class that derives from it such as `javax.swing.Panel`. Non-visual JavaBeans components do not have this requirement.

To be recognized and used in JDeveloper, components must conform to the JavaBeans specification.

To be useful in a program, a JavaBean must provide the means by which it can be manipulated or interact with other components. JavaBeans meet this requirement by defining properties, methods, and events.

All components have properties, methods, and events built into them. Some of the properties, methods, and events that components provide are actually inherited from ancestor classes, which means they share these elements with other components. For example, all UI components inherit a property called `background` that represents the background color of the component. Each component can also introduce its own unique properties, methods, and events. For example, the Swing Checkbox component has a property called `selected` that indicates whether or not this component initially appears checked.

18.7.2 How to Create a JavaBean

The first step in developing a bean for reuse is to create the JavaBean class. Using the Create Bean dialog, you can either create a new empty bean or extend an existing class to conform to the requirements of the JavaBeans component model.

To create a JavaBean:

1. In the Application Navigator, select the project you wish the bean to be added to.
2. From the main menu, choose **File** > **New,** or right-click and choose **New**.
3. In the New Gallery, in the **Categories** tree, expand **General** and select **Java**.
4. In the **Items** list, double-click **Bean**.
5. In the Create Bean dialog, accept the defaults, enter new values, or use the **Browse** buttons to navigate to an existing package and superclass.
6. Click **OK**.

18.7.3 How to Create a BeanInfo Class

The BeanInfo class defines a set of methods that allow bean implementors to provide explicit information about their beans. By specifying BeanInfo for a bean component,

you can hide methods, specify an icon for the toolbox, provide descriptive names for properties, define which properties are bound properties, and much more.

When you create a bean and install it on the Component Palette, in most cases you will want its properties and events to appear in JDeveloper's Inspector. If you followed the JavaBeans design and naming conventions while creating your bean, all the properties and events you defined, plus all those inherited from superclasses, appear automatically.

However, you may not use the JavaBeans design and naming conventions, or you may have existing classes that don't use them. In addition, you may not want to give the user of your bean access to every property at design time.

To handle these situations, you can create a BeanInfo class that will provide explicit information about a bean to JDeveloper, rather than having JDeveloper derive the information through automatic introspection. You create this class by extending the `SimpleBeanInfo` class.

To create a BeanInfo class using the BeanInfo dialog:

1. In the Application Navigator, select the project you wish the bean to be added to.
2. From the main menu, choose **File** > **New**, or right-click and choose **New**.
3. In the New Gallery, in the **Categories** tree, expand **General** and select **Java**.
4. In the **Items l**ist, double-click **BeanInfo**.
5. In the Create BeanInfo dialog, choose the bean you want to create a BeanInfo for. The dialog will generate the BeanInfo class name from the bean itself.
6. Accept the name of the package, or specify a different package to which the BeanInfo should be added.
7. Click **Browse** to choose a base class other than `SimpleBeanInfo` for the BeanInfo that you want to implement.
8. Click **OK** to add the new BeanInfo class to your project.

18.7.4 How to Implement an Event-Handling Method

In the Java Visual Editor, you see an event primarily as the event-handling method that must be implemented in the class that contains the component. For example, suppose you want to place a button named `button1` into a container called `Frame1`. In addition, you want something to happen when an end user clicks `button1`.

To implement the event-handling method

1. Select *button1* in the *Frame1* editor.
2. Navigate to the Event page of the Property Inspector.
3. Click to the right of `actionPerformed` (`actionPerformed` is the event generated when a button is pressed).

   This creates a default action-listener name, which you may edit.
4. Double-click the name to direct JDeveloper to create the appropriate method and take you to the method body.
5. JDeveloper switches to the `Frame1` source view and inserts an event-handling method into `Frame1` that is called when that event occurs.

   The method is called `button1_actionPerformed()` by default. The body of the method is initially empty.

6. Add code into the method to respond to the button press.

The end user sees all of the potential events from `button1` listed on the Events page of the Property Inspector. As the component writer, you are responsible for creating the component class in such a way that all the events it generates will appear in the Property Inspector. All the end user must do to use your bean is write the code that fills in the event-handling method.

18.7.5 What Happens When You Create an Event-Handling Method

Behind the scenes, JDeveloper also generates additional code in the `Frame1.java` file to handle the other aspects of event listening:

1. It generates an anonymous inner class for the action adapter that implements the `ActionListener` interface.
2. It instantiates the class in `Frame1`.
3. It registers itself as a listener for the *button1* event by calling `button1.addActionListener()`.

All of this code is visible in the source, but your primary task is to fill in the event-handling method that the action adapter calls when the event occurs.

18.7.6 Understanding Anonymous Adapters

The particular type of inner-class event adapters that JDeveloper generates by default are known as *anonymous* adapters. This style of adapter avoids the creation of a separate (named) adapter class. The resulting code is compact.

Example 18–5 contains code that is generated for an action-performed event using an anonymous adapter:

Example 18–5 Code Generated for an Action-performed Event Using an Anonymous Adapter

```
button1.addActionListener(new java.awt.event.ActionAdapter()
     public void actionPerformed(ActionEvent e) {
      button1_actionPerformed(e);
     }
}}

void button1_actionPerformed(ActionEvent e) {
 // your code to respond to event goes here
}
```

18.7.7 Understanding Standard Event Adapters

You can control how JDeveloper generates the adapter class by selecting the desired option from the Code Style page of the Project Properties dialog. The standard event adapters have only public- and package-level access, unlike anonymous adapters that have access to all variables in the scope where the adapter is declared.

Example 18–6 contains code that is generated for an action-performed event using a standard class:

Example 18–6 Code Generated for an Action-performed Event Using a Standard Class

```
// Registers the adapter as a listener to button1.
button1.addActionListener(new Frame1_button1_actionAdapter(this));
```

```
...
// Adapter class definition.
class Frame1_button1_actionAdapter extends java.awt.event.ActionAdapter {
   Frame 1 adaptee;

   Frame1_button1_actionAdapter(Frame1 adaptee) {
        this.adaptee = adaptee;
  }
public voidactionPerformed(ActionEvent e) {
   adaptee.button1_actionPerformed(e);
  }
}
void button1_actionPerformed(ActionEvent e) {
 // code to respond to event goes here
}
```

18.7.8 How to Make Standard Adapters the Default for Your Projects

JDeveloper generated the code in Example 18–5 using an anonymous inner class. Example 18–6 contains code that is generated for an action-performed event using a standard class. Both ways of using adapters provide the code to handle action-performed events, but the anonymous adapter approach is more compact.

To make standard adapters the default for your projects:

1. From the main menu, choose **Tools** > **Preferences**.
2. Select the **Java Visual Editor** node.
3. In the Event Settings group, select **Standard Adapter** as an event-handling option.
4. Choose **OK.**

Now JDeveloper will generate standard adapters (as opposed to anonymous, inner classes) for its events.

18.7.9 How to Select an Event-Handling Adapter

When creating a bean for reuse, you will want to define its events. Once you have defined its events, you will want to select an event-handling adapter.

To make standard adapters the default for your projects:

1. From the main menu, choose **Tools** > **Preferences**.
2. In the Preferences dialog, select the **Java Visual Editor** node.
3. On the Java Visual Editor page, under **Event Settings**, select **Standard Adapter** as an event-handling option.
4. Click **OK**.

Now JDeveloper will generate standard adapters (as opposed to anonymous, inner classes) for events.

18.7.10 How to Create an Event Set

JDeveloper provides the functionality to create a set of custom events and create an EventListener interface and an EventObject class to support those events.

To create an event set:

1. In the Application Navigator, select the project you wish the bean to be added to.
2. From the main menu, choose **File** > **New**, or right-click and choose **New**.
3. In the New Gallery, in the **Categories** tree, expand **General** and select **Java**.
4. In the **Items l**ist, double-click **EventSet**.
5. In the Create Event Set dialog, in the **Name** field, enter the name of the event set.
6. Specify the events you want to create, edit, or remove in the **Notifications** field.
7. Click **OK** to add the new event set classes to your project.

18.7.11 How to Create a Customizer

For complex beans, JDeveloper gives you the option of creating a customizer. A customizer allows you to modify the appearance and behavior of a bean within an application builder so it meets your specific needs. Sometimes properties are insufficient for representing a bean's configurable attributes. Customizers are used where sophisticated instructions would be needed to change a bean, and where property editors are too primitive to achieve bean customization.

- A simple customizer can edit the entire component at once. Usually a simple customizer presents a dialog box or panel that lets the user set many properties in the same step.
- A complex customizer provides an interactive interface to the component user, guiding the user through the steps to customize the component.

For a complex customizer, you might create a wizard that questions the user about how the component should be customized. Based on the user's responses, the customizer edits all affected properties of the component. Whenever a customizer generates an event informing JDeveloper that something happened, JDeveloper generates the appropriate code.

To create a customizer:

1. In the Application Navigator, select the project you wish the bean to be added to.
2. From the main menu, choose **File** > **New**, or right-click and choose **New**.
3. In the New Gallery, in the **Categories** tree, expand **General** and select **Java**.
4. In the Items list, double-click **Customizer.**
5. In the Create Customizer dialog, in the **Name** field, enter the name of the customizer.
6. If you want to add the customizer to a package other than the default package of your project, enter the package name in the **Package** field or click **Browse Packages**.
7. If you want the customizer to extend a class other than the default, enter the class name in the **Extends** field or click **Browse Classes**.
8. Click **OK** to add the new customizer class to your project.

18.7.12 How to Make a Component Capable of Firing Events

When you develop a bean, you must think of all the events that the bean should be able to generate.

To make a component capable of firing events:

1. Determine what kind of event needs to be fired, and either:
   - Select an appropriate existing event set from the AWT or JFC, or
   - Create a new event set.
2. Create event registration methods for the component.
3. Create an event notification/propagation mechanism for the event: `fire<yourEventName>Event()`
4. Call the event that is fired and call the event notification mechanism from the key points in your bean where such an event should be sent.

18.8 Refactoring Java Projects

Refactoring is an editing technique that modifies code structure without altering program behavior. A *refactoring operation* is a sequence of simple edits that transform a program's code but, taken together, do not change its behavior. After each refactoring operation the program will compile and run correctly. JDeveloper provides a collection of automated refactoring operations.

Use refactoring when you modify a program's source code to make it easier to maintain, extend, or reuse. Perform the modification as a series of refactoring steps. After each step you can rebuild and revalidate the program to ensure that no errors have been introduced.

Some examples of simple refactoring operations are:

- Renaming a method. This operation finds usages of the target method and then allows users to decide whether to replace each name occurrence.
- Duplicating a class. The definition of the class is replicated, and all occurrences of the class name in the replicated definition are replaced by the new name.
- Introducing a parameter into a method. The method definition is modified by the addition of a parameter, and each method call is modified to provide an argument of the appropriate type and value.

JDeveloper also provides more sophisticated refactoring operations such as:

- Extracting an interface from a class by deriving member declarations from selected class members.
- Pulling members of a class up into a superclass or pushing members down into a subclass by moving member definitions from one class to another.
- Extracting a class replaces a set of fields or methods with a new container object.
- Introducing a field, variable, parameter, or constant by replacing a selected expression with a reference to a new element constructed from the expression.
- Extracting a method by replacing highlighted consecutive statements with a call to a new method constructed from the statements.
- Extracting a method object to create a new method out of an existing block of code (similar to Extract Method) but moving it into a newly created inner class, converting all the local variables to fields of the class.
- Introducing a parameter object replaces a set of fields or methods with a new container object.

If the results of the refactoring operation are not as desired, you can undo the refactoring as you would any editing operation, by pressing `Ctrl+Z`.

18.8.1 Refactoring on Java Class Diagrams

If you rename or move a class using the in-place edit functionality on a diagram, the source code for the class will be refactored automatically. Renaming or moving a Java package on a diagram will automatically refactor the contents of that package.

Deleting a field, method, or inner class on a diagram will automatically apply the Delete Safely refactoring pattern. For more information, see Section 18.8.4, "How to Delete a Code Element".

To apply a refactoring pattern to a Java class, interface, enum, or member on a diagram, select the class or member on the diagram and choose the refactoring pattern from the Refactoring menu. Where a refactoring pattern is applied in this way, the appropriate dialog is displayed, including the facility to preview the results of the refactoring. For more information, see Section 18.8.6, "Refactoring Classes and Interfaces."

The following refactoring patterns are available for the Java classes, interfaces, and enums on a Java class diagram:

- Rename
- Move (applies to both single and multiple selections on the diagram)
- Duplicate
- Extract Interface
- Extract Superclass

The following refactoring patterns are available for the Java fields and methods on a Java class diagram:

- Rename
- Move
- Make Static
- Pull Members Up
- Push Members Down
- Change Method (Java methods only)

18.8.2 How to Invoke a Refactoring Operation

JDeveloper provides a wide range of automated refactoring operations that enable you to enhance code quality in such a way that it does not alter the external behavior of the code, yet improves its internal structure.

To invoke a refactoring operation:

1. Select a program element in a source editor window, navigator pane, or structure pane.
2. Right-click on the program element.
3. Choose an operation from the context menu.
4. You can also choose **Refactor** from the toolbar and select a refactoring operation from the drop-down list.

Refactoring context menus reflect the operations that are available in the current context. This is determined by the source element that was at the cursor when you right-clicked. These operations differ from element to element. Therefore, the refactoring context menus contain different items depending on where in JDeveloper you are right-clicking to display the menu.

For example, you can display different context menus containing different refactoring operations by right-clicking on:

- The structure menu
- The beginning of the line of a method
- The method's return type in the IDE
- The method's name in the IDE
- A parameter in the method's parameter list in the IDE

If the results of the refactoring operation are not what you want, you can undo the refactoring as you would any editing operation, by pressing `Ctrl+Z`.

18.8.3 How to Rename a Code Element

While developing your Java application you can easily rename the definition and all references to a package, class, interface, method, field, parameter, or variable. If you wish, you can first generate a preview — a list of the usages that will be replaced. Use the preview to inspect and modify or exclude selected usages, before causing the rest to be renamed.

The scope of a renaming operation is the full scope of the element in the project. Project, class, interface, and member usages are replaced anywhere they appear in the project. Parameters and variables are renamed only in the lexical scope of their definitions: other elements with the same name are not modified.

By default, the operation will be restricted to `.java` files, excluding comments (but not documentation comment tags that name code elements) and annotations. Usages that are not ambiguous will be replaced. Usages of class and interface names will be replaced if they are fully qualified or if they are named in import statements.

For package, type, and member elements, you can choose to extend the operation to comments or to other files. When extended to comments, replacements will be made in line comments, commented-out code, the bodies of documentation comments, and in annotations. When the operation is extended to other files, text replacements will also be made in project files of types designated as text files in the File Types page of the Preferences dialog. Replacements in comments and other files will be made more aggressively than replacements in Java code.

To rename a code element:

1. Select the element that is to be renamed in one of the two following ways:
   - In a Java Source Editor, select the name.
   - In a Navigator or Structure pane, select the name.
2. Invoke the command in one of the two following ways:
   - From the Main menu or the context menu, choose **Refactor** > **Rename**.
   - Press **Ctrl+Alt-R**.

     The Rename dialog opens.

3. In the **Rename To** box, enter the new name. The name must be valid and not already in use.
4. Set the depth of the text substitution.
   - Select **Search in Comments** to extend the operation to comments, the bodies of documentation comments, and to annotations.
   - (Package, type, and members elements only.) Select **Search in Non-Java files** to extend the operation to other types of text files in the project.
5. Select **Preview** if you wish to inspect the usages that will be replaced before committing to the renaming operation.
6. Click **OK**.

   If you selected **Preview**, to avoid all the usages being modified, finish the renaming operation from the Preview log window. For more information, see Section 18.8.3, "How to Rename a Code Element."

18.8.4 How to Delete a Code Element

While developing your Java application you can safely delete the definition of a class, interface, method, or field. The deletion will not be performed without your confirmation if the element is still in use.

If the element is in use a log showing the usages will be displayed. Use the list to inspect and resolve the usages. If you then confirm the deletion, any remaining usages will remain in the code as undefined references.

To delete a code element:

1. Select the element that is to be deleted in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a Navigator or Structure pane, select the name.
2. Invoke the command in one of the following ways:
   - From the Main menu or the context menu, choose **Refactor** > **Delete Safely**.
   - Press **Alt+Delete**.

     The Delete Safely dialog displays while the project files are searched for usages.
3. If the dialog closes, the element has been deleted. If it remains open after performing its search, the element has unresolved usages.
   - Click **View Usages** to inspect and resolve the usages. When finished, invoke the command again to delete the element.
   - Click **Ignore** to delete the element's definition without viewing the usages.

18.8.5 How to Preview a Refactoring Operation

When performing a refactoring operation that may modify many usages, it is useful to preview the usages to identify those that should be modified by hand or be excluded. You have the option, before committing these operations, of having usages listed in the Preview Log window, from which you can inspect and resolve them, and if you wish, commit the operation.

The log displays a collapsible tree of packages and Java files. Under each file, lines of code containing usages are listed.

To view a usage in an Edit window:

Double-click the entry in the log.

To exclude a usage from the refactoring operation:

Right click it and choose **Exclude**.

To commit the refactoring operation:

1. If you have made any edits that affect usages, click the **Refresh** icon in the log toolbar to rerun the usages search.
2. Click the **Do Refactoring** icon in the log toolbar.

18.8.6 Refactoring Classes and Interfaces

While developing your Java application you can easily define new classes and interfaces and repurpose existing ones.

18.8.6.1 How to Move a Package, Class, or Interface

When developing your Java application you can easily move a package, class, or interface to a different package. If you wish, you can first generate a preview — a list of the usages that will be replaced. Use the preview to inspect and modify or exclude selected usages, before completing the move.

When moving types, only primary classes and interfaces — those having the same name as their file — can be selected to be moved. In effect the file is renamed, and the definitions of secondary classes and interfaces remain with the primary. Accessibility will be preserved: if other classes in the original package refer to the class being moved, it will be given public access. If the class being moved refers to other classes in the original package, those classes will be made public.

The scope of an operation to move a class or interface is the entire project.

By default, the operation will be restricted to `.java` files, excluding comments (but not documentation comment tags that name code elements) and annotations. Usages that are not ambiguous will be replaced. Usages will be replaced if they are fully qualified or if they are named in import statements.

You can choose to extend the operation to comments or to other files. When extended to comments, text replacements will be made in line comments, commented-out code, the bodies of documentation comments, and in annotations. When the operation is extended to other files, replacements will also be made in project files of types designated as text files in the File Types page of the Preferences dialog. Replacements in comments and other files will be made more aggressively than replacements in Java code.

To move a class or interface:

1. Select the package, class, or interface that is to be moved, in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a navigator or in the Structure window, select the name.
2. Invoke the command in one of the following ways:

- From the Main menu or the context menu, choose **Refactor** > **Move**.
- Press **Ctrl+Alt-M**.

  The Move Vehicle dialog opens.

3. In the **Move To** box, enter the new package name, or click **[...]** to navigate to an existing package.
4. Set the depth of the text substitution.
   - Select **Search in Comments** to extend the operation to comments, the bodies of documentation comments, and to annotations.
   - Select **Search in Text Files** to extend the operation to other types of text files in the project.
5. Select **Preview** if you want to inspect the usages that will be replaced before committing to the move operation.
6. Click **OK**.

   If you selected **Preview**, to avoid all the usages being modified, finish the renaming operation from the Preview log window. For more information, see Section 18.8.3, "How to Rename a Code Element."

Classes can also be moved in the Application Navigator by dragging multiple classes from one package to another.

18.8.6.2 How to Duplicate a Class or Interface

While developing your Java application you can easily duplicate a class or interface.

Only primary classes and interfaces — those having the same name as their file — can be selected to be duplicated. The duplicated class or interface is added to the same package as the original.

Member names in the new class are given the same name as those in the original, except for those derived from the original class or interface name. When the original name is embedded in a member name, the new name is substituted.

To duplicate a class or interface:

1. In a Java Source Editor, select the name of the class or interface that is to be duplicated.

   Note: Only primary classes and interfaces - those having the same name as their file - can be selected to be moved.

2. From the Main menu, choose **Refactor** > **Duplicate**.

   The Duplicate *type* dialog opens.
3. In the **Class Name** box, enter the new name. You can also specify a new package.
4. Click **OK**.

   The new class will be added to the project.

18.8.6.3 How to Extract an Interface from a Class

While developing your Java application you can easily derive a new interface from selected methods and static fields defined in an existing class.

Optionally, you can also generalize declarations — such as the type specifications of parameters, variables, and members — by replacing each type name in the declaration with the new interface name. Not all such declarations can be replaced. For example, the replacement cannot be done for the declaration of a variable that is used in a method invocation, if that method was not extracted into the new interface. The replacements will be done anywhere in the project.

The declaration of the class will be modified to show that it is an implementation of the new interface.

To extract an interface:

1. Select the class from which the interface will be derived in one of the following ways:
   - In a Java Source Editor, select the class name.
   - In a navigator or in the Structure Window, select the class name.
2. From the Main menu, choose **Refactor** > **Extract Interface**.

   The Extract Interface dialog opens.
3. In the **Package** field, enter the name of the package of the new interface.
4. In the **Interface** field, enter the name of the new interface.
5. In the **Members to Extract** table, select the members that will be included in the new interface.
6. Select **Replace Usages** if you want to convert existing declarations that name the class into declarations naming the interface.
7. Select **Preview** if you wish to inspect the usages before you commit to the operation. This option is enabled only if you have selected **Replace Usages**.
8. Click OK.

   Otherwise, the interface will be created, and no usages will be replaced.

18.8.6.4 How to Extract a Superclass

You can create a superclass based on chosen members of the current class. The superclass will consist of field and method declarations that match the chosen members.

To extract a superclass:

1. In a navigator, in the Structure window, or in a Java Source Editor window, select the class name.
2. From the main menu, choose **Refactor** > **Extract Superclass**.

   The Extract Superclass dialog opens
3. In the **Package Name** box, enter the name of the package to which the new superclass will belong.
4. In the **Class Name** box, enter a name for the new superclass.
5. In the **Members to Extract** table, select the members that will be included in the new interface.

   If you want a method to be created as an abstract method in the superclass, check the **Abstract** box against that method. If you want dependencies of a method to be included in the superclass, check the **Dependencies** box.

6. Select **Replace Usages** if you wish to convert existing declarations that name the class into declarations naming the superclass.
7. Select **Preview** if you wish to inspect the usages before you commit to the operation.

   This option is enabled only if you have selected **Replace Usages**.
8. Click **OK**.

   If you selected Preview, finish the extraction operation from the Preview log window. For more information, see Section 18.8.3, "How to Rename a Code Element."

   Otherwise, the interface will be created, and no usages will be replaced.

18.8.6.5 How to Use Supertypes Where Possible

While developing your Java application you can easily generalize declarations — such as the type specifications of parameters, variables, and members — by replacing references to the selected class with references to one of its supertypes. Not all such declarations can be replaced. For example, the replacement cannot be done for the declaration of a variable that is used in a method invocation, if that method is not also defined in the supertype. The replacements will be done anywhere in the project.

To generalize declarations:

1. Select the class or interface whose declarations will be generalized in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a navigator or in the Structure window, select the name.
2. From the Main menu, choose **Refactor** > **Use Supertype Where Possible**.

   The Use Supertype dialog opens.
3. In the **Supertypes** table, select the supertype that the declarations will be generalized to.
4. Select **Preview** if you want to inspect the usages before you commit to the operation.
5. Click **OK**.

   If you selected **Preview**, finish the extraction operation from the Preview log window. For more information, see Section 18.8.3, "How to Rename a Code Element."

   Otherwise, the substitutions will be made immediately.

18.8.6.6 How to Convert an Anonymous Class to an Inner Class

You can convert an unnamed inner class (an anonymous class) into a named inner class.

To convert an anonymous class into an inner class:

1. In a Java source editor window, select the declaration of the anonymous class.
2. From the main menu, choose **Refactor** > **Convert Anonymous to Inner Class**.

   The Convert Anonymous to Inner Class dialog opens
3. In the **Class Name** box, enter the name to be given to the inner class.

4. If you want the inner class to be given the static modifier, check the **Static** box.
5. To convert the anonymous class into an inner class, click **OK**.

18.8.6.7 How to Move an Inner Class

You can move an inner class to a newly created class at the top level.

To move an inner class:

1. Select the inner class name in the structure pane or in a Java source editor window.
2. On the main menu select **Refactor** > **Move**. The Move Inner Class dialog opens.
3. If you do not want the new top level class to be created with the names already shown in the dialog, overwrite them or select new ones.
4. To create a new class at the top level with the details shown in the dialog, click **OK**.

18.8.7 Refactoring Members

While developing your Java application you can easily move member definitions from one class to another.

18.8.7.1 How to Move a Class Member

You can move a class member (for example, a method) to another class.

To move a non-static method:

1. Select the method name in the Structure window or in a Java Source Editor window.
2. From the main menu select **Refactor** > **Move**.

   If there is at least one suitable target to which the member can move, the Move Member dialog opens. Otherwise, a message box is displayed.
3. In the **Targets** panel, choose the class to which the member will be moved.
4. If you want new names to be used for the method and the parameter in the new location, enter new names into the **Method Name** and **Parameter Name** boxes.
5. Select how usages of the member will be handled after the move.
   - Select **Use Delegate** to handle usages through a newly created delegating method.
   - Select **Replace** to replace all usages with new ones that call the moved class member directly.

To move a static method:

1. Select the method name in the Structure window or in a Java Source Editor window.
2. From the main menu select **Refactor** > **Move**.

   The Move Members dialog opens.
3. In the **Target** panel, enter or choose the class to which the member will be moved.
4. For each member that you want to move, ensure that the checkbox to its left in the **Members to Extract** list is checked.

5. If you want the dependencies of a member to also be moved, check the corresponding checkbox in the Dependencies column.

18.8.7.2 How to Change the Signature of a Method

You can change the signature of a method. The signature of a method is the combination of the method's name along with the number and types of the parameters (and their order.)

To change the signature of a method:

1. Select the method name in the Structure window or in a Java Source Editor.
2. On the main menu select **Refactor** > **Change Method**.

   The Change Method dialog opens.
3. Make changes to the method name, return type, accessibility and parameters as required.

   If you change the name of the method to one that already exists in the class, you will later see a second dialog. Through this you can opt to replace all usages of the method that you are changing to usages of the existing method.
4. If you want to create tasks based on the changes you have made and add them to the Tasks window, check the **Add tasks to the task window** box.

Note: This feature does not apply to constructors.

18.8.7.3 How to Change a Method to a Static Method

You can assign the `static` modifier to a method.

To change a method to a static method:

1. Select the method name in the Structure window or in a Java Source Editor.
2. On the main menu select **Refactor** > **Make Static**.

   If the class is part of a class hierarchy, the Make Static dialog opens. Otherwise, the static modifier is added immediately.
3. If the Make Static dialog opens:
   - In the **Name** box, enter or select a name to be used as a reference in the modified method.

     The options listed are derived from local object names.
   - If you want to create a method that cannot be overridden, check the **Declare final** box.

18.8.7.4 How to Pull Members Up into a Superclass

While developing your Java application you can easily move the definitions of members from a class (the source class) to one of its superclasses (the target class). This operation can be applied to a class only if it has one or more potential target classes in the project. Members cannot be pulled up into library classes. Also, this refactor command is only available for a class that is declared with a superclass clause or a list of implemented interfaces.

By default, when a method is pulled up, its definition is moved from the source class to the target class. You can instead choose to abstract the method, in which case the

method definition will remain in the source class, and a declaration for it will be added to the target class. Abstracting a method will convert the target class to an abstract class, if it is not already.

A member that you wish to pull up may have dependencies. A member is a dependency if it is used in the definition of a member that is to be pulled up. Pulling a member up without also pulling its dependencies up will introduce undefined references in the target class. When you select a member to be pulled up, its dependencies will be indicated. You can choose whether or not to pull up the dependencies as well.

When a member declared to be private is pulled up, its access changes to protected.

To pull members up:

1. Select the class from which the members will be pulled in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a navigator or the Structure window, select the name.
2. From the main menu, choose **Refactor** > **Pull Members Up**.

   The Pull Members Up dialog will open.
3. From the **Target** drop-down menu, choose the superclass that will be the target class.
4. In the **Members to Extract** table, select the members you want to pull up.

   The members that are the dependencies of the selected members, if any, will be indicated.
5. In the **Abstract** column, select the checkbox if you wish the method is to abstracted to the target class.

   Note: Members that are to be abstracted do not have dependencies.
6. In the **Dependencies** column select the checkbox if you wish to also pull up all of the member's dependencies.

   This selection is transitive. It will cause dependencies of dependencies to also be pulled up.
7. Click **OK**.

18.8.7.5 How to Push Members Down into Subclasses

While developing your Java application you can easily move the definitions of members from a class (the source class) to its immediate subclasses (the target classes).

By default, when a method is pushed down, its definition is moved from the source class to the target classes. You can instead choose to leave a method declaration in the source class, converting it to an abstract class, if it is not already.

A member that you wish to push down may have dependencies. A member is a dependency if its definition uses a member that is to be pushed down. Pushing a member down without also pushing its dependencies down will introduce undefined references in the source class. When you select a member to be pushed down, its dependencies will be indicated. You can choose whether or not to push down the dependencies as well.

To push members down:

1. Select the class from which the members will be pulled in one of the following ways:
   - In a Java Source Editor, select the name.
   - In a navigator or the Structure window, select the name.
2. From the main menu, choose **Refactor** > **Push Members Down**.

   The Push Members Down dialog opens.
3. In the **Members to Extract** table, select the members you wish to push down.

   The members that are the dependencies of the selected members, if any, will be indicated.
4. In the **Abstract** column, select the checkbox if you wish an abstract definition of the member to be left in the source class.

   This selection is transitive. It will cause dependencies of dependencies to also be pushed down.
5. Click **OK**.

18.8.8 Refactoring Expressions

While developing your Java application you can easily convert expressions into named elements.

18.8.8.1 How to Inline a Method Call

You can incorporate the body of a method into the body of its callers and remove the original method. This is known as inlining a method call.

To inline a method call:

1. In a Java Source Editor, select an instance of the method call that you want to be inlined.
2. From the main menu select **Refactor** > **Inline**.
   - If there is only one call to the method in this class, the change is made immediately.
   - If there is more than one call to the method in this class, the Inline dialog opens.
3. If the Inline dialog has opened:
   - Choose between inlining only the selected instance of the call or inlining all instances of the call.
   - Click **OK**.

18.8.8.2 How to Introduce a Field

While developing your Java application you can easily convert an expression into a reference to a field. A new field declaration will be added to the class, and the selected expression will become its initialization. The original expression will be replaced by a reference to the new field.

An expression cannot be converted into a field if its type is `void`.

To introduce a field:

1. In the source editor, select the expression.
2. From the main menu, choose **Refactor** > **Introduce Field**.

   The Introduce Field dialog opens.
3. From the **Type** drop-down menu choose a type for the field.

   The menu lists all types that are consistent with the expression. This option will not be shown if only a single type is valid.
4. A suggested name will be shown in the **Name** text box.

   You can modify or replace it, or choose another suggestion from the drop-down menu.
5. Select an initialization:
   - Select **Current Method** to put the assignment statement for the field immediately preceding the statement that contains the expression.
   - Select **Field Declaration** to assign the value to the field in its declaration statement. This option will not be enabled if the expression has a variable or parameter with local scope.
   - Select **Constructor** to assign the value to the field in the constructor methods of the class. This option will not be enabled if the expression has a variable or parameter with local scope.
6. Click **OK**.

18.8.8.3 How to Introduce a Variable

While developing your Java application you can easily convert an expression into a reference to a variable. A new variable declaration will be added to the method, and the selected expression will become its initialization. The original expression will be replaced by a reference to the new member.

An expression cannot be converted into a member if its type is `void`.

To introduce a member:

1. In the source editor, select the expression.
2. From the main menu, choose **Refactor** > **Introduce Variable**.

   The Introduce Variable dialog opens.
3. From the **Type** drop-down menu choose a type for the field.

   The menu lists all types that are consistent with the expression. This option will not be shown if only a single type is valid.
4. A suggested name will be shown in the **Name** text box.

   You can modify or replace it, or choose another suggestion from the drop-down menu.
5. Select **Declare final** if you wish to add the final modifier to the variable's declaration.
6. Click **OK**.

18.8.8.4 How to Introduce a Parameter

While developing your Java application you can easily convert a constant expression in a method body into a new parameter for the method. The expression will be replaced by the new parameter name, the new parameter will be added to the method's parameter list, and in all invocations of the method the expression will be inserted as an additional argument.

Expressions can be introduced as parameters only if they are literals or operations on literals.

This operation is disallowed for methods that implement an interface. Altering the signature of such a method would invalidate the implementation.

To introduce a parameter:

1. In the source editor, select the expression.
2. From the main menu, choose **Refactor** > **Introduce Parameter**.

   The Introduce Parameter dialog opens.
3. From the **Type** drop-down menu choose a type for the field.

   The menu lists all types that are consistent with the expression. This option will not be shown if only a single type is valid.
4. A suggested name displays in the **Name** text box.

   You can modify or replace it, or choose another suggestion from the drop-down menu.
5. Select **Declare final** if you want to add the final modifier to the variable's declaration.
6. Click **OK**.

18.8.8.5 How to Introduce a Constant

While developing your Java application you can easily convert a constant expression into a constant reference. The new constant declaration initialized by the expression will be added to the class, and the original expression will be replaced by the name of the constant.

Expressions can be introduced as constants only if they are literals or operations on literals.

To introduce a constant:

1. In the source editor, select the expression.
2. From the main menu, choose **Refactor** > **Introduce Constant**.

   The Introduce Constant dialog opens.
3. From the **Type** drop-down menu choose a type for the field.

   The menu lists all types that are consistent with the expression. This option will not be shown if only a single type is valid.
4. A suggested name displays in the **Name** text box.

   You can modify or replace it, or choose another suggestion from the drop-down menu.
5. Click **OK**.

18.8.8.6 How to Extract a Method

While developing your Java application you can easily extract part of the body of one method to create another. The extracted code is replaced in the original method with a call to the new method. Local variables and parameters used in the extracted code become parameters of the new method. An assignment made by a statement in the extracted code, if any, will be converted in the original member to an assignment that takes the value of the call to the new method.

To be extractable, a piece of code must satisfy several restrictions:

- It must consist of a single complete expression, or a sequence of complete statements.
- It cannot make an assignment to more than one variable whose declaration is external to the selection.
- It cannot have more than one exit point. An exit point is a statement that throws an exception that is not caught in the selection, a `break` or `continue` statement for a loop outside of the selection, or a **return** statement.

The new method is added to the same class as the original. The new method is declared to be `private`.

> **Note:** Only the selected code block gets replaced by the extracted method. Other occurrences of the same code block do not get replaced.

To extract a method:

1. In the source editor, select the expression or the sequence of expressions that you wish to extract.
2. From the main menu, choose **Refactor** > **Extract Method**.

   The Extract Method dialog opens.
3. Enter a name for the new method.
4. In the **Parameters** list, specify the substitutions that will be made for the local variables and parameters that appear in the selected code:
   - In the **Name** column replacement names, which are similar or identical to the original names, are proposed. You can select and modify the names.
   - In the **Included** column, select the proposed parameters that will become the parameters of the new method. Those that you deselect will become uninitialized local variables in the new method.
   - Use the **Up** and **Down** buttons to order the parameters
5. Select **static** if you want to declare the new method to be static.

   This option is disabled if the method is forced to be static because it is called from a static method, or if it is forced to be non-static because it uses a non-static member.
6. Click **OK**.

   The new method is added to the class, and the code you selected will be replaced by a call to the new method.
7. If you deselected any of the proposed parameters in the **Parameters** list, edit the new method to initialize its local variables.

18.8.8.7 How to Replace a Constructor with a Factory Method

You can convert a constructor into a factory method.

To convert a constructor into a factory method:

1. Select the constructor name in the Structure window or in a Java Source Editor.
2. On the main menu select **Refactor** > **Replace Constructor With Factory Method**.

   The Replace Constructor With Factory Method dialog opens.
3. In the **Method Name** box, enter a name for the new method.

   A suggested name based on the current class name already appears in the box.
4. To convert the constructor into a factory method click **OK**.

18.8.8.8 How to Encapsulate a Field

You can change the fields of a class from being publicly accessible to being accessible only from within the class.

To encapsulate a field:

1. Select the field name (or its parent class) in the Structure window or in the Java Source Editor.
2. On the main menu select **Refactor** > **Encapsulate Fields**.

   The Encapsulate Fields dialog opens.
3. In the **Fields** table, check the box next to each field that you want to be encapsulated.

   In this dialog, you can also specify options for method/field accessibilities and the scope for replacements.
4. Select how you would like accessors to be replaced as part of the encapsulation.
5. If you want to create tasks based on the changes you have made and add them to the Tasks window, check the **Add tasks to the task window** box.
6. If you want to inspect the changes before you commit to the operation, select **Preview**.
7. Click **OK**.

   If you selected **Preview**, finish the extraction operation from the Preview log window. For more information, see Section 18.8.3, "How to Rename a Code Element."

18.8.8.9 How to Invert a Boolean Expression

While developing your Java application, you can select a boolean field, parameter or local variable and initialize it with the opposite value. JDeveloper automatically corrects all references to maintain the same code functionality. JDeveloper looks at all fields, parameters and local variables and inverts all usages. This refactoring changes the sense of a Boolean method or variable to the opposite one. A Boolean expression evaluating to `true` will be `false`. Likewise, a Boolean expression evaluating to `false` will be `true`.

For example, if you have a variable that is enabled and you want to change to change the meaning to disabled, the Invert Boolean menu choice changes usages to disabled.

To invert a boolean method:

1. In the source editor, select the boolean expression.
2. Right-click on the expression and choose **Refactor** > **Invert Boolean**.

   Table 18–1 contains an example of an inverted boolean expression.

Table 18–1 Invert Boolean Example

Before	After

`private double a;`
`...`
`public boolean method() {`
`  if (enabled){`
`  a =5;`
`  return true;`
`  }`
`  false;`
`}`
	`private double a;`
	`...`
	`public boolean method() {`
	`  if (disabled{`
	`  a =5;`
	`  return false;`
	`  }`
	`  return true;`
	`}`

18.9 Optimizing Application Performance

JDeveloper provides a suite of tools for analyzing the quality and performance of your Java code. Use these tools to improve both the quality of your code and your own programming skills.You can use JDeveloper's Auditing feature to analyze Java code for conformance to programming standards.

Auditing is the static analysis of code for adherence to rules and metrics that define programming standards. Auditing finds defects that make code difficult to improve and maintain. JDeveloper's auditing tools help you find and fix such defects. Code can be audited even when it is not compilable or executable.

- A rule is a qualitative test for the presence or absence of some feature. For example, common Java coding style requires that class names be capitalized. A violation occurs when a rule is not adhered to.
- A metric is a quantitative measurement of size or complexity. For example, a method that is too long, or covers too many cases should delegate some of its functionality to other methods. An over-threshold anomaly occurs when the specified upper bound is exceeded.

You can create and customize profiles, choose the rules to be used, and set parameters for individual rules. Browse the audit rules and metrics to learn more about them. For more information, see Section 18.9.25, "How to Browse Audit Rules, Code Assists, and Metrics."

Developers's audit and metrics features are extensible. Audit and metrics are two facets of a source code analysis and transformation framework that can be customized and extended. The public API for both audit and metrics is the `oracle.jdeveloper.audit` package.

To audit Java code:

- Run the auditor on source files to produce an audit report. For more information, see Section 18.9.6, "How to Run Audit to Generate an Audit Report."
- Use Code Assist to audit while editing. Audit violations are highlighted as you edit, and you can apply automated corrections.

- Audit from the command line to produce an audit report. For more information, see
- The Status window displays audit violations in the document selected in the Active view.

An audit report displays rule violations and measurements organized as a tree. A row of the tree corresponds to either a construct or a violation, and includes any measured values for the construct or theoretical violation. A construct is a method, class, file, project, or workspace.

The following properties are found in the rules:

Default fix

The fix that will be used for violations of this rule are when **Apply Default Fix** is applied to a construct.

Pattern

A regular expression used as a filter to find unconventional identifiers.

Severity

Use to sort rule violations in the audit report.

Visibility

A threshold based on the accessibility keyword. Violations will be reported only if they occur in classes or methods having at least the chosen visibility.

18.9.1 Understanding Audit Rules

Audit rules are static, qualitative, analyses of code.

In an auditing profile individual rules can be enabled and configured by setting their properties. When a code construct does not satisfy a rule, a rule violation is reported. Some rules define automatic fixes that you can choose to apply.

Table 18–2 Audit Rules

Rule	Description
Default Fix	The fix that will be used for violations of this rule are when **Apply Default Fix** is applied to a construct.
Pattern	A regular expression used as a filter to find unconventional identifiers.
Severity	Use to sort rule violations in the audit report.
Prefix	A regular expression used to find identifiers with unconventional prefix.
Visibility	A threshold based on the accessibility keyword. Violations will be reported only if they occur in classes or methods having at least the chosen visibility.

18.9.2 Understanding Audit Metrics

Audit metrics are static, quantitative analyses of code.

In an auditing profile individual metrics can be enabled and configured. Metrics are configured by setting a threshold: when a code construct exceeds the threshold, an over-threshold measurement is reported in the audit report.

JDeveloper measures the following metrics:

Depth of Inheritance Tree (DIT)

The depth of the inheritance tree of a class. By convention, java.lang.Object has DIT of 1, a class which directly extends java.lang.Object has DIT 2, and so on.

Number of Statements (NOS)

The size, in Java statements, of a method, class, or other construct.

Cyclomatic complexity (V(G))

The branching complexity of a method. Constructs which enclose methods, such as classes and projects, are assigned the maximum complexity measured for an enclosed method. Values above 10 are generally considered problematic.

18.9.3 Using the Auditing Tools

You can use auditing tools to view audit reports and to investigate and correct rule violations and over-threshold measurements. A new tab will be created in the Log window when auditing starts, and the audit report will be displayed in it.

Auditing is the static analysis of code for adherence to rules and metrics that define programming standards. Auditing finds defects that make code difficult to improve and maintain. The JDeveloper auditing tools help you find and fix such defects. Code can be audited even when it is not compilable or executable.

18.9.3.1 Using the Audit Window Report Panel

An audit report is a set of rule violations and metrics measurements presented as a tree organized into constructs. A construct is a method, class, package, file, project, or workspace. If the audit profile includes rules, the table will have a Severity column that shows the designated severity of the constructs. If the audit profile includes metrics, the table will have an additional column for each metric showing the measurements for the constructs.

To sort the report by the contents of a column, click the column header. To reverse the sort order, click again.

18.9.3.2 Using the Audit Window Toolbar

From the Log window toolbar you can perform the operations shown in Table 18–3.

Table 18–3 Audit Window Toolbar Icons

Icon	Name	Description
	Refresh	Click to rerun the audit on the same selection with the same profile.
	Cancel	Click to terminate a running audit. Note that this may give partial results.
	Export	Click to open the Export Results Dialog, from which you can save the report to a file. You may save the results in XML, HTML, or plain text.
	Expand All	Click to expand all the container nodes in the report, exposing all the rows.
	Collapse All	Click to collapse all the container nodes in the report, hiding all but the top-level constructs.

Table 18–3 (Cont.) Audit Window Toolbar Icons

Icon	Name	Description
	Group Constructs By	Click to open the Group By dialog, from which you can specify the types of container constructs that will be shown. Grouping by constructs enables you to organize the results better, track defects and violations quickly, and analyze the results easily.
	Fix	Choose a fix for a rule violation from the dropdown menu. For an individual rule violation, choose among the fixes defined for that violation's type. For a group construct, the only choice is **Apply Default Fixes**, which applies the default fix defined for its type, if any.
	Show Over Threshold Only	Toggle the display of measurements that are within acceptable limits. The threshold is a settable property of metrics.

18.9.3.3 Using Filters

You specify filters to prune the set of Java classes whose violations are shown. You can filter by package names, class names, or both. A filter consists of one or more patterns separated by commas.

A pattern can contain the following special characters:

- * matches any number of characters
- ? matches any single character
- ! at the beginning of a pattern denotes an exclusion pattern

The set of classes that passes a filter is determined by considering the patterns in order. A non-exclusion pattern adds all classes that match the pattern to the set, an exclusion pattern removes all classes that match the pattern from the set. Table 18–4 contains the filters you can specify

Table 18–4 Filters

Name	Description
Package	Enter filter patterns that will apply to all but the last element of fully qualified class names. If this field is empty it has no effect.
File	Enter filter patterns that will apply only to the last element of fully qualified class names. If this field is empty it has no effect.
Apply	Click to apply the given **Package** and **File** filters to the report's rows.
Clear	Click to erase the **Package** and **File** filters, and to restore the report's rows.

18.9.3.4 Using the Audit Window Context Menu

Select one or more constructs (container nodes) or rule violations (leaf nodes) and right-click to open the context menu. From the context menu you can perform the operations shown in Table 18–5 on the selected constructs or rule violations.

Table 18–5 Audit Window Context Menu Items

Name	Description
Apply Default Fixes	Choose to apply the default fix, if any, to each selected rule violation or to all the violations in the selected constructs. You can define the default fixes using the **Tools** > **Preferences** > **Audit: Profiles** page.
Apply <*Fix*>	Choose to apply this fix to the selected rule violation in the construct.
About <*Rule*>	Choose to display an explanation of the rule that applies to this rule violation.
Hide <*Rule*> Violations	Choose to remove all violations of the selected rule from the report.
Show Hidden Violations	Choose to restore all previously hidden violations.
Show Over Threshold Only	Click to toggle the display of measurements that are within acceptable limits.
Cancel	Choose to terminate a running audit.
Refresh	Choose to rerun the audit.
Group By	Choose to open the Group By dialog, from which you can specify the types of container constructs that will be shown.
Expand All	Click to expand all the container nodes in the report, exposing all the rows.
Collapse All	Click to collapse all the container nodes in the report, hiding all but the top-level constructs.
Go to Source	Choose to open the source file at the point of the rule violation. If you wish, you can edit the file and correct the violation.
Export	Choose to open the Export Results Dialog, from which you can save the report to a file.

18.9.4 How to Audit Java Code in JDeveloper

JDeveloper's auditing tools help you find and fix defects that make code difficult to improve and maintain. You can audit code even when it is not compilable or executable. The focus of an audit is defined by a profile, which is a set of audit rules and metrics.

You can browse the audit rules and metrics to learn more about them. For more information, see Section 18.9.25, "How to Browse Audit Rules, Code Assists, and Metrics."

To audit Java Code:

1. Create an Audit Profile that specifies the rules, code assists, and metrics used to analyze Java programs. In an Audit Profile, individual rules and metrics can be enabled and configured by setting their properties. When a code construct does not satisfy a rule, a rule violation is reported.
2. Run the Audit Report.
   - From the main menu, choose **Build Audit....** For more information, see Section 18.9.6, "How to Run Audit to Generate an Audit Report."
   - You can also audit Java code from the command line by invoking ojaudit.exe, which is included in your JDeveloper installation. For more information, see Section 18.9.5, "Auditing Java Code from the Command Line."

3. Inspect the completed Audit Report for rule violation. For more information, see Section 18.9.15, "How to Inspect an Audit Report Violation or Measurement."

   An Audit Report displays rule violations and measurements organized as a tree. A row of the tree corresponds to either a construct or a violation, and includes any measured values for the construct or theoretical violation. A construct is a method, class, file, project, or workspace.

4. Fix an audit rule violation manually by editing the source, or for some rules, by selecting an automated fix. For more information, see Section 18.9.16, "How to Fix an Audit Rule Violation.".

5. If you want to run the audit again, you can modify an audit profile by enabling or disabling rules, code assists, and metrics, or by changing their configuration. For more information, see Section 18.9.22, "How to Modify an Audit Profile."

You can save the finished audit report as an XML file or as a formatted HTML or text file. For more information, see Section 18.9.14, "How to Save an Audit Report." Formats are defined by XSL stylesheet files in the `/jdev//audit/stylesheets` directory (this directory is not created until audit is run). To create a custom format, adapt a copy of one of the predefined stylesheet files, and add it to the directory.

18.9.5 Auditing Java Code from the Command Line

You can audit a workspace, a project, or a source file from the command line by invoking ojaudit.exe, which is included in your JDeveloper installation, in the `<jdev_install>/jdeveloper/jdev/bin` directory.

Synopsis

```
ojaudit option... file...
```

Table 18–6 contains the parameters you can use during the audit.

Table 18–6 Command Line Parameters

Parameter	Description
file	The workspace (`.jws`), project (`.jpr`), or source (`.java`) file to be audited.
`-classpath` *path*	Set class path for files to audit, if a project is not being audited.
`-encoding` *code*	The character encoding for the report. If absent, the character encoding specified for the project is used (see the Compiler page of the project's Project Properties dialog).
`-disable` *rule*	Disable rule in profile. To supply multiple values, repeat this option.
`-enable` *rule*	Enable rule in profile. To supply multiple values, repeat this option.
`-fail` *severity*	Set the issue severity that the Auditor will regard as failure.
`-f[ix]`	Applies default fixes to the code. This option modifies source files.
`-h[elp]`	Prints help for the command help and exits.
`-o[utput]` *file*	The pathname of the output file. If omitted, output is written to standard output.

Table 18–6 (Cont.) Command Line Parameters

Parameter	Description
`-p[rofile] name`	(required) The profile to use. It is either one of the profiles defined in JDeveloper (as set in the **Audit** > **Audit Profiles** page of the **Tools** > **Preferences** dialog), or the path name of an exported Audit profile file. Case and whitespace are ignored when searching for a matching profile.
`-profilehelp`	Print defined profile names and exit.
`-project file`	The project context to use for parameters that are source files. If all parameters are projects or workspaces, this option is not required.
`-q[uiet]`	Suppresses the copyright message.
`-sourcepath path`	Set source path for files to audit, if a project is not being audited
`-s[tyle] file`	The XSLT stylesheet to apply to the report. The name can either be a style sheet defined in JDeveloper, or a pathname to a style sheet file. If absent, the output will be an XML file. Case and whitespace are ignored when searching for a matching predefined stylesheet.
`-stylehelp`	Print defined style sheet names and exit.
`-title text`	The title to use for the report. If absent when `-untitled` is not specified, a default title will be used.
`-untitled`	Causes the title to be omitted from the report.
`-v[erbose]`	Causes all execution messages to be displayed.
`-version`	Prints the command's version and exits.
`-w[orkspace]`	The workspace context to use for parameters that are not workspaces. If no workspace is designated, a default workspace is synthesized.
`workspace file`	Sets workspace context for files to audit.

18.9.6 How to Run Audit to Generate an Audit Report

Developer allows you to audit your Java programs and generate an audit report.

To audit Java code in JDeveloper:

1. In the Application Navigator, select one or more applications, projects, or Java source files. The Audit command also works for selections from other views, such as editors and the Structure window.
2. From the main menu choose **Build** then **Audit target**.

   The Audit <File, Project, or Application> Dialog dialog appears.
3. Choose a profile to use in one of the two following ways:
   - From the **Profile** dropdown menu choose a profile to use.
   - Click **Edit** to create or modify a profile.
4. Click **Run**.

   An audit report appears in the Log window, and the audit begins. If you wish to stop the audit, click the stop icon in the log's toolbar.

18.9.7 How to Audit Serializable Fields That Do Not Have The serialVersionUID

An object is marked serializable by implementing the `java.io.Serializable` interface, which signifies that the object can be flattened into bytes and subsequently inflated in the future.

There is an identifier called `serialVersionUID` that enables versioning.

You can run an audit that flags all classes that implement java.io.Serializable but do not also have the serialVersionUID.

To set audit rules

- From the main menu, choose **Tools** then **Preferences** then **Audit** then**Profiles**.

18.9.8 How to Audit Unserializable Fields

An object is marked serializable by implementing the java.io.Serializable interface, which signifies that the object can be flattened into bytes and subsequently inflated in the future.

To turn off serialization on a field of an object, tag the field of the class of the object with the Java's transient keyword. If a class is marked as serializable, but contains unserializable fields that are not marked as transient, then the class is not serializable. You can run an audit to detect these unserializable fields.

To set audit rules:

- From the main menu, choose Tools then Preferences then Audit then Profiles.

18.9.9 Viewing an Audit report

Audit reports are displayed as tabbed panes of the Log window. Use the audit report to investigate and correct rule violations and over-threshold measurements.

18.9.10 Refreshing an Audit Report

Use refresh to rerun an audit using the same profile. You may wish to perform a refresh after you have made changes and fixes to your code.

To refresh an audit report:

- Click in the Log Window toolbar, or right-click and choose **Refresh**.

The Export Audit Results dialog is cleared, and a new audit will begin. If you wish to stop the audit, click in the Log's toolbar.

18.9.11 Organizing Audit Report Columns

You can rearrange the audit report columns to follow your preferred organization.

To organize audit report columns:

- Drag the column headers left or right to your preferred position.

18.9.12 How to Organize Audit Report Rows

Audit report rows are rule violations or measurements, or groups of violations and measurements. The report is organized as a tree. A row of the tree corresponds to either a construct or a violation, and includes any measured values for the construct or

a theoretical violation. A construct is a method, class, file, package directory, project, or workspace.

You can choose the constructs that are shown in the report.

To organize audit report rows:

1. Click the Group By icon in the Log window toolbar.

   The Group By dialog opens.

2. Select the constructs you wish to see.
3. Click **OK**.

To sort a metrics report:

- Click a column header to sort rows by that column. To reverse the sort order, click again.

18.9.13 How to Filter Audit Report Rows

You can use a class name pattern filter to hide violations and measurements in classes that do not match the filter.

A filter is a sequence of patterns separated by commas, semicolons, or spaces. A pattern can contain the following special characters:

- * matches any number of characters
- ? matches any single character
- ! at the beginning of a pattern denotes an exclusion pattern

The set of visible classes is determined by considering the patterns in order. A non-exclusion pattern adds all classes that match the pattern to the set; an exclusion pattern removes all classes that match the pattern from the set. Patterns are matched against classnames.

To filter audit report rows:

1. In the **Package** box of the audit log window, enter a sequence of patterns that will apply to all but the last element of fully-qualified class names. You can leave this box empty if you specify a File filter.
2. In the **File** box, enter a sequence of patterns that will apply only to the last element of fully-qualified class names. You can leave this box empty if you specify a Package filter.
3. Click **Apply**.

   The report will be redisplayed to show only the selected rows.

Click Clear to delete text from the Package and File boxes.

18.9.14 How to Save an Audit Report

You can save an audit report as an XML file or as a formatted HTML or text file.

Formats are defined by XSL stylesheet files in the `<jdev_install>/jdev/<system>/audit/stylesheets` directory (this directory is not created until audit is run). To create a custom format, adapt a copy of one of the predefined stylesheet files, and add it to the directory.

To save an audit report:

- Click in the Log Window toolbar, or right-click and choose Export.

  The Export Audit Results dialog opens. Choose a title, format, and destination for the report, and click **OK**.

18.9.15 How to Inspect an Audit Report Violation or Measurement

JDeveloper generates a report of all audit rule violations.

To inspect an audit rule violation:

1. In the audit report, select the construct you wish to view.
2. Right-click and choose **Go to Source**, or double-click the construct.

   An editor for the source file opens with the cursor positioned at the location of the rule violation or the code element measured
3. Right-click on a violation or anomaly, and select **About** to learn more about the rule that has been violated

18.9.16 How to Fix an Audit Rule Violation

You can fix an audit rule violation manually by editing the source, or for some rules, by selecting an automated fix.

To manually fix an audit rule violation:

1. In the audit report, select the rule violation (a leaf node in the Constructs tree).
2. Right-click and choose **Go to Source**.

   An editor for the source file opens with the cursor positioned at the location of the rule violation.
3. Edit the code to correct the cause of the violation.

To apply an automated fix to an audit rule violation:

1. In the audit report, select the rule violation (a leaf node in the **Constructs** tree).
2. Right-click, and choose an **Apply <Rule> Fix** menu item, if any.

   or

   Click in the Log window toolbar, and choose one of the **Apply <Rule> Fix** menu items.

18.9.17 How to Fix a Construct's Audit Rule Violations

You can apply automated fixes to all the rule violations in a construct. Default fixes will be applied to each rule violation in the construct that has a `Default Fix` property with a value other than `None`.

To fix a construct's audit rule violations:

1. In the audit report, select the construct (a container node in the **Constructs** tree.
2. You can apply default fixes in one of the two following ways:
   - Right-click, and choose **Apply Default Fixes**.
   - Click in the Log window toolbar, and choose **Apply Default Values**.

18.9.18 How to Hide Audit Rule Violations

You can suppress the display of all the rule violations of a given type in the audit report. It is not possible to suppress individual rule violations.

To hide audit rule violations:

1. In the audit report, select a rule violation (a leaf node in the **Constructs** tree).
2. Right-click, and choose **Hide <Rule> Violations**.

   All of the violations of <*Rule*> are removed from the audit report. The removed rules are not tallied in their parent construct's summaries. Empty constructs are removed if **Show Over Threshold Only** is enabled. If not, just the violations are removed.

To restore hidden audit rule violations:

1. In the audit report, right-click to open the context menu.
2. Choose **Show Hidden Violations**.

   All of the previously hidden rule violations are restored to the audit report.

18.9.19 How to Hide Audit Report Measurements

Metrics reports display measurements for the constructs in the analyzed code. You can focus the report on over-threshold measurements by hiding the others. The threshold is a settable property of metrics.

To show only over-threshold measurements:

In the Log window toolbar, click the over threshold icon. Click again to show all measurements.

Removed measurements are not tallied in their parent construct's summaries. Empty constructs are removed if Show Over Threshold Only is enabled. If not, just the violations are removed.

18.9.20 Managing Audit Profiles

An audit profile defines the focus of an audit by specifying the rules, code assists, and metrics that will be used to analyze Java code. While several profiles are predefined, you can create others. You can modify an audit profile by enabling or disabling rules, code assists, and metrics, or by changing their configuration.

Certain audit profiles are used by default with some JDeveloper processes and features.

- The Code Assist profile is used by the Source Editor, Status window, Application Overview, and File List.
- The Audit While Compiling profile is used at the end of a compile when Audit While Compiling is selected in the Audit page of the Preferences dialog.
- The Audit Rules profile is the initial default for the Audit command. However, this is not permanent because the Audit dialog remembers whatever profile was last selected.

18.9.21 How to Create an Audit Profile

JDeveloper allows you to specify the rules, code assists, and metrics used to analyze Java programs.

To create an audit profile:

1. From the main menu, choose **Tools** then **Preferences**.

   The Preferences dialog opens.

2. Choose the Audit - Profiles page.
3. From the Profile dropdown menu, choose a profile to copy.
4. Select the rules, assists, and metrics to enable in the new profile. For more information, see Section 18.9.26, "How to Activate and Deactivate Components of an Audit Profile."
5. Configure the selected rules, assists, and metrics, if desired. For more information, see Section 18.9.27, "How to Set Property Values for an Audit Test."
6. Click **Save As**.
7. Enter a name for the new profile, and click **Save.**

   Note: Names are not case or space sensitive, though case and space are preserved. If the new name differs only in case or space from an existing name, a warning message appears to inform you of this.

   The new profile name is shown in the in the Audit Profiles preferences page's Profile box.

8. Click **OK**.

18.9.22 How to Modify an Audit Profile

JDeveloper allows you to modify audit profiles you created to analyze Java programs.

To modify an existing audit profile:

1. From the main menu, choose **Tools** > **Preferences**.

   The Preferences dialog opens.

2. Choose the Audit - Profiles page.
3. From the **Profile** dropdown menu, choose a profile to modify.
4. Select the rules, assists, and metrics to enable in the new profile. For more information, see Section 18.9.26, "How to Activate and Deactivate Components of an Audit Profile."
5. Configure the selected rules, assists, and metrics, if desired. For more information, see Section 18.9.27, "How to Set Property Values for an Audit Test."
6. Click **OK**.

18.9.23 How to Delete an Audit Profile

To delete an existing audit profile:

1. From the main menu, choose **Tools** >**Preferences**.

The Preferences dialog opens.

2. Choose the Audit - Profiles page.
3. From the **Profile** dropdown menu, choose a profile to be deleted. (You cannot delete the predefined profiles.)
4. Click **Delete**.

   The profile's name will be removed from the **Profile** box.
5. Click **OK**.

18.9.24 How to Import or Export an Audit Profile

You can import or export audit profiles. This would enable you to share profiles, for example, or to maintain a checked in profile used by ojaudit and a nightly build. Audit profiles are imported or exported as XML files.

To import or export an audit profile:

1. In the **Tool**s menu, select **Preferences** to open the Preferences dialog.
2. In the Preferences dialog, open the Audit - Profiles page
3. Click **Import** or **Export**, and select the profile you want to import or export.

18.9.25 How to Browse Audit Rules, Code Assists, and Metrics

Browse the audit tests to learn more about them.

To browse the audit rules:

1. From the main menu, choose **Tools** >**Preferences**.

   The Preferences dialog opens.
2. Choose the Audit - Profiles page.
3. Select one of the **Rules**, **Code Assists**, and **Metrics** tabs.
4. In the list tree in the left panel, select a category.

   A description of the category will be shown in the Explanation box.
5. Expand the category and select a rule, code assist, or metric, depending on the tab selected.

   A description of the selected item will be shown in the Explanation box, and its properties and settings will be shown in the right pane
6. Click **OK** to close the dialog.

18.9.26 How to Activate and Deactivate Components of an Audit Profile

Activate and deactivate rules, code assists, and metrics for an audit profile from the Audit Profiles preferences page while creating or modifying the profile.

To activate or deactivate a rule, code assist, or metric:

1. Select the project in the Application Navigator
2. Choose **Tools** then **Preferences - Audit - Profiles** page.
3. In the dialog, select one of the **Rules**, **Code Assists,** and **Metrics** tabs.

4. In the list tree in the left panel, expand the category.
5. Click the checkbox for a test to activate or deactivate it.
6. Optionally, configure the rule, code assist, or metric. For more information, see Section 18.9.27, "How to Set Property Values for an Audit Test."
7. Click OK when you are done.

To activate or deactivate a category:

In the left-hand list, click the checkbox for a rule category to activate or deactivate all the category's items. Clicking on any item in the category while pressing the Ctrl key achieves the same result.

18.9.27 How to Set Property Values for an Audit Test

To set an audit rule's property values:

1. Select the project in the Application Navigator.
2. Choose **Tools** then **Preferences - Audit - Profiles** page.
3. In the dialog, select one of the **Rules**, **Code Assists**, and **Metrics** tabs.
4. In the list tree in the left pane, expand a category and select an item.
5. The test's property names and current values are shown in the right panel.
6. Change the property values by choosing or entering alternative values.
7. Click **OK** when you are done.

18.10 Profiling a Project

The profiler gathers statistics on your program that enable you to more easily diagnose performance issues, such as bottlenecks by identifying methods consuming more time, which method is called the most, how memory is used, and what kind of objects are being created.

The Profiler monitors and logs a running program's use of processor and memory resources. It gathers statistics that enables you to more easily diagnose the performance issues and correct the inefficiencies in your code.

JDeveloper offers two kinds of profilers: The CPU Profiler and the Memory Profiler, for local as well as remote profiling.

- The CPU Profiler is used to analyze your application's impact on the processor. Use the CPU Profiler to test functions of your application, such as startup and initialization, repainting, and compiling.
- The Memory Profiler provides a visual and statistical analysis of how your program utilizes memory in the Java heap.

18.10.1 Understanding Memory Profiler Views

Memory profiling displays which parts of the application are using the most memory. It also enables you to investigate which objects are responsible for holding the most memory. The Memory Profiler displays data in three views: Classes and Allocators for new objects/garbage collection reporting, and the References view for heap snapshots.

The Classes view shows new objects/garbage collection data organized by Java class. It is used to discover which classes allocated the most memory.

The Allocators view shows the threadgroups, threads and methods that created the most memory.

The References view contains a hierarchical display of all classes, objects and references to objects in the application heap at the time of the snapshot.

Profilers are invoked from the **Run** menu. You can use the Profiler pages of the Edit Run Configuration dialog to specify the mode of profiling you would like to analyze your code.

In the dialog, you can also specify how you want to profile your programs, locally or remotely. To profile your program locally, set **Profiler Connection Parameters**, and to profile your program remotely, set **Remote Profiling Parameters** in the Profiler page of the Edit Run Configuration dialog.

18.10.2 Profiling an Application

You can profile your application locally or remotely. In local profiling, you profile your application within JDeveloper by choosing one of the CPU Profile or Memory Profile commands from **Run** menu, or outside JDeveloper by choosing **Attach Profiler** from the Run menu. In remote profiling, you run a profiling session remotely. You profile your application external to JDeveloper with, or without, profiler parameters. The application's JVM can be on the same host or on a network.

18.10.3 Configuring Profilers

You can configure options for the CPU and Memory profilers by right-clicking on a project and clicking **Project Properties**. Click **Run/Debug/Profile** > **Edit** > **Tool Settings** > **Profiler** to set the options.

18.10.4 Understanding CPU Profiling

You use the CPU Profiler to gather statistics on the performance of your application. The CPU Profiler can be used to test functions of your application, such as startup, initialization, repainting, and compiling.

The CPU Profiler collects sample data on a running application at regular intervals based on the Sample Interval setting, which you can change using the CPU Profiler options page. For more information, see Section 18.10.13, "How to Set Options for the CPU Profiler."

The Profiler provides two CPU profiling options: Sample CPU Time and Count Method Calls. You can use Sample CPU Time to determine the areas of your code which are accounting for the most (or least) execution time. Count Method Calls display which methods are called and the number of times they are called.

> **Note:** The Profiler does not allow you to sample time and count method calls at the same time. Counting method calls will cost you a high level of CPU time, and also distorts the profiler results.

Counting method calls is the most expensive form of profiling, both in terms of CPU overhead and memory usage, because the process uses extra CPU for every method entry and exit and it collects data for every method that is called during the use case.

The CPU Profiler window displays data that the Profiler returns. It provides several options to view, sort, and organize Profiler results, as well as to begin, label, compare, and end Profiler use cases.

18.10.5 Understanding Memory Profiling

The profiler has two tools for memory profiling: New objects/garbage collection reporting and reference snapshots (also known as heap dumps). Use new objects/garbage collection reporting to find out what parts of an application are using the most memory. Use reference snapshots to find out what objects are responsible for holding the most memory. The options allow you to use both at the same time, but usually you would not need a snapshot unless you have spotted a memory leak, and you would not need another new objects/G.C. report if you are investigating a particular leak.

The Memory Profiler window displays data that the profiler returns. It provides several options to view, filter, sort, and organize Profiler results, as well as to begin, label, compare, and end Profiler use cases

18.10.6 Understanding Profiler Performance

The Profiler itself consumes CPU and memory resources during use. The overhead varies depending on the profiling mode in operation. In general, CPU sampling (with a 20 ms. sample interval) and reference snapshots are the fastest and use the least memory.

Both CPU method call count and memory new objects/garbage collection profiling are much more expensive in terms of CPU and memory in both the Profiler and profilee. This is because they intercept essentially every method call and record per method, per call stack, per thread data.

In the case of reference snapshots, it takes only a few seconds for the Profiler agent to write a snapshot file. All other processing is done outside the application, so memory snapshots have little effect on application performance.

Table 18–7 contains some actual performance times for an application's benchmark.

Table 18–7 Application Benchmark Times

Profile Type	Relative Time	Time (seconds)	Notes
No profiling	1.0	720	
CPU samples	1.025	738	20ms sample rate
CPU samples	1.29	930	10ms sample rate
Memory new objects/garbage collection	1.56	1125	
Reference Snapshot	1.02	735	

Table 18–8 contains the numbers produced by a shorter run:

Table 18–8 Shorter Run Times

Profile Type	Relative Time	Time (seconds)	Notes
No profiling	1.0	30.7	
CPU samples	1.4	41.8	20ms sample rate
CPU samples	1.7	51.2	10ms sample rate
Method call count	4.6	141.2	With method filter
Method call count	6.6	201.3	No filter

As the Profiler output is essentially the same for both the 10 and 20 ms. sample intervals, the 20 ms. interval is preferable.

The two tables illustrate that while profiling overhead can vary dramatically for different use cases, CPU samples and reference snapshots are always less expensive than the other alternatives.

In JDeveloper, you can profile a usecase in the default allocated 512MB memory, but some use case require a memory of 1GB or more for new objects/G.C. memory profiling.

18.10.7 Understanding Profiler Use Cases

The aim of profiling is to discover those parts of an application that use more resources than desired, and improve them so that optimization of resources can be achieved. The process is to find individual commands, transactions, or sequences of actions that are perceived to be slow, use too much memory or leak memory. You then investigate the cause, make changes intended to improve performance, and then measure again to see if the changes had the desired effect.

In the profiler, we call such a command, transaction, or sequence of actions a use case. The profiler is designed to allow you to isolate meaningful use cases and measure them precisely. By default, the profiler measures nothing until you begin a use case and stops measuring when you end the use case. The data displayed is always relative to a given use case.

The toolbar in the profiler tab indicates the current use case status and provides controls to start and stop use cases and navigate between them.

The **Begin Use Case** icon in the top left of the toolbar is used to start a new use case. The **End Use Case** icon beside it is used to stop a running use case. When you start a use case, the **End Use Case** toolbar icon is enabled and the **Begin Use Case** toolbar icon is disabled.

If you save an active use case, the use case stops immediately; the **End Use Case** toolbar icon is disabled, and **Begin Use Case** toolbar icon is enabled.

Both the begin and end use case controls are disabled when:

- A use case has ended but is still being processed by the profiler.
- The profilee application has stopped running.
- The profiler is no longer connected.
- You open a saved session.

18.10.8 How to Profile a Project in JDeveloper

The Profiler monitors and logs a running program's use of processor and memory resources. It gathers statistics that enable you to diagnose performance issues and correct the inefficiencies in your code.

The steps for profiling a project are:

1. Set options for the CPU and Memory Profilers. For more information, see Section 18.10.13, "How to Set Options for the CPU Profiler" and Section 18.10.15.3, "How to Set Options for the Memory Profiler."

   For example, you can specify if you want the Profiler to sample CPU time usage by your application, or to count method calls. Or you can specify if you want the

Profiler to report data on new objects and garbage collection, or to provide a heap snapshot.

2. Start the CPU or Memory Profiler inJDeveloper. For more information, see Section 18.10.14, "How to Start the CPU Profiler."

   You can also run a profiling session remotely. When you start a remote profiling session, the Profiler connects and profiles remote applications as if they were local. For more information, see Section 18.10.16, "Profiling Remotely."

3. Isolate meaningful use cases in your program and measure them precisely. For more information, see Section 18.10.7, "Understanding Profiler Use Cases."

   The **Begin Use Case** control in the top left of the toolbar is used to start a new use case. The **End Use Case** control beside it () is used to stop a running use case. When you start a use case, the E**nd Use Case** toolbar icon is enabled and the **Begin Use Case** toolbar icon is disabled.

4. Inspect the data that the profiler returns. The Profiler windows provide options to view, filter, sort, and organize Profiler results, as well as to begin, label, compare, and end Profiler use cases.

5. You can also save the results of your Profiling session for later analysis.

After you have completed your Profiling session, you can then use its statistics to more easily diagnose the performance issues and correct the inefficiencies in your code.

18.10.9 CPU Profiling

The CPU Profiler tabulates and displays statistical data on the performance on your application, either in terms of time usage, or methods called.

The CPU Profiler enables you to profile your code in one of two ways:

- Sample CPU Time to identify which parts of an application are taking the most time.
- Count Methods Calls to confirm that methods are being called and how many times they are called.

18.10.10 Understanding CPU Profiler Views

CPU profiling tabulates the processing time spent by each method in your application by displaying Java Platform calls, and counting those method calls. The CPU Profiler displays data in two views: Hotspots and Call Stacks.

The Hotspots view lists all Java platform methods and all methods they call, sorted by time usage in the CPU Profiler's time sampling mode of operation. It also displays the cumulative amount of CPU time spent in each method. During the method call count operation, the Hotspots view shows all the methods and the number of times they were called.

The Call Stacks view, during the CPU Profiler's time sampling mode of operation, lets you view the Java platform methods called in their call hierarchy. In the count method call mode, the Call Stacks view lists the Java platform methods called, sorted by thread group.

18.10.11 Understanding CPU Time Sampling Results

The CPU profiler, when used in time sampling mode, analyzes your program and reports results on CPU time usage. The data is displayed in two views: Hotspots and Call Stacks.

The Hotspots view lists all methods sorted by the value of the time spent in them. The hierarchy beneath each method shows the callers of the method, and in turn, their callers, and so on.

The topmost methods in Hotspots are often just noise; the path that the event dispatcher takes to accomplish a task. You can use the Stack Filter to narrow the view down to only relevant methods. Alternatively, you can use the Edit Filter dialog to enter a find or filter expression that will specify the methods to be displayed.

The Call Stacks view lets you see the methods called in their call hierarchy. That is, each level is called by the level immediately above it. Table 18–9 contains the data columns that display in the Calls Stacks View.

Table 18–9 Profiler Data Columns

Column	Description
Name	Displays the fully qualified method names.
CPU%	Displays the percentage of the currently selected data column. For example, if the CPU column is selected, indicated by a downward-pointing triangle, this column is named CPU%. If CPU Shallow column is selected, the column name would be CPU Shallow%.
CPU (ms)	Displays the cumulative amount of CPU time spent in seconds, in each method and all the methods it calls.
CPU Shallow (ms)	Displays the amount of CPU time spent in each method individually.
Blocked (ms)	Displays the cumulative blocked time spent in each method and all the method it calls. The column is available if the **Collect Blocked and Wait Time** checkbox is selected in the Edit Run Configuration dialog.
Blocked Shallow (ms)	Displays the blocked time spent in each method. The column is available if the **Collect Blocked and Wait Time** checkbox is selected in the Edit Run Configuration dialog.
Wait (ms)	Displays the cumulative wait time of each method and all the method it calls. The column is available if the **Collect Blocked and Wait Time** checkbox is selected in the Edit Run Configuration dialog.
Wait Shallow (ms)	Displays the wait time of each method. The column is available if the **Collect Blocked and Wait Time** checkbox is selected in the Edit Run Configuration dialog.
Elapsed (ms)	Displays the cumulative elapsed time of each method and all the method it calls. The column is available if the **Collect Elapsed Time** checkbox is selected in the Edit Run Configuration dialog.
Elapsed Shallow (ms)	Displays the elapsed time of each method. The column is available if the **Collect Elapsed Time** checkbox is selected in the Edit Run Configuration dialog.
I/O (ms)	Displays the cumulative Input/Output time of each method and all the method it calls. The column is available if the **Collect IO Time** checkbox is selected in the Edit Run Configuration dialog.

Table 18–9 (Cont.) Profiler Data Columns

Column	Description
I/O Shallow (ms)	Displays the Input/Output time of each method. The column is available if the **Collect IO Time** checkbox is selected in the Edit Run Configuration dialog.

Note: If you see more than one top-level method per thread in the Stacks view, the stack depth is too low for your application. To increase the stack depth, you need to update the CPU Profiler options page in the Edit Run Configuration dialog.

18.10.12 Understanding Method Call Counts Results

The CPU Profiler, when used to count method calls, confirms that methods are being called and tells you how many times. Method call counting is useful when you want to know why a particular method is taking too much CPU time. You can use the Profiler's count method call data to analyze the callers to that method, and edit your code appropriately to reduce the number

18.10.13 How to Set Options for the CPU Profiler

You can specify if you want the Profiler to sample CPU time usage by your application, or to count method calls.

To set CPU Profiler options:

1. In the navigator, double-click the project you want to profile to open the Project Properties dialog.
2. Click **Run/Debug/Profiler**.
3. Click **Edit**.
4. In the Edit Run Configuration dialog, set the options as desired on the **Tool Settings - Profiler - CPU** page.

   You can specify if you want the profiler to sample CPU time or count method calls.
5. When finished, click **OK**.

18.10.14 How to Start the CPU Profiler

Starting a CPU profiling session will automatically run your program. Once the CPU profiler window is open, you can begin a use case to profile your application.

To start the CPU Profiler:

1. In the navigator, select a runnable node, for example, **Application1.java**.
2. From the main menu, choose **Run** > **CPU Profile** ***project***.

   If no default run target is specified in the Launch Settings page of the Edit Run Configuration dialog (**Application** menu > **Project Properties** > **Run/Debug/Profile**), the Choose Default Run Target dialog opens. Use this dialog to specify the default run target
3. Click the Begin Use Case icon to begin a profiling use case.

Note: If you want to profile your application immediately when the profiler is launched, select the **Begin Use Case on Application Startup** checkbox in the **Profiler** page of Edit Run Configuration dialog.

18.10.15 Memory Profiling

The Memory Profiler enables you to find out how your program is using the Java heap. You can find inefficient heap usage and any suspect memory behavior.

The profiler has two tools for memory profiling: new objects/garbage collection (hereafter referred to as new/G.C. reporting) and reference snapshots (also known as heap dumps). Note the following points while using these tools:

- Use new/G.C. reporting to find out what parts of an application are using the most memory.
- Use reference snapshots to find out what objects are responsible for holding the most memory.

Memory profiling and method call counts, slows down the system and the application considerably. Note the following points of information before you start profiling an application:

- Keep your use cases specific and short.
- Wait for the application to start, before you start the use case.
- When profiling large applications, JDeveloper requires more than the default memory size. If your applications slows down, navigate to *<jdev_install>/jdev/bin* directory and try launching JDeveloper from command line with the following command:

  ```
  jdev -J-Xmx1024m
  ```

  This command allocates 1 GB of virtual memory to JDeveloper, overriding the default memory of 512 MB.
- A web application deployed in Oracle WebLogic Server is considered as a large application. Hence, launch JDeveloper with the `-J-Xmx1024m` parameter to increase the memory size before you start profiling.
- Some large applications or use cases may fail when profiled in Windows, try profiling such applications or use cases in Linux. Windows operating systems limit the contiguous virtual memory for JVM, but there is no such limitation in Linux.

18.10.15.1 Understanding Memory Profiler Views

The Memory Profiler displays data in three views: Classes view, Allocators view for new/G.C. reporting, and the References view for heap snapshots.

- The Classes view shows new objects/garbage collection data organized by Java class. It is used to discover which classes allocated the most memory. You can switch to this view by clicking the **Classes** tab at the bottom of the Memory Profiler.
- The Allocators view shows the threadgroups, threads, stacks, and methods that created the most memory. You can switch to this view by clicking the **Classes** tab at the bottom of the Memory Profiler.
- The References view contains a hierarchical display of all classes, objects and references to objects in the application heap at the time of the snapshot.

18.10.15.2 Understanding Reference Snapshots

A reference snapshot is a hierarchical display of classes, objects and references to objects in the application heap at the time of the snapshot. There are two fundamental kinds of references: references from JVM garbage collection roots to objects, and references from one object to another.

JVM garbage collection roots include references from an active thread, references from JNI local and global variables, and internal JVM references. These are the references that keep objects in memory; if there is no path from a gc root to an object, it is eligible for garbage collection. The snapshot contains only objects that are reachable from one or more JVM gc roots.

Note that static references and references from class loaders are not garbage collection roots. A class loader can be garbage-collected when there are no more references to the class loader, any classes loaded by the class loader or any objects of these classes. When a class loader is garbage collected, all classes it has loaded are also garbage collected, as well as all objects that are only referred to by static references from these classes.

There are four types of references from objects to objects: normal (strong) references, soft references, weak references and pseudo references. When you are looking for memory leaks, you normally only care about strong references. You can toggle between viewing only strong references, or all references.

New and old references in different snapshots are indicated using different font styles. When you compare two memory snapshots, the methods in the current snapshot show up in boldface font. If there are methods that are also in the other snapshot, they are shown in bold-italic font in the current snapshot.

There are several ways to compare reference snapshots. In the Memory Profiler References view, for example, you can choose **Compare With** to select a different use case (snapshot) from the current one. Or, if you have two profiler tabs open (for example, if you have opened a saved profile session) select **Compare With** and **Other**, then choose a use case from another session.

18.10.15.3 How to Set Options for the Memory Profiler

You can specify if you want the Profiler to sample CPU time usage by your application, or to count method calls.

To set memory profiler options:

1. In the navigator, double-click the project you want to profile to report data on new objects and garbage collection, or to provide a heap snapshot.
2. Click **Run/Debug/Profiler**.
3. Click **Edit**.
4. In the Edit Run Configuration dialog, set the options as desired on the **Tool Settings - Profiler - CPU** page.

   You can specify if you want the profiler to collect new objects/garbage collection data, take a heap snapshot, or do both.
5. When finished, click **OK**.

18.10.15.4 How to Start a Memory Profiling Session

Starting a Memory profiling session also automatically runs your program. Once the Memory profiler window is open, you can begin a use case to profile your application.

To start the Memory Profiler:

1. In the navigator, select a runnable node.
2. From the main menu, choose **Run** > **Memory Profile *`project`***.
3. Click the **Begin Use Case** icon to start the profiling session.

18.10.16 Profiling Remotely

You can run a profiling session remotely. When you start a remote profiling session, the Profiler connects and profiles remote applications as if they were local. Since you still run the Profiler locally, you can profile applications on other computers provided that they have a reachable IP address or DNS name.

The main difference between remote and local profiling is the way in which you begin the profiling session. For local profiling, JDeveloper automatically launches the program that you want to profile (the profilee) and then attaches the Profiler to that program. For remote profiling, you must manually launch that profilee program and attach the Profiler later. Once the profilee is launched and the JDeveloper Profiler is attached to it, remote profiling is no different from local profiling. Remember that you can use remote profiling whether or not the profilee process is running on the same machine as JDeveloper.

Note: If you are profiling your application remotely, you can start your external application with or without profiler parameters. An application started with profiler parameters can have JVM can be on the same host (local) or on a network. An application started without profiler parameters must have JVM on the same host, and you may attach the profiler later.

Remote profiling and local profiling each have advantages over the other. When you remote profile, the Profiler and profilee can be run on two different computers so that they are not competing for the same resources. However, transferring large amounts of data over a network could make profiler performance significantly slower.

18.10.17 Understanding Profiler Agent Support for JVMs

To profile remotely in JDeveloper, you invoke the profiler agent. The JDeveloper profiler agent supports 32- and 64-bit JVMs in Windows and Linux. You specify a JVM for a project using the Edit Run Configuration - Launch Settings page.

The names of the profiler agent files are based on the supported architecture and JVM size:

- profiler_x32.dll
- profiler_x32.so
- profiler_x64.dll
- profiler_x32.so

If you launch the profilee from JDeveloper, the profiler automatically detects the size of the JDeveloper JVM.

- On Linux, it should use API to detect which of 32-bit or 64-bit architecture remote JVM is and use appropriate agent. (This can be generalized to support additional architectures.)
- On Windows, it should use API to detect which of 32-bit or 64-bit architecture remote JVM is and, if it is not the same as JDev JVM architecture, not show it in the list of available JVMs to attach

If you attach locally and the JDeveloper JVM is 32-bit, the profiler attempts to load the 32-bit agent and if that fails, tries to use the 64-bit agent. If the JDeveloper JVM is 64-bit, the profiler only tries the 64-bit agent.

18.10.18 How to Invoke the Profiler Agent

To profile a program remotely in JDeveloper, you must start the Java process from the command line. From the command line, you also invoke the Profiler Agent. Once the process has started, you can connect the JDeveloper Profiler to the Profiler Agent.

You can invoke the Profiler Agent using the `-agentlib` or `-agentpath` option.

To invoke the Profiler Agent using the -agentlib option:

At the command line, enter the following execution string:

```
java
-agentlib:<Profiler-Agent-Library>=<sub-option1>[=<value1>],<sub-option2>
[=<value2>]... -classpath
 -classpath <Project_Directory>\classes <Java_Main_Class>
```

> **Note:** While specifying the -agentlib, you may specify the absolute path of the agent excluding the .dll extension, or add the agent's path to your PATH variable and then specify the agent without absolute path and extension.

Example 1

```
java
-agentlib:C:\JDeveloper\jdeveloper\jdev\lib\
profiler_x32=jarpath=C:\JDeveloper\jdeveloper\jdev\lib\
profiler-agent.jar,port=4000,enable=t,startup=connect
-classpath c:\MyApp\MyProject\classes MyMainClass
```

Example 2

```
set PATH=C:\JDeveloper\jdeveloper\jdev\lib;%PATH%

java
-agentlib:profiler_
x64=jarpath=C:\JDeveloper\jdeveloper\jdev\lib\profiler-agent.jar,port=4000,
enable=t,startup=connect
 -classpath c:\MyApp\MyProject\classes MyMainClass
```

To invoke the Profiler Agent using the -agentpath option:

At the command line, enter the following execution string:

```
java
 -agentpath:<Path_to_Agent_Library>=<option1>[=<value1>],<option2>[=<value2>]...
 -classpath <Project_Directory>\classes <Java_Main_Class>
```

where

<Path_to_Agent_Library> is the full path to the profiler_x32.dll or profiler_x64.dll. For example, <jdev_install>\jdeveloper\jdev\lib\profiler_x32.dll.

> **Note:** While specifying the -agentpath, you must specify the absolute path to the agent including the .dll extension.

Example

```
-agentpath:C:\JDeveloper\jdeveloper\jdev\lib\
profiler_x32.dll=jarpath=C:\JDeveloper\jdeveloper\jdev\lib\
profiler-agent.jar,port=4000,enable=t,startup=connect,depth=1000,interval=20
-classpath c:\MyApp\MyProject\classes MyMainClass
```

Table 18–10 contains the suboptions that are available with the -agentlib and -agentpath options.

Table 18–10 Suboptions

Suboption	Description
port=<*port*>	Specifies the port over which the data will be transferred. Defaults to 4000.
jarpath=<*path*>	Path to profiler JAR file. This JAR is located at <jdev_install>\jdeveloper\jdev\lib\profiler-agent.jar. If the JAR path is not specified, the JAR must be on bootstrap classpath.
enable=[t][c][m][r]	Enables agent capabilities. ■ t = CPU time sampling ■ m = memory new/gc ■ c = count method calls ■ r = heap reference snapshot Only one of t, c or m may be specified. r may be specified in combination with m or by itself.
startup=time|mem|count|refs|connect	If a use case is to begin on startup, you may specify one of the following: ■ time - CPU time sampling ■ count - count method calls ■ mem - memory alloc/free ■ refs - heap reference snapshot (may be specified in combination with mem or by itself) ■ connect - waits for connection before allowing application to run It is not possible to have a use case running unless the profiler is connected. startup=mem,refs also enables profiler agent's memory and heap reference snapshot capabilities (enable = mr).
depth=<size>	Sets maximum stack depth used for collection. Defaults to 1000.
interval=<sample-interval>	Sample interval in milliseconds. Defaults to 20. This is only applicable to CPU time sampling.

Table 18–10 (Cont.) Suboptions

Suboption	Description
`wait=y|n`	Report wait and blocked times. Default is n. This is only applicable to CPU time sampling.
`refpath=<path>`	Sets the path used to write reference snapshot files. Required if `enable=r` is specified. The same path is used for all snapshots, so it is important that the file be copied before the next use case ends. The profiler and file client do this automatically.
`stackfilter=<expression>`	Specifies classes to include or exclude in stack traces. For example: `stackfilter=(!java.lang.*)` This expression will ensure that no reported stack trace contains any class whose name begins with `java.lang.*`. For example, stack traces with `java.lang.Integer` and `java.lang.reflect.Method` not appear. Any time used in such methods is collected and added with the next caller.
`methodfilter=<expression>`	Collects only stacks containing the specified methods specified in the expression. For example: `methodfilter=(com.mycorp.MyClass.MyMethod*)` This expression will ensure that no stack will appear unless it contains at least one call to `com.mycorp.MyClass.myMethod`. The * in the filter ensures that methods for any type signature are collected; in Java, there could be more than one method that matches.
`memfilter=<expression>`	Object classes to include or exclude from memory reports. For example: `memfilter=(!java.lang)` This expression will ensure that no class that contains java.lang in its string will be displayed.

Command line examples

- ```
  java
   -agentlib:profiler_x32=port=4000,
  jarpath=C:\JDeveloper\jdeveloper\jdev\lib\profiler-agent.jar,
  enable=t,depth=1000,startup=time,interval=20 -classpath
  C:\JDeveloper\jdeveloper\mywork\Application1\Project1\classes
  project1.Application1
  ```
- ```
  java
  -agentlib:profiler_x64=port=4000,
  jarpath=C:\JDeveloper\jdeveloper\jdev\lib\profiler-agent.jar,
  enable=m,startup=connect -classpath
  C:\JDeveloper\jdeveloper\mywork\Application1\Project1\classes
  project1.Application1
  ```
- ```
  java
  -agentlib:profiler_x32=port=4000,
  jarpath=C:\JDeveloper\jdeveloper\jdev\lib\profiler-agent.jar,
  startup=connect,mem -classpath
  C:\JDeveloper\jdeveloper\mywork\Application1\Project1\classes
  project1.Application1
  ```
  ```

- ```
  java
  -agentpath:C:\JDeveloper\jdeveloper\jdev\lib\profiler_
  x64.dll=
  jarpath=C:\JDeveloper\jdeveloper\jdev\lib\profiler-agent.jar,
  port=4000,enable=t,startup=connect,depth=1000,interval=20
  -classpath
  C:\JDeveloper\jdeveloper\mywork\Application1\Project1\classes
  project1.Application1 *
  ```

18.10.19 How to Connect the Profiler Remotely to a Java Program

To profile a program remotely in JDeveloper, you must first start the Java program session and invoke the Profiler Agent. For more information, see Section 18.10.18, "How to Invoke the Profiler Agent."

Once the session has started, you can connect the JDeveloper Profiler to it. Connecting the Profiler involves first preparing JDeveloper for remote profiling.

To set up a remote profiling session:

1. In the Application Navigator, select the project to be remotely profiled.
2. Select a run configuration and click **Edit**. For more information, see Section 19.3, "How to Configure a Project for Running."
3. In the Remote page under **Profiler** node, select the process type as local or remote.
4. Click **OK** when you are done.
5. Start the Java program session, if you have not already. For more information, see Section 18.10.18, "How to Invoke the Profiler Agent."

To connect remotely:

1. In the Application Navigator, select the project to be remotely profiled.
2. From the Run menu, attach the profiler to CPU Profile or Memory Profile, as desired.
3. In the Attach Profiler to Running JVM dialog, confirm that **Attach to Remote Process** option is selected and host/port information is correct.
4. Click **OK**.
5. In the Profiler tab, click **Begin Use Case** to start profiling a use case.
6. When you are done, detach the profiler. From the **Run** menu, select **Detach**.

> **Note:** The **Connect on Application Startup** and **Begin Use Case on Application Startup** checkboxes in the Profiler page of the Edit Run Configuration dialog have no impact if you attach the profiler to a local process. The checkboxes are applicable when you attach profiler to a remote process only.

18.10.20 How to Dynamically Attach and Detach the Profiler To a Running Process

JDeveloper allows you to dynamically attach and detach a profiler to a running process. This is similar to profiler attached remote profiling, but doesn't require you to specify profiler parameters on the command line when you launch the application. For example, if you are running an application with no plan to profile it when you started it, but later you wish to profile it because it is exhibiting some performance problems.
  ```

In such a scenario, dynamically attaching a profiler to a running process saves you from restarting your JVM to attach a profiler to it.

Dynamically attaching a profiler has its limitations too. You can do CPU time sampling or take memory reference snapshots (heap dumps), but you cannot do method call counts or new/gc memory profiling as they require byte code instrumentation, which can only be requested on the command line before the JVM is launched.

To dynamically attach/detach the profiler to a running process:

1. Start the program outside of JDeveloper, for example, in a different command window.
2. Open JDeveloper and open the Attach Profiler to Running JVM dialog (**Run** > **Attach Profiler** > **CPU Profile** or **Memory Profile)**.
3. In the Attach Profiler to Running JVM dialog, select the **Attach to Local Process** option.
4. From the list of JVMs, select the program's JVM running on the local system.
5. Click **OK**. The profiler connects to the running JVM.

To detach profiler from a running process, select **Detach** from the **Run** menu. If you disconnect the profiler by closing the Profiler tab, you cannot reattach to it. To reattach later, you must use Detach command.

To reattach the profiler to a process:

You can detach and attach the profiler to a process as many times as you want, but you must remember the following points before reattaching:

- Ensure that profiler is detached before reattaching it again.
- When you reattach the profiler, you start a new profiler session, and the data of previous session is lost. To save data of previous session, select **Save As** from the **File** menu.

When you reattach profiler to a process, you can change your configuration settings and reattach with the new configuration settings. For example, if you are profiling CPU, you can change CPU configuration to enable or disable collect I/O time, CPU time, wait time, and so on. If you are memory profiling, you can reattach profiler to take heap dumps. You can also switch back and forth between CPU profiling and memory profiling.

However, you cannot reattach if you change your CPU profiling configuration to count method calls, or memory profiling configuration to New/gc. These require capabilities the profiler cannot acquire after the application starts running. To do either of these, you must launch the application with appropriate profiling parameters and attach profiler using the **Attach to Remote Process** command.

18.10.21 How to Set Profile Points

The Profile Points allow you to set a method filter on a method or class. This means that only stacks that contain those methods, or classes, are reported by the profiler. It is used to filter data so you only see a method, what calls it, and what it calls.

To set a profile point:

1. pen the file in the source editor, and right-click in the left margin next to a line of executable code.

2. From the context menu, choose Toggle Profile Point. The profile point icon is displayed on the left margin of the parent class or method name.

Disabling a Profile Point

You can disable a profile point in any of the following ways:

- In the source editor, right-click the profile point symbol in the left margin and choose **Disable Profile Point**.
- In the Profile Point window (**View** > **Profile Points**) right-click the profile point you want to disable and choose **Disable**.
- To disable all current profile points, right-click in the Profile Points window, and choose **Disable All** from the context menu.

Deleting a Profile Point

When you no longer need to examine the code at a profile point location, you can delete the profile point. You can delete a profile point in any of the following ways:

- In the source editor, right-click the profile point symbol in the left margin and choose.
- In the Profile Point window (**View,** then **Profile Points**) right-click the profile point you want to remove and choose **Delete**.
- To remove all current profile points, right-click in the Profile Points window, and choose **Delete All** from the context menu.

18.10.22 Saving and Opening Profiler Sessions

The profiler allows you to save output for later viewing and analyzing.

To save a running profiler session to disk, in the File menu, select Save As. In the Save As dialog, enter an appropriate name to describe your session, for example, `uianalysis.opr`, or accept the default.

> **Note:** If you are saving an active use case, it will be automatically terminated. The End Use Case toolbar icon is disabled, and Begin Use Case toolbar icon is enabled.

To open a saved profiler session, in the **File** menu, select **Open**, and navigate to the session you want to view.

A saved session is visually distinguishable from an active session by its unique profiler icon. In a saved session, the use case begin and end icons are disabled, but the use case navigation controls are active when multiple use cases are available.

18.10.23 How to Open HPROF Format Heap Dumps

JDeveloper allows you to open heap dumps in HPROF binary format and display them in the profiler. HPROF binary format heap dumps can be created by the HPROF profiler, by the JDK tools jmap and jconsole, or through the Java parameter `-XX:+HeapDumpOnOutOfMemoryError`.

For example, to create the HPROF file using the jmap command, you could enter on the command line:

```
jmap -dump:format=b,file=heap.hprof <jdev process id>
```

For more information about HPROF, see
http://java.sun.com/javase/reference/index.jsp

To open an HPROF binary file:

1. From the **File** menu, select **Open**.
2. Browse and select the HPROF binary format file, and then click **Open**.

The heap dump opens in the profiler References editor tab.

Note: Opening large heap dumps is a slow process and may take several minutes to open. On Windows XP, the Profiler cannot open large (> 990MB) HPROF files.

18.11 Modeling Java Classes

A Java class diagram allows you to visually create classes, interfaces, enums, and inheritance and composition relationships, and to view existing Java classes and interfaces. If you want to visualize a particular facet of your application, add only the classes that contribute to that aspect to the diagram.

To model Java classes you should start with a Java class diagram, although you can subsequently add other elements to the diagram. For more information, see Section 18.11.10, "How to Create a Diagram of Java Classes." You can create and modify the classes that comprise your application directly through the diagram. Changes made on the Java class diagram are immediately available in the Java source editor, and vice versa.

18.11.1 Modeling Dependencies

Dependencies are represented on the diagram as a dashed line with an open arrowhead in the direction of the dependency and are used for documentation purposes only and do not change the underlying Java code.

18.11.2 Creating Java Classes, Interfaces, and Enums

Java classes, interfaces, or enums are created on a diagram by clicking on the **Java Class** icon, **Java Interface** icon or **Java Enum** icon on the Java Component Palette for the diagram, and then clicking on the diagram where you want to create the class. The Java source file for the modeled class or interface is created in the location specified by your project settings.

Java Class, **Java Interface**, and **Java Enum** icons are represented on a diagram as rectangles containing the name and details of the Java class. Java classes and interfaces are divided into compartments, with each compartment containing only one type of information.

An ellipsis (...) is displayed in each compartment that is not large enough to display its entire contents. To view a modeled class so that all the fields and methods are displayed, right-click the class and choose **Optimize Shape Size,** then **Height and Width.**

Each type of class on a diagram is identified by a stereotype in the name compartment. This is not displayed by default.

Members (fields and methods) display symbols to represent their visibility. The visibility symbols are: **+ Public**, **- Private**, **# Protected**. If no visibility symbol is used, the field or method has package visibility.

18.11.2.1 Modeling Java Interfaces

An interface is normally used to group together method signatures for groups of methods that together define a coherent service. Classes that want to provide the service defined by an interface do this by implementing the interface. Interface names must be unique within a namespace. Because interfaces can be used to specify a set of services that other classes provide, they can be used to enforce some level of consistency on those other classes.

Note: Modeled Java interfaces can inherit from other interfaces using extends relationships.

For more information, see Section 18.3.6, "How to Create a New Java Interface."

18.11.2.2 Modeling Inner Java Classes and Inner Java Interfaces

A diagram can include primary or inner classes from different packages, the current application, or from libraries. Inner Java classes and inner interfaces are defined as members of their 'owning' class. Hence, they are also referred as member classes.

Inner classes and inner interfaces are displayed in the inner classes compartment of the modeled Java class or interface on the diagram. Inner classes are prefixed with the term Class, and inner interfaces are prefixed with the term Interface, between the visibility symbol and the class or interface name.

To create an inner class or inner interface on a modeled Java class or interface, either add the inner class to the implementing Java code, or create a new Java class or interface as an internal node on an existing modeled class.

Inner Java classes and inner Java interfaces cannot have the same name as any containing Java class, Java interface or package or contain any static fields or static methods.

18.11.2.3 Modeling Enums

Enumerated types, or enums, containing fields and methods, can be created on a diagram. Enumerated types cannot implement interfaces, extend other classes, or be extended by another class.

18.11.3 Modeling Composition on a Java Class Diagram

A variety of references (previously referred to as associations) can be created quickly between classes and interfaces on a diagram using the various reference icons on the Java Class component palette for the diagram. References created between modeled Java classes are represented as fields in the source code of the classes that implement the references. Compositional relationships are represented on the diagram as a solid line with an open arrowhead in the direction of the reference. Table 18–11 displays the references that can be modeled on a diagram.

Table 18–11 References Between Classes or Interfaces

Reference	Description
Reference (Object)	A singular, direct reference from one class or interface to another. This is represented in the code of the reference's originating class as a field of type `<destination_class>`.
Reference (Array)	A reference to an array of another class or interface. This is represented in the code as an array of type `<destination_class>`.
Reference (Collection)	This is represented in the code as a Collection declaration, and adds an `@associates <{type}>` Javadoc tag to the source to identify this reference as well as the required import java.util.Collection; statement.
Reference (List)	This is represented in the code as a List declaration, and adds an `@associates <{type}>` Javadoc tag to the source to identify this reference as well as the required import java.util.List; statement.
Reference (Map)	This is represented in the code as a Map declaration, and adds an `@associates Javadoc` tag to the source to identify this reference as well as the required import java.util.Map; statement.
Reference (Set)	This is represented in the code as a Set declaration, and adds an `@associates Javadoc tag` to the source to identify this reference as well as the required import java.util.Set; statement.

Note: If you want to quickly change the properties of a reference on a diagram, double-click it to display the Code Editor and change the details of the reference.

Labels are not displayed on references by default. To display the label for a reference, right-click the reference and choose Visual Properties, then select Show Label. The default label name is the field name that represents the reference. If you select this label name on the diagram and change it, an `@label <label_name>` Javadoc tag will be added before the field representing the reference in the code.

You can change the aggregation symbol used on a reference on a diagram by right-clicking the reference, choosing Reference Aggregation Type, then choosing None, Weak (which adds an `@aggregation` shared Javadoc tag to the code representing the reference), or Strong (which adds an `@aggregation` composite Javadoc tag to the code representing the reference). Aggregation symbols are for documentary purposes only.

18.11.4 Modeling Inheritance on a Java Class Diagram

Inheritance structures, which are represented in the Java source as `extends` statements, can be created on a diagram of Java classes using the **Extends** icon on the Java Class Component Palette for the diagram. Extends relationships are represented on the diagram as a solid line with an empty arrowhead pointing towards the extended class or interface.

Where an interface is implemented by a class, this can be created using the **Implements** icon on the Java Component Palette for the diagram. Creating an implements relationship adds `implements` statement to the source code for the implementing class. Implements relationships are represented on the diagram as a

dashed line with an empty arrowhead pointing towards the implemented Java interface.

18.11.4.1 Extending Modeled Java Classes

Extends relationships model inheritance between elements in a class model. Extends relationships can be created between Java classes and between Java interfaces, creating an extends statement in the class definition. Enums cannot extend other classes, or be extended by other classes.

> **Note:** As multiple class inheritance is not supported by Java, only one extends relationship can be modeled from a Java class on a diagram. Multiple extends relationships can be modeled from a Java interface.

18.11.4.2 Implementing Modeled Java Interfaces

Implements relationships specify where a modeled Java class is used to implement a modeled Java interface. This is represented as an implements keyword in the source for the Java class. Implements relationships are represented on class diagrams as dashed lines with an empty arrowhead pointing towards the interface to be implemented. Enums cannot implement interfaces.

If the implemented interface is an extension (using an extends relationship) of other modeled interfaces, this is reflected in the Java source code for the interface.

A class that implements an interface can provide an implementation for some, or all, of the abstract methods of the interface. If an interface's methods are only partially implemented by a class, that class is then defined as abstract.

18.11.5 Modeling Java Fields and Methods

You can create members (fields and methods) of a Java class or interface on a diagram. The fields and methods are added to modeled Java classes and interfaces on a diagram by double-clicking the modeled Java class or interface then adding the field or method using the Java Source Editor.

- Fields are used to encapsulate the characteristics of a modeled Java class or Java interface. All modeled fields have a name, a datatype and a specified visibility.

  When a field or method is displayed on a class on a diagram, it is prefixed with + (if declared as **public**), - (if declared as **private**) or # (if declared as **protected**). Static fields are underlined on the diagram.

- Methods are defined on a class to define the behavior of the class. Methods may have return types, which may be either a scalar type or a type defined by another class.

18.11.6 Modeling Packages on a Java Class Diagram

A package is a general purpose mechanism for organizing elements into groups. It may contain many different types of elements including packages, files, classes and model elements (for example; classes, interfaces, entity objects). Packages may be nested within other packages.

A package owns the elements within it and provides the context and namespace for those elements. Elements owned by the same package must have unique names within the package. Each element is directly owned by a single package, but can be referenced

(imported) from other packages; in other words, referred to by other elements in other packages.

If a package is renamed on a diagram, or moved to another package, the contents of the moved or renamed package will be refactored automatically to reflect this package change.

Package names must be unique within a namespace, even if the names have different capitalization.

Java packages can be either created on a diagram, or dragged onto a diagram from the navigator. To open a diagram for a Java package, right-click the package on the diagram and choose **Drill Down**.

18.11.7 How to Display Related Classes on a Diagram

Java classes and interfaces related to those currently displayed on the diagram can be brought onto the diagram. This includes classes or interfaces that are extended, implemented, or referenced by the selected class or interface.

> **Note:** Where classes are added to a diagram from the project's source path, and are not already part of the project, those classes are automatically added to the current project.

To display related classes on a diagram, use one of the two following ways:

- Select the class or interface, on the diagram, for which you want to display related elements, then choose **Model** > **Show** > **Related Elements**.
- Right-click the class or interface, on the diagram, for which you want to display related elements, then choose **Show** > **Related Elements**.

18.11.8 How to Hide References between Java Classes

Relationships between Java classes or interfaces can be visualized on a diagram using references. You can hide a reference on a diagram.

To hide a reference between Java classes:

- Right-click the reference you want to hide and choose Hide Reference.

To display a hidden reference between Java classes:

1. Double-click the modeled Java class in which the field representing the reference is defined.
2. Click the **Source** tab at the bottom of the editor window.
3. Remove the `attribute` Javadoc tag from above the member representing the reference.

   The hidden reference will be displayed on the diagram.

> **Note:** Hiding a reference between Java classes on a diagram adds a comment to the Java source for the class, so if this class is also on any other diagrams, that reference will also be hidden on those diagrams.

18.11.9 What Happens When You Model a Java Class

The definitions of the classes on a diagram, their members, inheritance, and composition relationships are all derived directly from the Java source code for those classes. These are all created as Java code, as well as being displayed on the diagram. If you change, add to, or delete from, the source code of any class displayed on the diagram, those changes will be reflected on those classes and interfaces on the diagram. Conversely, any changes to the modeled classes are also made to the underlying source code. Some information relating to composition relationships, or references, captured on a Java class diagram is stored as Javadoc tags in the source code.

A Java class diagram can contain shapes from other diagram types (Oracle ADF Business Components, UML elements, Enterprise JavaBeans, and database objects). A form of UML notation is used to display the classes on your diagram. Modeled UML classes can be transformed to modeled Java classes. Likewise, modeled Java classes can be transformed to modeled UML classes. You can annotate a diagram of Java classes using notes, dependency relationships and URL links.

18.11.10 How to Create a Diagram of Java Classes

Java classes, interfaces, and enums can be visually created on a Java class diagram, together with their members, inheritance and composition relationships.

You can create UML classes, UML use cases, offline database objects, business components, Enterprise JavaBeans, and web services on a Java class diagram.

Java classes on a diagram must have valid Java class names and must be unique within the class's package. If you define a modeled Java class as abstract you should create a concrete Java class with a generalization to the abstract Java class. The names of modeled abstract Java classes are displayed in italics.

To create a diagram of Java classes:

1. Create a new diagram using the **Java Class Diagram** icon in the New Gallery.
2. Create the nodes you require on the diagram using the Java Class Component Palette.

   Also, Java classes and interfaces available to the current project can be dragged from the navigator and dropped on the diagram to either modify the code for those classes, or to visualize the structure of existing code.

> **Tip:** You can also annotate your diagram by creating and attaching notes to diagram elements, and adding URL links to other locations such as files or web locations.

18.12 Unit Testing with JUnit

JUnit is an open source regression testing framework for Java. Use JUnit to write and run tests that verify Java code. For detailed information about JUnit, visit the JUnit website, *http://www.junit.org/*

Use JUnit wizards in JDeveloper to create test fixtures, cases, and suites. In addition to wizards for creating test components for generic projects, specialized wizards for business components projects are provided.

JUnit is an optional feature that can be installed and integrated with JDeveloper.

After you install JUnit as an extension, additional online documentation is installed. These help topics appear under Help Table of Contents under the folder **Creating a JUnit Test for a Java Project**.

18.12.1 How to Install JUnit

JUnit is an optional feature that is not distributed with JDeveloper. You must download and install it if you wish to use it.

> **Note:** JUnit is provided under an IBM public license agreement. You must accept to the license agreement before downloading JUnit.

To install JUnit in JDeveloper:

1. Use the IDE Update Wizard to download JUnit from the Oracle Technology Network (OTN).
2. Exit and restart JDeveloper.
3. JUnit will be installed. Use the Extension Manager if you wish to uninstall it.

18.12.2 Creating a JUnit Test for a Java Project

A JUnit test application consists of the following components:

- One or more test cases, which invoke the methods that are to be tested, and make assertions about the expected results. While test case classes generated by default have 'Test' in their names, the user can specify any valid Java name.
- Test fixtures, which provide the state in which the tests are run. Any class can serve as a test fixture, but JDeveloper provides wizards to help you create specialized test fixture classes. While test fixture classes generated by default have 'Fixture' in their names, the user can specify any valid Java name.
- A test suite, which invokes the test cases. Default test suite classes have 'AllTests' in their names.
- A runner, which invokes the test suite and collates and displays the results of the tests.

18.12.3 How to Create a JUnit Custom Test Fixture

A test fixture is a set of objects, having known values, that provide data for the test cases. Any class can serve as a test fixture, but JDeveloper provides wizards to help you create custom test fixture classes and various specialized test fixture classes.

To create a JUnit custom test fixture class:

1. In the Navigator, select the project.
2. Choose **File** > **New** to open the New Gallery.
3. In the Categories tree, expand **General** and select **Unit Tests**.
4. In the Items list, double-click **Test Fixture.**
5. Complete the wizard to create the test fixture class. The class created by the wizard will be opened for editing.
6. In the Categories tree, expand **General** and select **Unit Tests (JUnit)**.
7. In the **Items** list, double-click **Custom Test Fixture**.

8. Complete the wizard to create the test fixture class. For more information at any time, press F1 or click Help from within the dialog.
9. In the Categories tree and expand the **Unit Tests**.
10. Click **Test Suite** > **OK**.
11. In the Items list, double-click **Test Fixture**.
12. Complete the wizard to create the test fixture class. For more information at any time, press F1 or click Help from within the dialog.

    The class created by the wizard will be opened for editing.
13. Modify the file as needed. In particular, to the setUp() method add code that initializes test fixture objects, and to the tearDown() method add code that releases any resources they acquire.

18.12.4 How to Create a JUnit JDBC Test Fixture

A test fixture is a set of objects, having known values, that provide data for the test cases. A JDBC test fixture provides code that establishes a database connection for the test cases to use.

To create a JUnit JDBC test fixture class:

1. In the navigator, select the project.
2. Choose **File,** then **New** to open the New Gallery.
3. In the Categories tree, expand **General** and select **Unit Tests (JUnit)**.
4. In the Items list, double-click **JDBC Test Fixture**.
5. Complete the dialog to create the test fixture class. For more information at any time, press F1 or click Help from within the dialog.

    The class that was created will be opened for editing.
6. Modify the file as needed. In particular, to the `setUp()` method add code that initializes test fixture objects, and to the `tearDown()` method add code that releases any resources they acquire.

18.12.5 Creating a JUnit Test Case

A test case class has one or more methods that perform tests by calling JUnit assertions. Example 18–7 shows a typical test case in JUnit 3.x. It passes test fixture data to the method being tested, and then compare the result with a known value to confirm that it is what is expected.

Example 18–7 JUnit 3.x Test Case

```
public void testCountChars()
{
      int expected = 4;
      int actual = fixture1.countChars('a');
      assertEquals(expected, actual);
}
```

Example 18–8 JUnit 4 Test Case

```
@Test
public void testCountChars()
```

```
{
      int expected = 4;
      int actual = fixture1.countChars('a');
      Assert.assertEquals(expected, actual);
}
```

In the test case shown in Example 18–8, `countChars()` is being tested, and the result of the test is checked by `assertEquals()`, which is one of a variety of assertion methods defined in the JUnit Assert class. The state of the test fixture, `fixture1`, is established in the `setUp()` method, which will have been called before the test case is called, as shown in Example 18–9.

Example 18–9 setUp() method

```
protected void setup() throws Exception
{
fixture1 = new StringFixture("Goin' to Kansas City, Kansas City, here I come.");
}
```

To create a JUnit test case class:

1. In the navigator, select the project or the particular class that you want to test.
2. Choose **File,** then **New** to open the New Gallery.
3. In the Categories tree, expand **General** and select **Unit Tests (JUnit)**.
4. In the Items list, double-click **Test Case**.
5. Complete the wizard to create the test fixture class. The class created by the wizard will be opened for editing.

18.12.6 Creating a JUnit Test Suite

A test suite is a class that invokes test cases.

The JUnit Test Suite wizard has options to insert a main() method and a call to a TestRunner class. Within JDeveloper, this will open the JUnit TestRunner log window to display the test results. Edit the method if you wish to use a different test runner.

In the JUnit 3.x test suite shown in Example 18–10, the `suite()` method creates a TestSuite instance and adds the test cases to it. Edit this method if you wish to add or remove test cases.

Example 18–10 JUnit 3.x Test Suite

```
public class AllTests {
    public static Test suite() {
    TestSuite suite;
    suite = new TestSuite("project1.AllTests");
    return suite;
    }
```

In the JUnit 4 test suite shown in Example 18–11, the test case classes are written with @Suite and @RunWith annotations.

Example 18–11 JUnit 4 Test Suite

```
@RunWith(Suite.class)
@Suite.SuiteClasses( {})
public class AllTests1 {
```

```
    public static void main(String[] args) {
    String[] args2 = { AllTests1.class.getName() };
    org.junit.runner.JUnitCore.main(args2);
    }
}
```

To create a JUnit test suite class:

1. In the navigator, select the project.
2. Choose **File,** then **New** to open the New Gallery.
3. In the Categories tree, expand **General** and select **Unit Tests (JUnit).**
4. In the Items list, double-click **Test Suite**.
5. Complete the wizard to create the test fixture class. The class created by the wizard will be opened for editing.
6. Modify the file as needed. In particular:
   - In the suite() method add the test cases.
   - In the main() method replace the runner invocation, if desired.

18.12.7 How to Add a Test to a JUnit Test Case

You can add a unit test for a method to an existing JUnit test case class.

To add a test to a JUnit test case class:

1. In the code editor, select a method for which you want to create a new unit test.
2. From the main menu, choose **Source** > **New Method Test**. The New Method Test dialog is opened.
3. Select **Add to Existing TestCase Class.**
4. From the **Class Name** dropdown box, or by using **Browse**, select the test case class that you want to add the new test to.
5. To add the new test to the test case, click OK.

18.12.8 How to Update a Test Suite with all Test Cases in the Project

You update a test suite with all test cases in a project.

To update a test suite:

1. In a class that has a suite() method, from the main menu, choose **Source** > **Refresh Test Suite**. The Refresh Test Suite dialog is opened.
2. Ensure that all items in the list of test cases are checked.
3. To update the test suite, click **OK.**

18.12.9 How to Run JUnit Test Suites

When your test suite has been successfully compiled you can run it.

To run a JUnit test suite:

1. In the navigator, select the test suite class.
2. Right click it, and choose **Run**.

The test will execute, and the test runner will display the results.

19

Running and Debugging Java Programs

This chapter describes how to use the tools and features provided by JDeveloper to run and debug Java programs. For information about writing and compiling a Java program, see Chapter 18, "Programming in Java."

This chapter includes the following sections:

19.1 About Running and Debugging Java Programs

JDeveloper offers several techniques to monitor and control the way Java programs run. When running Java programs, JDeveloper keeps track of processes that are run and debugged, or profiled. In addition, JDeveloper offers both local and remote debugging of Java, JSP, and servlet source files.

19.2 Understanding the Run Manager

The Run Manager keeps track of processes that are run, debugged, or profiled. When two or more such processes are active at the same time, the Run Manager window is automatically displayed. When a process has completed, it is automatically removed from the Run Manager.

To open the Run Manager:

- Choose **View** > **Run Manager** from the main menu.

To terminate a process with the Run Manager:

- Right-click a process in the Run Manager and choose **Terminate** from the context menu.

To view the Run Log:

- Right-click a process in the Run Manager and choose **View Log** from the context menu.

19.3 How to Configure a Project for Running

Settings that control the way programs are run - such as the target, launch options, and the behavior of the debugger, logger, and profiler - are collected in run configurations.

A project may have several run configurations, each set up for a specific facet of the project or phase of the development process. A run configuration can be bound to the project and be available to all who work on the project, or it can be custom configuration, for your use only.

A default run configuration is created for each new project. You can modify run configurations, and you can create a new configuration by copying an existing one.

To select a run configuration:

1. From the main menu choose **Application** > **Project Properties**.
2. Select the **Run/Debug** page.
3. From the **Run Configurations** list, select a run configuration.

To modify a run configuration:

1. Select a run configuration as described above.
2. Click **Edit**.

   The Edit Run Configuration dialog is opened.
3. Make the required changes to the preferences on the dialog pages.

   For help while using the dialog pages, press **F1**.

To create a run configuration:

1. Select a run configuration as described above.
2. Click **New**.

   The Create Run Configuration dialog is opened.
3. In the **Name** box, enter a name for the new run configuration.
4. In the **Copy Settings From** dropdown box, choose an existing run configuration to copy from.
5. To create a new run configuration having the same settings as the one it was copied from, click **OK**.

19.4 Running an Applet

JDeveloper lets you run applets in the AppletViewer or in the Integrated WebLogic Server instance. The AppletViewer provides a test bed to run your applet without launching the web browser. When you want to run your applet in a browser, you can run in the Integrated WebLogic Server instance.

After creating your applet and ensuring that the classpath is set up properly in the HTML file, you can run it by executing the Run command in one of the following ways:

1. In the navigator, select the HTML file that contains the `<APPLET>` tag.
2. To run the applet, right-click the HTML file and choose **Run**.
3. In the dialog, select the way you want to start the target applet and click **OK**:
   - **In AppletViewer**: The applet is launched in the Applet Viewer.
   - **In the Server Instance**: The integrated server is started and the applet is run in the server.

19.4.1 Using an HTML File to Store Arguments

An applet runs in an HTML page, from which it obtains its display size and other parameters. To run an applet in JDeveloper, you need to provide an HTML file containing the appropriate `<APPLET>` tag.

Parameter names are case-sensitive, although parameter tags are not:

```
<APPLET CODE="foo.class" WIDTH=200 HEIGHT=20> </APPLET>
```

You can also pass parameters to the applet by including a `<PARAM>` tag between the `<APPLET>` and `</APPLET>` tags:

```
 <PARAM NAME=foo VALUE=true>
```

Example 19–1 shows an HTML fragment that is used to pass parameters.

Example 19–1 HTML Fragment That is Used to Pass Parameters

```
<H1>Test File</H1>
<HR>
<APPLET CODE="Test3.class" WIDTH=500 HEIGHT=120>
<PARAM NAME=level VALUE="8">
<PARAM NAME=angle VALUE="45">
<PARAM NAME=delay VALUE="1000">
<PARAM NAME=axiom VALUE="F">
<PARAM NAME=incremental VALUE="true">
<PARAM NAME=incremental VALUE="true">
</APPLET>
<HR>
<A HREF="Test3.java">The source</A>
...
```

19.5 How to Run a Project or File

After building your project or file, you can run it.

To run a project or file:

1. In the navigator, select the project or file you want to run.
2. Run an application in any of these ways:
   - For a project only, from the main menu choose **Run** > **Run Project**.
   - From the context menu, select **Run**.
   - Click the **Run** icon on the toolbar.

   The main method of your Java application is started.

19.5.1 How to Run a Project from the Command Line

The following conditions must exist to run a project from the operating system command line:

- The project is a standalone executable.
- You must select the class file containing the application main() method.

To launch an application:

Enter the following:

```
java -cp <jdev_install>\jdeveloper\jdev\mywork\Workspace1\Project1\classes
package1.Application1
```

To launch the executable JAR file from the command line:

Enter the following:

```
java -jar <application>.jar
```

where `<application>` is your JAR file name.

19.5.2 How to Change the Java Virtual Machine

You may need to change the Java Virtual Machine (VM) for which you are developing because of operating system considerations. For example, for client-side applications, you would use the HotSpot Client VM, whereas for executing long-running server applications, you would use the Server VM.

To change the Java Virtual Machine:

1. Right-click a project in the navigator and choose **Project Properties** from the context menu.
2. Open the Run/Debug/Profile page.
3. Select a run configuration and click **Edit**.

   This opens the Edit Run Configuration dialog.
4. On the Launch Settings page, in the **Virtual Machine** list box, select an available option.

   The selected JVM is used when running and debugging the project.
5. Click **Help** for additional information.

19.5.3 Setting the Classpath for Programs

When you run a Java program from the command line, you must provide the Java Virtual Machine (JVM) with a list of the paths to the class files and libraries that comprise your application. The form of the classpath changes depending on the method you use to run the Java program.

Your Java classes can be stored in Java Archive (`*.jar`) files, or as separate class (`*.class`) files in their package directory. There are differences in the ways Java handles JAR files and package directories.

- When you refer to JAR files in your `CLASSPATH`, you use the fully qualified path to the JAR file.

- When you refer to package directories in your CLASSPATH, you use the path to the parent directory of the package.
- You can refer to both JARs and package directories in a CLASSPATH statement. When you refer to more than one CLASSPATH in the same statement, each CLASSPATH is separated with a semicolon(;).

Once you have defined the classpath, you pass the value to the JVM in different ways, depending on how you run your Java program.

- Set the CLASSPATH environment variable to run a standalone application using java.exe.
- Set the CLASSPATH environment variable to.use the -classpath option of java.exe.
- Embed the CLASSPATH in the <APPLET> tag of an .html file to run an applet in an Internet browser.

You have the option of using either the -classpath option when calling an SDK tool (the preferred method) or by setting the CLASSPATH environment variable.

19.5.3.1 Setting the CLASSPATH Environment Variable (for java.exe)

The java.exe is included as part of the Java2 Standard Edition (J2SE). It is intended to be used as a development tool, and is not licensed for distribution with your Java programs. It is used to test your Java applications from the command prompt.

In order to run a Java application from the command prompt, the system environment variable CLASSPATH must be defined to include all of the classes necessary to run your program. This includes any library classes provided with JDeveloper that your program uses.

19.5.3.2 Using the JDeveloper Library CLASSPATH

JDeveloper ships hundreds of library classes to help you generate your Java programs. The classes come from J2SE, third-party developers, and Oracle Corporation. Each of the libraries is kept separate for easy upgrade. As a result, many archive files may need to be included in your classpath to ensure that any program you create in JDeveloper can be run from the command prompt.

Oracle recommends that you list only the paths to each of the libraries that your project uses. If you list paths that your project does not use, your program will still run, but for performance reasons, you will want to eliminate any unnecessary libraries.

The command to set the CLASSPATH variable takes this format:

```
set CLASSPATH=path1;path2;path3;...path_n
```

> **Note:** Never use quotation marks in the classpath even when there is a space character in one of the paths.

19.5.3.3 Setting the CLASSPATH to Include Your Projects

If you have used the default directory for your output path, you can test your Java application using java.exe by appending the following directory to your classpath:

```
C:\<jdev_install>\jdeveloper\jdev\mywork\Workspace1\Project1\classes
```

Having set this variable, you can use `java.exe` to run your application from the output directory `mywork`.

If you have deployed your Java program to any other directory, you need to add the path to the parent directory of the application package.

The `CLASSPATH` variable is a long string that can be difficult to type accurately. To save time and reduce errors, you can set the `CLASSPATH` as a system environment variable.

19.5.3.4 Setting the CLASSPATH Parameter (for java.exe)

The Java Runtime Engine (`java.exe`) doesn't use the `CLASSPATH` environment variable. The `CLASSPATH` must be included as a parameter to the java.exe command. The format for the command is:

```
java -cp <classpath> package.Application
```

Where *classpath* is the complete `CLASSPATH` to your Java program and the dependency classes it uses. The quotation marks are optional if there are no spaces in any of the `CLASSPATH` directory names.

19.5.3.5 Embedding the CLASSPATH Parameters in the <APPLET> Tag

When running applets, the browser uses a `CLASSPATH` you supply in the `ARCHIVE` and `CODEBASE` parameters to the `<APPLET>` tag in the host `*.html` file.

The `CODEBASE` parameter sets the root directory where the Internet browser will look for your class files. If the classes are stored in the same directory as the HTML page calling the applet, you can omit the `CODEBASE` parameter entirely. Otherwise, use either an absolute or relative path from the HTML file to the location of the `CODEBASE` directory. Use forward slashes (/), not backslashes(\) to indicate directories.

The `ARCHIVE` parameter lists the locations and names of the JAR files that contain your program and its supporting library files, similar to the `CLASSPATH` used with applications. There are three important differences:

- The names of the Java Archive files are separated by commas (,), not semicolons(;).
- If the Java Archive files are in subdirectories of the `CODEBASE`, use forward slashes (/), not backslashes(\), to indicate directories.
- Due to the limitations enforced by the Java security model for applets, classes referenced by your `ARCHIVE` parameter can only be located in subdirectories of the `CODEBASE` directory. This means that if you attempt to set the location of an archive file using a parent directory (../) you will receive a security violation error.

19.6 About the Debugger

Debugging is the process of locating and fixing errors in your programs. The JDeveloper integrated debugger enables you to debug Java applications, applets, servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans (EJBs). You can debug a single or several objects on the same or different machine as JDeveloper supports distributed debugging.

The Debugger provides you with a number of features to investigate your code, and identify and fix problem areas. Two types of debugging are available to analyze your code - local and remote.

A local debugging session is started by setting breakpoints in source files, and then starting the debugger. When debugging an application such as a servlet in JDeveloper,

you have complete control over the execution flow and can view and modify values of variables. You can also investigate application performance by monitoring class instance counts and memory usage. JDeveloper will follow calls from your application into other source files, or generate stub classes for source files that are not available.

Remote debugging requires two JDeveloper processes: a *debugger* and a *debuggee* which may reside on a different platform. Once the debuggee process is launched and the debugger process is attached to it, remote debugging is similar to local debugging.

JDeveloper provides a number of special-purpose debugging windows that enable you to efficiently identify the problematic areas in your code:

- The Breakpoints Window displays the breakpoints for the current workspace and project. For more information, see Section 19.7.1, "Using the Breakpoints Window."
- The Smart Data Window displays the data which is being used in the code that you are stepping through. For more information, see Section 19.7.2, "How to Use the Smart Data Window."
- The Data Window displays the arguments and local variables for the current context. Note that **Full Debug Info** must be selected in the Compiler page of the Project Properties dialog. For more information, see Section 19.7.3, "How to Use the Data Window."
- The Watches Window displays the values for a watched program. A watch evaluates an expression according to the current context. If you move to a new context, the expression is reevaluated for the new context. For more information, Section 19.7.4, "How to Use the Watches Window."
- The Inspector Window displays a single data item in its own floating window. An inspector evaluates an expression according to the current context. For more information, see Section 19.7.5, "How to Use the Inspector Window."
- The Heap Window displays information about the heap in the program you are debugging and helps you to detect memory leaks in your program. For more information, see Section 19.7.6, "How to Use the Heap Window."
- The Stack Window displays the call stack for the current thread. For more information, see Section 19.7.7, "How to Use the Stack Window."
- The Classes Window displays information about the classes which have been loaded as your application runs, including the name and package of each class. The debugger can also display the number of live instances of each class and the amount of memory being consumed by those instances. For more information, Section 19.7.8, "How to Use the Classes Window."
- The Monitors Window displays information for active monitors in your application, as well as information about the status of threads accessing those monitors. This window is useful for examining deadlocks and other thread synchronization problems. For more information, see Section 19.7.9, "How to Use the Monitors Window."
- The Threads Window displays the threads and the thread groups, highlights the current thread, and shows the name, status, priority, and group of each thread. For more information, see Section 19.7.10, "How to Use the Threads Window."

You can open the debugger windows by choosing **View** > **Debugger**.

19.6.1 Understanding the Debugger Icons

Table 19–1 contains the various JDeveloper debugger and runner icons. These icons are available from areas in the JDeveloper user interface, including the Debugger window and the Log window.

Table 19–1 Debugger and runner icons

Icon	Name	Description
	Array	Represents an array class in any JDeveloper data-related window.
	Add Breakpoint	Represents the Breakpoint toolbar button used to create a breakpoint.
	Breakpoints menu	Represents the **View** > **Debugger** > **Breakpoints** menu option or the tab icon for the Breakpoints window.
	Class	Represents the **View** > **Debugger** > **Classes** menu option, the tab icon for the Classes window and a class in the Classes window (grayed if the class has tracing disabled).
	Class Without Line Number Tables	Appears in the Classes window. Represents a class which does not have line number tables (obfuscated class)
	Current Execution Point	Represents the current execution point shown in the source editor margin which you can display by choosing the **Run** > **Show Execution Point** menu option.
	Current Thread	Represents the current thread in the Threads window.
	Data	Represents the **View** > **Debugger** > **Data** menu option; the **View** > **Debugger** > **Smart Data** menu option; and the tab icon for the Data window and Smart Data window.
	Debug (Shift + F9)	Represents the **Run** > **Debug <*project_name*>** menu option; the debug toolbar button, a debugging process contained in the processes folder in the Run Manager Navigator, a log page for a debugging process, the debug layout, and the Remote Debugging and Profiling Project Wizard
	Debug Listener Node	Represents a debug listener node in the **Run** > **Manager** navigator.
	Debug with Diagram	Represents the **Run** > **Debug with Diagram <*project_name*>** menu option. Lets you create a UML sequence diagram while debugging.
	Disabled Breakpoint	Represents a disabled breakpoint in the source editor margin and a disabled breakpoint in the Breakpoints window. The icon also represents the Breakpoint toolbar button to disable a breakpoint
	Delete Breakpoint	Represents the Breakpoint toolbar button to remove a breakpoint.
	Edit Breakpoint	Represents the Breakpoint Toolbar button, which you can use to edit the selected breakpoint

Table 19–1 (Cont.) Debugger and runner icons

Icon	Name	Description
	Garbage Collection	Represents the **Run** > **Garbage Collection** menu option and the Garbage Collection toolbar button which you can click
	Interface	Represents an interface in the Classes window
	Heap	Represents the **View** > **Debugger** > **Heap** menu option and the tab icon for the Heap window
	Heap Folder	Represents a folder in the Heap window.
	Method	Represents a method in the Stack window
	Monitors	Represents the **View** > **Debugger** > **Monitors** menu option and the tab icon for the Monitors window.
	Object	Represents an object in any JDeveloper data-related window
	Package	Represents a package in the Classes window (grayed if the package has tracing disabled)
	Pause	Represents the **Run** > **Pause** menu option and the Pause toolbar button which you can click.
	Primitive	Represents a primitive item in any JDeveloper data-related window.
	Resume	Represents the **Run** > **Resume** menu option and the Resume toolbar button which you can click.
	Run	Represents a running process in the Run Manager navigator, in a log page for a running process, and in the toolbar to run the selected node.
	Run to Cursor (F4)	Represents the **Run** > **Run to Cursor** menu option. Lets you run to a specified location and execute the code until it reaches that location
	Stack	Represents the **View** > **Debugger** > **Stack** menu option and the tab icon for the Stack window.
	Stack Folder	Represents the static folder in the Data window
	Step to End of Method	Represents the **Run** > **Step to End of Method** menu option and the Step to End of Method toolbar button which you can click.
	Step Into (F7)	Represents the **Run** > **Step Into** menu option and the Step Into toolbar button which you can click.
	Step Out	Represents the **Run** > **Step** > **Out** menu option and the Step Out toolbar button which you can click.
	Step Over	Represents the **Run** > **Step Over** menu option and the Step Over toolbar button which you can click.
	Terminate	Represents the Terminate toolbar button which you can click to stop debugging your application.
	Thread	Represents the **View** > **Debugger** > **Thread** menu option and the tab icon for the Thread window.
	Threads	Represents the **View** > **Debugger** > **Threads** menu option and the tab icon for the Threads window.

Table 19–1 (Cont.) Debugger and runner icons

Icon	Name	Description
	Thread Group	Represents a thread group in the Threads window.
	Unverified Breakpoint	Represents an unverified breakpoint in the source editor margin, and an unverified breakpoint in the Breakpoints window

19.6.2 How to Debug a Project in JDeveloper

Your code must be compiled with debugging information before you can make use of some of the debugger features such as viewing arguments and local variables in the Data window.

To set breakpoints and step through your code:

1. In a source editor, set a breakpoint on an executable statement by clicking in the margin to the left of the statement. For more information, see Section 19.8, "Managing Breakpoints."

   The unverified breakpoints icon appears in the left margin.

2. Select **Run** > **Debug** *[filename.java]*.

   The class runs and stops at the first breakpoint.

3. From the toolbar, click **Step Into** to trace into a method call or click **Step Over** to step over a method call.

4. Look in the Stack window to examine the sequence of method calls that brought your program to its current state. Double-click a method to display the associated source code in the source editor.

5. In the Smart Data and Data windows, examine the arguments and variables.

6. Display the Threads window to see the status of other threads in your program.

To edit and recompile:

1. When you have found lines of code to change, you can end the debugging session by clicking **Terminate** on the toolbar, or by choosing **Run** > **Terminate**.

2. Edit your code in the source editor.

3. In the navigator, click the appropriate object node.

4. Choose **Run** > **Build <*filename.java*>** from the main menu. The affected files in your project are recompiled, and you can run the debugger again.

19.6.3 How to Debug ADF Components

JDeveloper allows you to debug with breakpoints using the ADF Declarative Debugger. If an error cannot be easily identified, you can use the ADF Declarative Debugger in JDeveloper to set breakpoints. When a breakpoint is reached, the execution of the application is paused and you can examine the data that the Oracle ADF binding container has to work with, and compare it to what you expect the data to be. Depending on the types of breakpoints, you may be able to use the step functions to move from one breakpoint to another.

JDeveloper provides three windows for debugging ADF components:

- The ADF Data Window displays relevant data based on the selection in the ADF Structure window when the application is paused at a breakpoint. For more information, see Section 19.7.3, "How to Use the Data Window."
- The EL Evaluator Window evaluates EL Expressions when a breakpoint is reached during a debugging session. Only JSF applications can utilize the EL Evaluator.
- The ADF Structure Window displays a tree structure of the ADF runtime objects and their relationships when the application is stopped at a breakpoint. For more information, see Section 3.11.6, "Structure Window."

You can control what type of information is displayed in each of the debugger windows. To see what options are available in each window such as which columns to display, right-click in a window and choose **Preferences** from the context menu. Or, you can choose **Tools** > **Preferences** from the main menu and expand the Debugger node to display a preferences page for each debugger window. You can also save the debug information as text or HTML output file. For more information. see Section 19.6.7, "How to Export Debug Information to a File."

To use the JDeveloper debugger to control the execution of a program:

1. Run to a breakpoint. For more information, see Section 19.8, "Managing Breakpoints."

   A breakpoint is a trigger in a program that, when reached, pauses program execution. This allows you to examine the values of some or all of the program variables. When your program execution encounters a breakpoint, the program pauses, and the debugger displays the line containing the breakpoint in the source editor.

2. Step into a method and execute a single program statement at a time. For more information, see Section 19.6.11, "Stepping Into a Method."

   If the execution point is located on a call to a method, the Step Into command steps into that method and places the execution point on the method's first statement.

3. Step over a method. For more information, see Section 19.6.12, "Stepping Over a Method."

   If you issue the Step Over command when the execution point is located on a method call, the debugger runs that method without stopping, instead of stepping into it. Program statements are executed one at a time.

4. Run to the cursor location. For more information, see Section 19.6.16, "How to Run to the Cursor Location."

   This allows you to go to a particular location in the program without having to single step or set a breakpoint.

5. Pause and resume the debugger. For more information, see Section 19.6.17, "How to Pause and Resume the Debugger."

   You can pause your program when the program is running in the debugger. You can then use the debugger to examine the state of your program with respect to this program location. When you have finished examining that part of the program, you can then continue running the program.

6. Terminate a debugging session. For more information, see Section 19.6.18, "How to Terminate a Debugging Session.".

When finished, you can modify program values as a way to test hypothetical bug fixes during a program run. If you find that a modification fixes a program error, exit the

debugging session, fix your program code, and recompile the program to make the fix permanent.

19.6.4 How to Configure a Project for Debugging

JDeveloper allows you to control how your program is debugged, including enabling and disabling packages and classes and configuring remote debugging options.

To configure debugger and remote debugger options in JDeveloper:

1. Choose **Application** > **Default Project Properties** (to set preferences that apply to all projects) or choose **Application** > **Project Properties** (to set preferences that apply only to the current project).
2. Select the **Run/Debug/Profile** node.
3. Select a run configuration. For more information, see Section 19.3, "How to Configure a Project for Running.".
4. Click **Edit**.
5. Select the Debugger node.
6. Set the options on the Debugger and Remote pages.
7. Click **OK** when finished.

19.6.5 How to Set the Debugger Start Options

By setting up the debugger start option, you are specifying how you would like the debugger to behave when you start a new debugging session. Specifically, decide if you want the debugger to execute until a breakpoint is reached, or if you want the debugger to stop when it reaches your project's code (for example, at the beginning of your application's main method).

To set the debugger start options:

1. From the main menu choose **Tools** > **Preferences** and open the Debugger page.
2. Select a Start Debugging Option:
   - **Run Until a Breakpoint Occurs**

     When you start debugging, the debugger will let the program you are debugging execute until a breakpoint is reached.
   - **Step Over**

     When you start debugging, the debugger will let the program you are debugging execute until a method in a tracing-enabled class is reached, but it will not stop in a class static initializer method.
   - **Step Into**

     When you start debugging, the debugger will let the program you are debugging execute until any method, including a class static initializer method, is reached.

19.6.6 How to Launch the Debugger

You must build the project before debugging it.

To build a project and start the debugger:

1. In the Application Navigator, select the project.
2. Right-click and choose **Project Properties**. The Project Properties dialog opens.
3. Open the **Compiler** page.
4. If not already enabled, select **Full Debug Info**.
5. Click **OK** to close the dialog.
6. Use one of the following methods to start the debugger:
   - To start the debugger using the current run configuration, from the main menu choose **Run** > **Debug <*project name*>**.
   - To start the debugger using your choice of run configuration, select the dropdown menu beside the **Debug** icon on the toolbar and click the required run configuration name.

If the project builds successfully, the debugger starts.

19.6.7 How to Export Debug Information to a File

You can export debug information generated by the JDeveloper debugger to either a text or HTML output file from within any of the debugger windows.

To export debug information to file:

1. Start debugging by clicking **Debug** from the toolbar.
2. Once the debugger has stopped at a breakpoint, locate the debugger window containing the information you would like to export.
3. Right-click in a debugger window and choose **Preferences** from the context menu.
4. In the appropriate **Preferences - Debugger** page below **Columns**, select which columns you want to show or hide in the debugger window and output file. Click **OK** to close the Preferences dialog.
5. In the debugger window, right-click and choose **Export**.
6. In the Export dialog, enter the name of the file. The output file is saved as a text file with tabs between columns and new lines between rows. To export to an HTML file, add the extension as `.html` or `.htm` (case-insensitive).

If the project builds successfully, the debugger starts.

19.6.8 Using the Source Editor When Debugging

When the debugger stops (for example, at a breakpoint after completing a step command, or when paused), the source file for the current class will open in the source editor and will be marked with the execution point, as shown in Figure 19–1.

Figure 19–1 Execution Point Icon

If JDeveloper cannot locate the source file for the class while debugging, the Source Not Found dialog is displayed prompting you for the source file location.

You can use the source editor to debug in the following ways:

- To set a breakpoint, click in the source editor's margin.
- To remove a breakpoint, click the breakpoint in the source editor's margin.

Figure 19–2 Breakpoint Icon

Using Context Menu Items

The debugger adds several menu items to the source editor's context menu including those shown in Table 19–2.

Table 19–2 Context Menu Items

Item	Function
Run to Cursor	Lets you run to the current location of the cursor and execute the code until it reaches that location.
Watch (Ctrl+F5)	Lets you add an expression to the Watches Window.
Inspect	Lets you open up a floating Inspector window.
Step Into Method at Cursor	Executes Run to Cursor, and then steps into the method that the cursor is currently on.

Using Tooltips

The debugger will show tool tips in the source editor if you hover the mouse over the name of a data item. By default, the tooltip will show the name, value, and type of the data item; providing an easy way to quickly inspect a data item without adding it in Data window or Watches window. If the data item is an array or object, you can inspect children of the selected item deep in the object hierarchy. The tooltip displays 20 children data items, use the navigation buttons to view remaining data items.

The columns which display in the tooltip depend on the column settings that were enabled in the **Tools** > **Preferences – Debugger – Tooltip** page.

If the project builds successfully, the debugger starts.

19.6.9 Using Java Expressions in the Debugger

Java expressions are used in the Watches window, Inspector window, Breakpoint Conditions, and Breakpoint Log Expressions. The debugger accepts Java expressions in the forms shown in Table 19–3.

Table 19–3 Java Expressions Accepted by Debugger

Java Expression	Form
Simple variable name	`rect`
Field access	`rect.width`
Method call	`myString.length()`
Array element	`myArray[3]`
Array length	`myArray.length`
Comparison operation	`rect.height == 100` `myArray.length > 7`

Table 19–3 (Cont.) Java Expressions Accepted by Debugger

Java Expression	Form
Arithmetic operation	`rect.width * rect.height` `x + y + z`
Logical operation	`frame1.enabled && frame1.visible` `textField1.hasFocus || textField2.hasFocus`
Instance of operator	*<my_value>* `instanceof java.lang.String`
Shift operator	`x << 2` `y >> 1`
Binary Operator	`keyEvent.modifiers &` `java.awt.event.InputEvent.CTRL_MASK`
Question-colon operation	`y>5 ? y*7 : y*4`
Static field name	`java.awt.Color.pink`
Fully qualified class name	`java.awt.Color`

If the project builds successfully, the debugger starts.

19.6.10 Moving Through Code While Debugging

The JDeveloper debugger lets you control the execution of your program; you can control whether your program executes a single line of code, an entire method, or an entire program block. By manually controlling when the program should run and when it should pause, you can quickly move over the sections that you know work correctly and concentrate on the sections that are causing problems. For more information, see Section 19.6.5, "How to Set the Debugger Start Options."

The debugger lets you control the execution of your program in the following ways:

- Stepping Into a Method
- Stepping Over a Method
- Controlling Which Classes are Traced Into
- Locating the Execution Point for a Thread
- Running to the Cursor Location
- Pausing and Resuming the Debugger
- Terminating a Debugging Session

The Step Into and Step Over commands offer the simplest way of moving through your program code. While the two commands are very similar, they each offer a different way to control code execution.

The smallest increment by which you step through a program is a single line of code. Multiple program statements on one line of text are treated as a single line of code – you cannot individually debug multiple statements contained on a single line of text. The easiest approach is to put each statement on its own line. This also makes your code more readable and easier to maintain.

19.6.11 Stepping Into a Method

The **Step Into** command executes a single program statement at a time. If the execution point is located on a call to a method, the **Step Into** command steps into that method and places the execution point on the method's first statement.

If the execution point is located on the last statement of a method, choosing Step Into causes the debugger to return from the method, placing the execution point on the line of code that follows the call to the method you are returning from.

The term *single stepping* refers to using **Step Into** to run successively though the statements in your program code.

You can step into a method in any of the following ways:

- Select **Run** > **Step Into**.
- Press **F7**.
- Click the **Step Into** button from the toolbar.

Figure 19–3 Step Into Button

Unlike previous JDeveloper releases, you cannot start debugging by pressing the **Step Into** button. Step Into will only cause stepping on an already-started debugging process.

When you set the debugger to start by stepping into, the debugger will let the program you are debugging execute until a method in a tracing-enabled class is reached.

As you debug, you can step into some methods and step over others. If you are confident that a method is working properly, you can step over calls to that method, knowing that the method call will not cause an error. If you aren't sure that a method is well behaved, step into the method and check whether it is working properly.

19.6.12 Stepping Over a Method

The **Step Over** command, like **Step Into**, enables you to execute program statements one at a time. However, if you issue the **Step Over** command when the execution point is located on a method call, the debugger runs that method without stopping (instead of stepping into it), then positions the execution point on the statement that follows the method call.

If the execution point is located on the last statement of a method, choosing Step Over causes the debugger to return from the method, placing the execution point on the line of code that follows the call to the method you are returning from.

You can step into a method in any of the following ways:

- Select **Run** > **Step Over**.
- Press **F8**.
- Click the **Step Over** button on the toolbar.

Figure 19–4 Step Over Button

Unlike previous releases of JDeveloper, you cannot start debugging by pressing the **Step Over** button. Step Over will cause stepping only on an already-started debugging process.

When you set it to start by stepping over, the debugger will let the program you are debugging execute until a method in a tracing-enabled class is reached, but it will not stop in class static initializer method.

As you debug, you can step into some methods and step over others. If you are confident that a method is working properly, you can step over calls to that method, knowing that the method call will not cause an error. If you aren't sure that a method is well behaved, step into the method and check whether it is working properly.

19.6.13 Controlling Which Classes Are Traced Into

Normally, you should set the tracing include and exclude lists in the project properties before you start debugging. However, if you need to change the tracing include and exclude lists, you can do so from the Classes window. Right-click in the Classes window and choose **Tracing** from the context menu. The Tracing dialog appears in which you can adjust the tracing include and exclude lists.

When you specify a package to be included or excluded from tracing, all descending classes within that package are included or excluded as well unless you've specified them individually.

To closely examine part of your program, you can enable tracing on only the files you want to step through in the debugger. For example, you usually don't want to step through classes that are in the J2SE library because you're not going to troubleshoot on them; you usually only want to trace into your own classes.

19.6.14 How to Step Through Behavior as Guided by Tracing Lists

If you exclude a class or package, and you instruct the debugger to step into that class, the debugger runs straight through that code without pausing. The debugger pauses at the next line of code in a class which has not been excluded. The tracing include and exclude lists are used for all step commands including Step Into, Step Over, Step Out, and so on. Using these lists does not prevent you from setting a breakpoint in a class which has been excluded. If the debugger stops at such a breakpoint, the step commands will be disabled.

To enable tracing for a class, you can adjust the tracing include or exclude list by adding or removing a class or package:

1. Right-click a project in the navigator and choose **Project Properties** from the context menu.
2. Select the **Run/Debug/Profile** node.
3. Choose a run configuration and click **Edit**.
4. In the Edit Run Configuration dialog select the Debugger node.
5. In the **Tracing Classes and Packages to Include** and **Tracing Classes and Packages to Exclude** parameters, enter the name of the packages or classes you want to include or exclude in the appropriate field, separated by a semicolon (;).

   Alternately, click **Edit** to open the Tracing Classes and Packages to Include/Exclude dialog, then click **Add** or **Remove**. If you click **Add**, the Class and Package Browser dialog appears. If you click **Remove**, the selected class or package is removed from the appropriate tracing List. Navigate to the class or

package you want to add and click **OK**. The class or package is added to the appropriate tracing list.

By leaving the include lists blank, you are actually specifying that you would like to enable tracing in all packages except for those specifically listed in the exclude list. For example:

```
include:
exclude:java;javax
```

19.6.15 How to Locate the Execution Point for a Thread

When you're debugging, the line of code that is the current execution point for the current thread is highlighted and the execution point icon appears in the left margin of the source editor.

The execution point marks the next line of source code to be executed by the debugger.

To find the current execution point:

1. Choose **Run** > **Find Execution Point** from the main menu.
2. Right-click a thread in the Threads window and choose **Go To Source of Thread**.

   The debugger displays the block of code containing the execution point in the source editor.

19.6.16 How to Run to the Cursor Location

When stepping through your application code in the debugger, you may want to run to a particular location without having to single step or set a breakpoint.

To run to a specific program location:

1. In a source editor, position your text cursor on the line of code where you want the debugger to stop.
2. Run to the cursor location in any of the following ways:
   - In the source editor, right-click and choose **Run to Cursor**.
   - Choose the **Run** > **Run to Cursor** option from the main menu.
   - Press **F4**.

Any of the following conditions may result:

- When you run to the cursor, your program executes without stopping, until the execution reaches the location marked by the text cursor in the source editor.
- If your program never actually executes the line of code where the text cursor is, the **Run to Cursor** command will cause your program to run until it encounters a breakpoint or when your program finishes.

19.6.17 How to Pause and Resume the Debugger

You can pause your program when the program is running in the debugger. You can then use the debugger to examine the state of your program with respect to this program location. When you have finished examining that part of the program, you can then continue running the program.

When you are using the debugger, your program can be in one of two possible states: running, or paused by the debugger. When your program is waiting for user input, it

is still considered to be running. When your program is in the running mode, **Pause** is available. When your program is paused by the debugger, the available debugger buttons include **Resume**, **Step Over**, and **Step Into**.

You can pause the debugger in the following ways:

- Choose **Run** > **Pause** from the main menu.
- Click the **Pause** icon from the debugger toolbar.

Figure 19–5 Pause Icon

Your program may be paused at a location for which there is no source available. In this case, the Source Not Found dialog is displayed prompting you for the source file location or whether to generate stub files.

Also, your program may be paused at a location where tracing is disabled because the class is on the tracing exclude list. For example, your program may be paused in the `java.lang.Object.wait` method.

While the debugger is paused, you can force garbage collection to occur. The results of the garbage collection are immediately reflected in the Classes and the Heap window. This enables you to find memory leaks in your application.

To resume the debugger when it is paused, choose **Run** > **Resume**.

19.6.18 How to Terminate a Debugging Session

Sometimes while debugging, you will find it necessary to restart the program from the beginning. For example, you might need to restart the program if you step past the location of a bug.

To terminate the current debugging session:

- Choose the **Run** > **Terminate - <*program name*>** menu option, or
- Click **Terminate** in the debugger toolbar.

Terminating a debugging session closes all debugger windows. However, this action does not delete any breakpoints or watches that you have set, which makes it easy to restart a debugging session.

19.6.19 How to View the Debugger Log

The Debugger log displays information about the debugging process. You can view the Debugger log at any time while the debuggee process is still active.

To view the Debugger log while the process is still active, use one of the two following ways:

- In the **View** menu, select **Debugger** and then select **Log**.
- In the Run Manager, right-click the process and select **View Log** in the context menu.

19.6.20 How to Debug an Applet

JDeveloper allows you to control how your Applet program is debugged.

To debug an applet:

1. In the navigator, select the HTML file that contains the `<APPLET>` tag.
2. Click **Debug** in the toolbar.

   The applet starts. The debugger will stop at breakpoints you have set in your applet source code.

19.6.21 How to Debug a Javascript Program

JDeveloper allows you to control how your Javascript program is debugged, including configuring your browser for remote debugging.

To configure Javascript debugger options in JDeveloper:

1. Choose **Application** > **Default Project Properties** (to set preferences that apply to all projects) or choose **Application** > **Project Properties** (to set preferences that apply only to the current project).
2. Select the **Run/Debug/Profile** node.
3. Select a run configuration. For more information, see Section 19.3, "How to Configure a Project for Running."
4. Click **Edit**.
5. Select the Javascript node under Launch Settings.
6. Select your browser.

   Choose **FireFox/Mozilla** and you'll get more options to control your Javascript debugging. If JDeveloper is not already configured for Firefox as your debugging browser, follow these steps:

   - Enter the path of Firefox browser executable file (`firefox.exe`) in **Browser Command Line**, or click **Browse** and select the executable file.
   - Click the **Install debuggee extension in browser** button to install the debugging extension in Firefox. Firefox opens with a page that provides a link to install the extension. Click the **Install OracleJSDebugAgent for Windows** link and install the Oracle Javascript Debug Agent Extension. Restart Firefox to complete the installation.
7. Click **OK** to close the Edit Run Configuration dialog, and then close the Project Properties dialog.

To debug a Javascript program:

1. In the Application Navigator, select the HTML/JSP/JS file that contains the Javascript code.
2. Right-click and choose **Debug** from the context menu.
3. In the How Should the Target be Started dialog, if you are debugging a JS file or an HTML file without server programming, select **In the Browser without Starting Server Instance**. If you are debugging a JSP file or an HTML file with server programming, choose **In the Server Instance**. Click **OK**.

   The program starts in Firefox browser and, in JDeveloper, the debugger stops at the first breakpoint you have set in the source code.

19.7 Using the Debugger Windows

JDeveloper provides a number of special-purpose debugging windows to help you analyze your code.

19.7.1 Using the Breakpoints Window

Information about set breakpoints can be viewed in the Breakpoints window. For more information about this window including its context menu options, press F1 in the Breakpoints window.

To open the Breakpoints window to displays a list of set breakpoints:

- Choose **View** > **Debugger** > **Breakpoints** from the main menu. The Breakpoints window appears.

To change which columns are displayed in the Breakpoints window:

- Right-click in the Breakpoints window and choose **Preferences** from the context menu. Under Columns, select the columns you want to be displayed in the Breakpoints window.
- Or, in the Breakpoints window, right-click on the columns heading and select the desired column names.

19.7.2 How to Use the Smart Data Window

Unlike the Data window which displays all arguments, local variables, and static fields for the current method, the Smart Data window displays only the data that appears to be relevant to the source code that you are stepping through. Specifically, the debugger analyzes the source code near the execution point and finds the variables, fields, and expressions, that are used in the lines of code that you are stepping through.

For more information, see Section 19.6.15, "How to Locate the Execution Point for a Thread."

The Smart Data window also displays the current return value of a non-void method when you set a breakpoint in the method and issue a **Step to End of Method** command or **Step Out** command. The return value is not displayed for **Step Over** or **Step Into** commands.

By default, the debugger analyzes only one line of code for each location and analyzes up to two locations. You can adjust these settings in the **Tools** > **Preferences - Debugger - Smart Data** page which you can also access by right-clicking in the Smart Data window and choosing **Preferences** from the context menu.

To open the Smart Data window:

1. Set a breakpoint in the Source Editor and start a debugging session.
2. Click **Debug** from the toolbar.
3. When the debugger hits a breakpoint, select **View** > **Debugger** > **Smart Data**.

To change which columns are displayed in the Smart Data window:

- Right-click in the Smart Data window and choose Preferences from the context menu. Under Columns, select the columns you want to be displayed in the Smart Data window.
- Alternatively, in the Smart Data window, right-click on the columns heading and select the desired column names.

If the project builds successfully, the debugger starts.

19.7.3 How to Use the Data Window

You use the Data window to display information about variables in your program. The Data window displays the arguments, local variables, and static fields for the current context, which is controlled by the selection in the Stack window. If you move to a new context, the Data window is updated to show the data for the new context. If the current class was compiled without debug information, you will not be able to see the local variables. The debugger analyzes the local variable memory locations in the stack frame to show you as much information as possible.

> **Note:** By default, the Data window displays local variable information while debugging a program. To disable local variable information in Data window, clear the **Full Debug Info** checkbox in the Compiler page of the Project Properties dialog. The **Full Debug Info** checkbox is selected by default.

The Data window also displays the current return value of a non-void method when you set a breakpoint in the method and issue a **Step to End of Method** command or **Step Out** command. The return value is not displayed for **Step Over** or **Step Into** commands.

To open the Data window:

1. Open source files in the Source Editor and set breakpoints.
2. In the toolbar, click **Debug**.
3. When the debugger pauses at a breakpoint, select **View** > **Debugger** > **Data** from the main menu.

To view array elements in Data window:

1. Start debugging the project and open Data window.
2. Select the array in the Data window and expand to view its elements. If the array contains more than 20 elements, the Data window displays first 20 elements.
   - To view the next 20 entries, click **Next**.
   - To view the previous 20 entries, click **Previous**.
   - To view the first 20 entries, click **First**.
   - To view the last 20 entries, click **Last**.
   - To change the default display size of 20, select the array, right-click and select **Adjust Range** from the context menu, and enter the new value in the **New Count** field. Click **OK** when you are done.

To change which columns are displayed in the Data window:

- Right-click in the Data window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Data window.
- Or, in the Data window, right-click on the columns heading and select the desired column names.

If the project builds successfully, the debugger starts.

19.7.4 How to Use the Watches Window

A *watch* enables you to monitor the changing values of variables or expressions as your program runs. After you enter a watch expression, the Watches window displays the current value of the expression. As your program runs, the value of the watch changes as your program updates the values of the variables in the watch expression.

A watch evaluates an expression according to the current context which is controlled by the selection in the Stack window. If you move to a new context, the expression is re-evaluated for the new context. If the execution point moves to a location where any of the variables in the watch expression are undefined, the entire watch expression becomes undefined. If the execution point returns to a location where the watch expression can be evaluated, the Watches window again displays the value of the watch expression.

To open the Watches window:

1. Open source files in the Source Editor and set breakpoints.
2. Click **Debug** from the toolbar.
3. When the debugger pauses at a breakpoint, select **View** > **Debugger** > **Watches** from the main menu.

To change which columns are displayed in the Watches window:

1. Right-click in the Watches window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Watches window.
2. Alternatively, in the Watches window, right-click on the columns heading and select the desired column names

To add a watch:

- Right-click an item in the Data window and choose **Watch** from the context menu.
- Drag and drop variables, fields, and objects from the Data window to the Watches window.
- Select text in the source editor, right-click, and choose **Watch** from the context menu.

To watch a static field:

Enter the full name of the class followed by a period (.) and the name of the field. For example:

```
java.io.File.separator
```

To watch the current exception while stopped at an exception breakpoint, enter:

```
_throw
```

19.7.5 How to Use the Inspector Window

The Inspector window allows you to single out a selected variable, field or object, and display the same information that is available in the Watch or Data windows. For more information about this window, including its context menu options, press F1 in the Inspector window.

The Inspector window is slightly different from the other windows in that it floats by default, and you can have multiple instances of Inspector windows. Each Inspector window contains one data item. You can drag one Inspector window into another and dock them together.

To open the Inspector Window:

1. Set at least one breakpoint in the Source Editor.
2. Click **Debug** from the toolbar.
3. When the debugger reaches a breakpoint, select a variable in the Source Editor, right-click, and choose **Inspect**.

   The floating Inspect window appears and contains the variable you selected. If you want to inspect something else, enter a new expression or variable in the text field, or select a previous one from the dropdown list.

   If no variable or expression is selected, the Inspect dialog appears pre-populated with the text under the cursor in the editor as the expression to inspect. Click **OK** to open the Inspector window.

The Inspector window will appear floating in the center of your screen, but you can dock the Inspector window with other windows. To prevent docking, press the **Ctrl** key while moving the window. An inspector evaluates an expression according to the current context of the Stack window. For more information, see Section 19.7.7, "How to Use the Stack Window."

If you move to a new context, the expression is reevaluated for the new context. If the execution point moves to a new location where any of the variables in the expression are undefined, the entire expression becomes undefined. If the execution point returns to a location where the expression can be evaluated, the inspector again displays the value of that expression.

To change which columns are displayed in the Inspector window:

- Right-click in the Inspector window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Inspector window.
- Or, in the Inspector window, right-click on the columns heading and select the desired column names.

19.7.6 How to Use the Heap Window

The Heap window displays information about the heap in the program you are debugging and helps you to detect memory leaks in your program. You can view all instances of a class as well as why an object has not been garbage collected.

Two types of folders display in the Heap window:

- **Class Folder**

  Displays the name of the class and how many instances of the class exist in memory, and when expanded lists the specific instances and their addresses in the heap.

- **Reference Path Folder**

  Contains all the "root" references which point, either directly or indirectly, to a specific object. Root references are static fields, stack variables, pinned objects. The garbage collector will not discard an object if there are any root references.

Expanding a root reference will show you the reference path from the root reference to the specified object.

To open and use the Heap window:

1. Open source files in the Source Editor and set breakpoints.
2. In the toolbar, click **Debug**.
3. When the debugger hits the breakpoint, select **View** > **Debugger** > **Heap** from the main menu.
4. Right-click in the Heap window and choose **Add New Type** from the context menu. Alternatively, drag a class node from the Classes window into the Heap window. Or, right click on a class node in the Classes window and choose **Display in Heap** from the context menu. Information about the classes appears in the Heap window.

To change which columns are displayed in the Heap window:

- Right-click in the Heap window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Heap window.
- Alternatively, in the Heap window, right-click on the columns heading and select the desired column names from the context menu.

19.7.7 How to Use the Stack Window

The Stack window displays the call stack for the current thread. When you highlight a line in the Stack window, the Data window, Watches window, and all Inspector windows are updated to show data for the highlighted method.

To open the Stack window:

1. Open source files in the Source Editor and set breakpoints.
2. Click **Debug** from the toolbar.
3. When the debugger pauses at a breakpoint, from the main menu, select **View** > **Debugger** > **Stack**.

To view the stack of a thread:

1. Start debugging the project and open Stack window.
2. Select the thread from the dropdown list, above the columns. The Stack window immediately reflects the stack of the selected thread.

To change which columns are displayed in the Stack window:

1. Right-click in the Stack window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Stack window.
2. Alternatively, in the Stack window, right-click on the columns heading and select the desired column names.

19.7.8 How to Use the Classes Window

The Classes window displays which classes have been loaded and may also include useful information, such as the number of instances of a class. In conjunction with the Classes window, the debugger also includes a garbage collection tool when you want

to force a run of the Java garbage collector. When you run the garbage collector, the impact is shown immediately in the Classes window. You can only force a run of the garbage collector when you are using a virtual machine that allows the debugger to do so.

To open the Classes window:

1. Set a breakpoint in the Source Editor and start a debugging session.
2. When the debugger hits a breakpoint, select **View** > **Debugger** > **Classes**.

   The Classes window displays all the classes that are currently loaded, how many instances of that class are being used, and how much memory that number of instances requires.

To choose information that is displayed in the Classes window:

- Right-click an item in the Classes window and choose **Preferences** from the context menu. Under Columns, select the columns you want to be displayed in the Classes window.
- Alternatively, in the Classes window, right-click on the columns heading and select the desired column names.

To change the ascending or descending view order:

- Click at the top of each column to change the sort order. You can sort by:
  - Name
  - Count
  - Memory
  - File

If the Show Packages check box is selected, by default the classes are displayed in a tree structure, where each branch represents a package. Also, the icon and entry next to each class or package indicates whether the class is included or excluded from tracing. The special icon shown in Figure 19–6 for a class without line number tables is used for classes to indicate that tracing is not possible because the class has been stripped or obfuscated.

Figure 19–6 Icon Indicating Tracing Is Not Possible

In the Classes window, choose Preferences from the context menu to select which columns to view from the following available options:

- Count
- Memory
- File

19.7.9 How to Use the Monitors Window

Java supports multithreading at the language level through the use of *synchronization*. Synchronization is the coordinating of activities and data access among multiple threads. The mechanism that Java uses to support synchronization is the *monitor*. The Monitors window displays status and control information for active monitors.

To open the Monitors window:

1. Open source files in the Source Editor and set breakpoints.
2. In the toolbar, click the **Debug** icon.
3. When the debugger stops at the breakpoint, select **View** > **Debugger** > **Monitors**.

To choose information that is displayed in the Monitors window:

- Right-click an item in the Monitors window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Classes window.
- Alternatively, in the Monitors window, right-click on the columns heading and select the desired column names.

19.7.10 How to Use the Threads Window

The Threads window displays the names and status of all the threads and thread groups in your program.

To open the Threads window:

1. Open source files in the Source Editor and set breakpoints.
2. Click **Debug** from the toolbar.
3. When the debugger stops at a breakpoint, choose **View** > **Debugger** > **Threads** from the main menu.

   The step commands including **Step Over**, **Step Into**, and **Set Next Statement** apply to the current thread. To select a different thread, right-click a thread and choose **Select Thread** from the context menu.

   When you highlight a thread in the Threads window, the Stack window is automatically updated to show the stack for the highlighted thread.

To change which columns are displayed in the Threads window:

- Right-click in the Threads window and choose **Preferences** from the context menu. Under **Columns**, select the columns you want to be displayed in the Threads window.
- Alternatively, in the Threads window, right-click on the columns heading and select the desired column names.

19.7.11 How to Set Preferences for the Debugger Windows

You can choose to customize various debugger window settings including the column resize mode and other options you want to display.

> **Tip:** If the debugger has trouble connecting to the debuggee (the program you are debugging), try increasing the connection retry setting.

To set any of the Debugger window preferences:

1. Choose **Tools** > the **Preferences - Debugger** page.

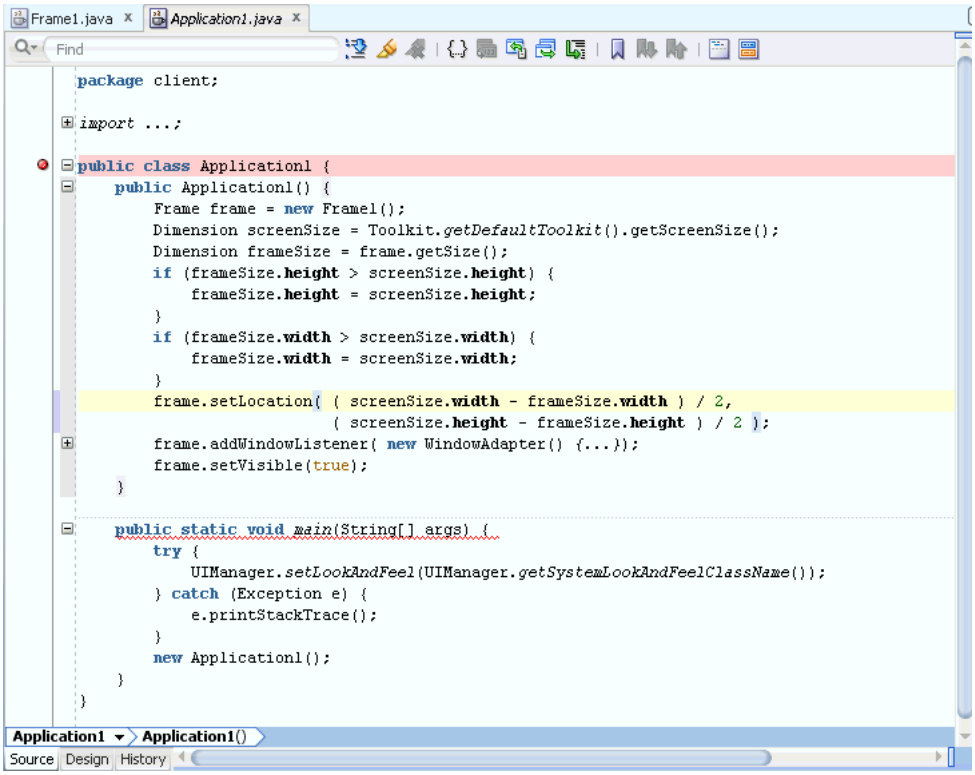
   The debugging panel appears with customizable fields.
2. Make your selections from the fields and options provided.

3. To set any options for a specific debugger window, expand the Debugger node and click the appropriate window node. For example, if you want to change the columns displayed in the Smart Data window, click **Smart Data**.
4. Edit any of the available options as desired.
5. Click **OK** when you are done.

19.8 Managing Breakpoints

A breakpoint is a trigger in a program that, when reached, pauses program execution allowing you to examine the values of some or all of the program variables. By setting breakpoints in potential problem areas of your source code, you can run your program until its execution reaches a location you want to debug. When your program execution encounters a breakpoint, the program pauses, and the debugger displays the line containing the breakpoint in the source editor. You can then use the debugger to view the state of your program. Breakpoints are flexible in that they can be set before you begin a program run or at any time while you are debugging. Figure 19–7 displays an example breakpoint in a Java Application source file.

Figure 19–7 Breakpoint in Source Editor



Breakpoints set on comment lines, blank lines, declarations, and other non-executable lines of code are invalid and will not be verified by the debugger.

The JDeveloper debugger supports a number of different types of breakpoints:

- Source breakpoints
- Exception breakpoints
- Method breakpoints

- Class breakpoints
- File breakpoints
- Deadlock breakpoints

Deadlock breakpoints are useful in situations when you find it difficult to locate the source of the deadlock. When a deadlock breakpoint is encountered, the debugger halts. The deadlock breakpoint is automatically enabled when you start debugging.

Information about set breakpoints can be viewed in the Breakpoints window.

19.8.1 About Verified and Unverified Breakpoints

While debugging, you can place a breakpoint to the left of any line of code in the source editor. However, for a breakpoint to be valid, it must be set on an executable line of code. Before a method is first executed, the debugger verifies all valid breakpoints in the method. Breakpoints set on comment lines, blank lines, declarations, and other non-executable lines of code are invalid and will not be verified by the debugger.

When a breakpoint has been verified as valid, the icon displayed in the source editor margin and in the Breakpoints window changes to the icon shown in Figure 19–8.

Figure 19–8 Verified Breakpoint Icon

19.8.2 Understanding Deadlocks

A deadlock occurs when one or more threads in your program are blocked from gaining access to a resource or waiting on a condition that cannot be satisfied. A common deadlock in Java is a monitor block cycle deadlock.

A monitor block cycle deadlock occurs when two or more threads are unable to proceed because each is waiting to enter synchronized code that one of the others has already entered.

Example 19–2 shows a typical Java synchronization deadlock.

Example 19–2 Java Synchronization Deadlock

```
synchronized (a)
 {
 ...
 synchronized (b)
 {
 ...
  }
  ...
 }
```

At the same time, thread 2 is executing the following code:

```
synchronized (b)
{
...
  synchronized (b)
  {
  ...
```

```
    }
    ...
}
```

A deadlock will occur if thread 1 enters the `synchronized (a)` as thread 2 enters the `synchronized (b)`. Thread 1 will be blocked from entering synchronized (b) until thread 2 finishes the `synchronized (b)` and thread 2 will be blocked from entering synchronized (a) until thread 1 finishes the synchronized (a). A deadlock is also called a "deadly embrace." This example is for two threads but the same situation could occur for 3, 4, 5, and so on threads. The deadlock breakpoint can detect this type of deadlock.

Another kind of deadlock is where one thread calls the wait method on a particular object and no other threads call the notify method on that object. The most common cause of this kind of deadlock is timing. The notifying thread may have called notify before the waiting thread called wait. The important thing to know about calling wait is that even if notify was already called many times before, the wait method waits until notify is called again. Also, notify doesn't return any kind of error if there was no thread waiting. The deadlock breakpoint cannot detect this type of deadlock.

If you think your program is hanging, click **Pause** to pause your program in the debugger, and open the Monitors window. Perhaps you can see that one thread is waiting, investigate the code. If you can see that another thread probably called notify before the first thread called wait, there is a deadlock. This kind of deadlock is very hard to detect. You must know your code well in order to figure out which other thread should have called notify.

19.8.3 Understanding the Deadlock Breakpoint

The JDeveloper debugger sets a persistent deadlock breakpoint when it starts running. A deadlock breakpoint is useful in situations when you find it difficult to locate the source of the deadlock. When the debugger encounters a deadlock breakpoint, the debugger halts. It can detect a monitor block cycle deadlock as described above. The Monitors window can be useful when working with deadlocks.

The deadlock breakpoint has the following characteristics:

- It is a persistent breakpoint that is created automatically when you use JDeveloper.
- It cannot be deleted, but it can be disabled.
- It pauses the debugger if a monitor block cycle deadlock is detected. A monitor block cycle deadlock occurs when two or more threads are unable to proceed because each is waiting to enter synchronized code that one of the others has already entered.

The JDeveloper debugger automatically creates a persistent deadlock breakpoint; this breakpoint will occur whenever a monitor block cycle is detected. You cannot delete a persistent breakpoint. You cannot create a new deadlock breakpoint, but you can edit the existing persistent deadlock breakpoint.

Not all Java Virtual Machines support deadlock detection; for example, the HotSpot VM does not support deadlock detection.

19.8.4 Understanding Grouped Breakpoints

Grouped breakpoints let you enable a set of breakpoints. When the debugger reaches a certain point in your code, you can instruct the debugger to enable a breakpoint or a group of breakpoints that was previously disabled.

For example, even though your code might be catching a `NullPointerException`, it may not be behaving correctly. In some cases, `NullPointerExceptions` occur more frequently than expected which causes the debugger to stop repeatedly for `NullPointerExceptions`, including those that are of no consequence to your code. This situation can be resolved by creating a breakpoint group, adding this breakpoint to the group, and disabling the breakpoint group so that the debugger does not stop at this breakpoint when debugging.

Next, you can create a source breakpoint in some code that you know is executed just before the problematic `NullPointerException` is thrown. You can set the actions for this source breakpoint so that when the source breakpoint occurs, it will automatically enable the breakpoint group which contains the exception breakpoint.

19.8.5 How to Edit a Breakpoint

JDeveloper allows you to edit the options of a breakpoint after you have added it in the source code.

To view and modify the options of a breakpoint:

1. If the Breakpoints window is not open, select **View** > **Debugger** > **Breakpoints** from the main menu.
2. In the Breakpoints window, select a breakpoint.
3. Right-click and choose **Edit**, or click the **Edit** icon on the Breakpoint toolbar.

   The Edit Breakpoint dialog appears with a **Definition** tab, a **Conditions** tab, and an **Actions** tab.
4. Make any necessary changes to the breakpoint options.
5. To accept the changes, click **OK**.

From the Edit Breakpoint dialog, you can:

- Set a breakpoint option.
- Set the threads to which the breakpoint will apply.
- Set a pass count for the breakpoint.
- Put the breakpoint in a breakpoint group.
- Choose what actions the debugger will take when the breakpoint occurs.

You can right-click to edit a breakpoint located in the source editor:

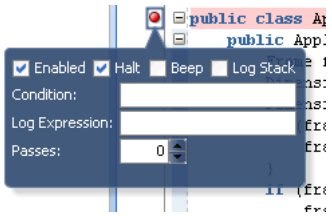
1. Right-click on a breakpoint icon in the gutter of the source editor.
2. Choose **Edit Breakpoint**.

   The Edit Breakpoint dialog displays, where you can specify the definition of the breakpoint.

You can also hover over a breakpoint in the source editor:

- With your mouse cursor, hover over a breakpoint icon in the gutter of an editor window.

Figure 19–9 Edit Breakpoints Dialog



The popup dialog shown in Figure 19–9 displays.

In the dialog, you can edit some of the most important breakpoint attributes, such as enabled/disabled, condition and more.

19.8.6 How to Set Source Breakpoints

A source breakpoint is a breakpoint set in the source code and is the default type of breakpoint.

You can set a source breakpoint in any of the following ways:

- In the source editor, click in the left margin next to a line of executable code.
- In the source editor, right-click in the left margin next to a line of code then choose **Toggle Breakpoint (F5)**.
- Choose **View** > **Debugger** > **Breakpoints** to display the Breakpoints window. Then, right-click anywhere in this window and choose Add Breakpoint from the context menu. From the submenu, select **Source** as the breakpoint type, then complete the package, source file name, and line number information in the dialog. The source filename should not include any directory information, but must include the extension of the file. For example:

  ```
  Application1.java or MyWebApp.jsp
  ```

  You'll probably want to set a least one breakpoint before you start debugging, but it is not necessary. While your program is running in the debugger, you can set a breakpoint. The program pauses when it reaches the breakpoint.

19.8.7 How to Control Breakpoint Behavior

You can control how the debugger behaves when a breakpoint occurs.

To control how the debugger behaves when a breakpoint occurs:

1. In the Breakpoints window toolbar, click **Add Breakpoint**; or select a breakpoint and click **Edit**.
2. Click the **Actions** tab in the New/Edit Breakpoint dialog. The **Actions** tab allows you to change these behaviors:
   - Halt execution (default)
   - Beep
   - Log breakpoint occurrence (enter a tag or an expression)
   - Enable a group of breakpoints
   - Disable a group of breakpoints

19.8.8 How Disable and Delete Breakpoints

When you disable a breakpoint, all the breakpoint settings remain defined, but the breakpoint is not triggered when your program is run; your program will not stop on a disabled breakpoint. Disabling a breakpoint is useful if you have defined a breakpoint that you don't need to use now, but might need to use at a later time.

To disable breakpoints:

- In the source editor, right-click the breakpoint symbol in the left margin and choose **Disable Breakpoint**.
- In the Breakpoints window (**View** > **Debugger** > **Breakpoints**) right-click the breakpoint you want to disable and choose **Disable**.
- To disable a group of breakpoints in the Breakpoints window, select the group that you want to disable, right-click and choose **Disable Group**.

  You can also disable breakpoints from the Breakpoint toolbar. Select the breakpoint or breakpoint group, and click **Disable** on the toolbar.
- To disable all current breakpoints, right-click in the Breakpoints window, and choose **Disable All** from the context menu.

To reenable disabled breakpoints:

- To enable a breakpoint that is disabled, right-click the disabled breakpoint symbol (or entry in the Breakpoints window), and choose **Enable**.
- To enable all breakpoints that have been set, right-click in the Breakpoints window, and choose **Enable All**.
- To enable a group of breakpoints, right-click a breakpoint group in the Breakpoints window, and choose **Enable Group**.

  You can also enable breakpoints from the Breakpoint toolbar. Select the breakpoint or breakpoint group, and click **Enable** on the toolbar.

To delete breakpoints:

When you no longer need to examine the code at a breakpoint location, you can delete the breakpoint. You can delete breakpoints either using the source editor or in the Breakpoints window.

- In the left margin of the source editor, click the breakpoint you want to delete.
- In the left margin of the source editor, right-click the breakpoint you want to delete, and choose **Toggle Breakpoint**.
- In the source editor, place the cursor in the line of code containing the breakpoint, and press **F5**.
- To delete all currently set breakpoints, right-click in the Breakpoints window and select **Delete All**.
- Select the breakpoint in the Breakpoints window and click **Delete Breakpoint** on the toolbar.

Caution: You cannot undelete a breakpoint.

19.8.9 How to Set Instance Breakpoints

Breakpoints typically have effect whenever they are reached. An instance breakpoint is associated with a specific instance of the class that defines the method where the breakpoint appears.

An instance breakpoint is a source breakpoint that has been associated with an instance filter that identifies the selected instances. Instance breakpoints do not persist between runs of the debugger. Instance filters are shown in the Instance Filters column of the Breakpoints window.

To set an instance breakpoint:

1. Set the source breakpoint that you will convert to an instance breakpoint. It must be in a method of the instance's class. For more information, see Section 19.8.6, "How to Set Source Breakpoints.".
2. Set a second breakpoint at some point where the desired instance will be accessible.
3. Define the instance filter:
   - Start or resume the debugger.
   - When the debugger stops at the second breakpoint, find the desired instance in the Data window, Smart Data window, or Watches window.
   - Right-click the instance, choose **Instance Filters**, and choose the source breakpoint that is to become an instance breakpoint.

     Repeat for other instances you wish to track.
4. Resume the debugger.

   The debugger will stop at the instance breakpoint only for the selected instances.

19.8.10 How to Set Exception Breakpoints

Breakpoints are typically attached to a particular line of code; they pause the debugger when a particular line of code is about to be executed. In addition, you can set a breakpoint to be activated when a certain type of exception is thrown. Exception breakpoints are not associated with a particular line of code.

To set an exception breakpoint:

1. In the Breakpoints window, click **Add Breakpoint** on the Breakpoint toolbar. From the submenu, choose **Exception Breakpoint**.

   The Create Exception Breakpoint dialog appears.
2. In the **Definition** tab, enter or choose the name of an exception class.
3. If desired, select or clear the **Break for Caught Exceptions** or **Break for Uncaught Exceptions** checkboxes. Both checkboxes are selected by default.
4. Click **OK**.

The debugger will now pause if an exception of the specified type is thrown.

By default, the debugger automatically creates a persistent exception breakpoint for uncaught throws for `java.lang.Throwable`. This breakpoint will occur whenever an uncaught exception is thrown. You cannot delete a persistent breakpoint, although you can disable it.

19.8.11 How to Make a Breakpoint Conditional

When you make a breakpoint conditional, the debugger pauses when a certain condition is met. When a breakpoint is first set, the debugger pauses the program execution each time the breakpoint is encountered. However, using the Edit Breakpoints dialog, you can customize breakpoints so that they are activated only in certain conditions.

The **Conditions** tab in the Edit Breakpoint dialog is where you enter an expression that is evaluated each time the debugger encounters the breakpoint while executing the program. If the expression evaluates to true, then the breakpoint pauses the program. If the condition evaluates to false, then the debugger does not stop at that breakpoint location.

For example, suppose you want a breakpoint to pause on a line of code only when the variable `mediumCount` is greater than 10.

To set a breakpoint condition:

1. Set a breakpoint on a line of code by clicking to the left of the line in the source editor.
2. Open the Breakpoints window by choosing **View** > **Debugger** > **Breakpoints**.
3. In the Breakpoints window, right-click the breakpoint you just set and choose **Edit**.
4. In the Edit Breakpoint dialog, click **Conditions**.
5. Enter an expression in the **Condition** field, for example, `mediumCount > 1`
6. Click **O**K.

You can enter any valid Java language expression in the Edit Breakpoint dialog, but all symbols in the expression must be accessible from the breakpoint's location, and the expression cannot contain any method calls. For an exception breakpoint, you may want to use the exception object in your condition by `using _throw`.

You can also right-click a breakpoint located in the source editor to set conditions:

1. Right-click on a breakpoint icon in the gutter of the source editor.
2. Choose **Edit Breakpoint**.

   The Edit Breakpoint dialog displays, where you can specify conditions.

You can also hover over a breakpoint in the source editor to set conditions:

- With your mouse cursor, hover over a breakpoint icon in the gutter of an editor window.

  The Edit Breakpoints popup dialog shown in Figure 19–9 displays. You can set conditions in the dialog.

19.8.12 Using Pass Count Breakpoints

The **Pass Count** field specifies the number of times that a breakpoint must be passed for the breakpoint to be activated. Pass counts are useful when you think that a loop is failing on the nth iteration. The debugger pauses the program the nth time that the breakpoint is encountered during the program run. The default value is 1.

If the Pass Count column is shown in the Breakpoints window, you can see the pass count value decrement each time the breakpoint line of code is encountered during the

program execution. If the pass count equals 1 when the breakpoint line is encountered, the breakpoint is activated, and the program pauses at that line.

When pass counts are used together with breakpoint conditions, the breakpoint pauses the program execution the *n*th time that the condition is `true`; the condition must be true for the pass count to be decremented.

19.8.13 How to Examine Breakpoints with the Breakpoints Window

To see the list of breakpoints, choose **View** > **Debugger** > **Breakpoints** from the main menu. Breakpoints that have been verified as valid by the debugger are indicated by the icon shown in Figure 19–8. You can use the Breakpoints window to quickly find the breakpoint location in your source code.

To use the Breakpoints window to locate a breakpoint in the source editor:

1. In the Breakpoints window, select a breakpoint.
2. Right-click and choose **Go to Source** from the context menu.

19.8.14 How to Manage Breakpoint Groups

You can enable or disable several breakpoints with a single action, by creating a breakpoint group and putting breakpoints into it. Once you've created a breakpoint group, you can enable, disable, or remove it like a single breakpoint.

You can also drag and drop a breakpoint into or out of a group in the Breakpoints window.

To create a breakpoint group:

1. In the Breakpoints window, right-click a breakpoint and choose **Edit** from the context menu.

   The Edit Breakpoint dialog appears.
2. In the **Breakpoint Group Name** field, enter a group name for this breakpoint.
3. Click **OK**.

   A new group is created in the Breakpoints window, and is indicated by a folder icon. The breakpoint you just edited is automatically put in the new group.

To move a breakpoint into a breakpoint group:

Either drag-and-drop the breakpoint into the breakpoint group, or follow these steps.

1. In the Breakpoints window, right-click a breakpoint and choose **Edit** from the context menu.

   The Edit Breakpoint dialog appears.
2. From the **Breakpoint Group Name** field, select a breakpoint group from the dropdown list, or enter a new group name.
3. Click **OK**.

   The breakpoint is added into the specified group.

To enable, disable, or remove a breakpoint group, in the Breakpoints window, right-click a breakpoints group, and choose **Enable Group**, **Disable Group**, or **Delete Group** from the context menu.

You can also enable or disable a group from the Breakpoint toolbar. With the group name selected in Breakpoints window, click the Enable or Disable icon on the toolbar. All the breakpoints of the selected group will be enabled or disabled.

19.9 Examining Program State in Debugger Windows

Even though you can view your program by running and stepping through it, you usually need to examine the values of program variables to uncover bugs. For example, it is helpful to know the value of the index variable as you step though a loop, or the values of the parameters passed in a method call. When your program is paused in the debugger, you can examine the values of variables, arguments, fields, and array items.

19.9.1 How to Inspect and Modify Data Elements

You can inspect and change the values of data items using the Data, Smart Data, Inspector, or Watches windows during the course of your debugging sessions.

When you inspect a data item, you evaluate it with different expressions while your debugging session is running. If desired, you can then modify program data values as a way to test hypothetical bug fixes during a program run. If you find that a modification fixes a program error, you can exit the debugging session, fix your program code accordingly, and recompile the program to make the fix permanent.

You can modify program data values during a debugging session as a way to test hypothetical bug fixes during a program run. If you find that a modification fixes a program error, you can exit the debugging session, fix your program code accordingly, and recompile the program to make the fix permanent.

When you modify the value of a variable, the modification is effective for that specific program run only; the changes you make through the Data or Watches windows do not affect your program source code or the compiled program. To make your change permanent, you must modify your program source code in the source editor, then recompile your program.

The new value needs to be type-compatible with the variable you want to assign it to. A good rule of thumb is that if the assignment would cause a compile-time or run-time error, it is not a legal modification value.

To inspect a data item:

1. Open the Data window while the debugger is stopped at a breakpoint.
2. Right-click an item in the Data window and choose **Inspect** from the context menu.

   The floating Inspector window opens displaying the item's name, value, and other related information. The columns which display in this window depend on those column settings that were enabled in the **Tools** > **Preferences - Debugger - Inspector** page. For more information, see Section 19.7.5, "How to Use the Inspector Window."
3. To evaluate the item for an expression, choose **Edit Expression** from the context menu.

   You can also add a watch expression or further inspect the data item.
4. When you are done, close the Inspector window.

To quickly inspect a data item:

If you just want to view a data item's value and do not want to evaluate it for any expression, you can use the Quick Inspect feature.

1. Open the Data window while the debugger is stopped at a breakpoint.
2. Configure the Data window to display the Quick Inspect column. Right-click in the header of the Data window columns and choose **Quick Inspect**. The Quick Inspect is the first column of the window.
3. Select the data item and click the green spherical icon.
4. A child window opens showing the children of the selected item, allowing you to quickly inspect variables deep in an object hierarchy. The quick inspect windows close automatically when you move mouse pointer away from the data item.

JDeveloper also allows you to inspect a data item without adding it in Data window. When the debugger has stopped at a breakpoint in the Source Editor, hover the mouse over a data item to view the its name, value, and type. If the data item is an object or an array, you can inspect children of the selected item deep in the object hierarchy.

To modify the value of a variable in the Data window:

1. Open the Data window while the debugger is stopped at a breakpoint.
2. Right-click an item in the Data window and choose **Modify Value** from the context menu.

   The Modify Value dialog appears with the selected item's name and its current value.
3. Enter a new value for the item.
   - If you are modifying a primitive value, you can enter a new value.
   - If you are modifying a reference pointer (other than a string), you can enter the memory address of an existing object or array.
   - If you are modifying a string, you can enter either a new string value or the memory address of an existing string.
4. Click **OK** to change the value for the item and to close the dialog.

   The new value appears in the Data, Smart Data, Inspector, or Watches windows.

19.9.2 How to Set Expression Watches

A *watch* enables you to monitor the changing values of variables or expressions as your program runs. After you enter a watch expression, the Watch window displays the current value of the expression. As your program runs, the value of the watch changes as your program updates the values of the variables in the watch expression.

A watch evaluates an expression according to the current context which is controlled by the selection in the Stack window. If you move to a new context, the expression is reevaluated for the new context. If the execution point moves to a location where any of the variables in the watch expression are undefined, the entire watch expression becomes undefined. If the execution point returns to a location where the watch expression can be evaluated, the Watches window again displays the value of the watch expression.

To open the Watches window:

- Choose **View** > **Debugger** > **Watches** from the main menu.

To add a watch from the Source Editor:

1. Select the expression you want to watch with your cursor.
2. Right-click and choose **Watch** from the context menu to add the expression to the Watches window.

   A dialog appears with the expression.
3. Edit the expression, if necessary.
4. Click **OK**.

Or, add a watch in the following ways:

- Select a data item in the Data window. Then right-click, and choose **Watch**.
- Right-click in the Watches window and choose **Add Watch**.
- Use the mouse to drag a data item from the Data window and drop it on the Watches window.

To edit a watch:

1. Select the expression in the Watches window, then right-click and choose **Edit Watch**.

   The Edit Watch dialog appears.
2. Enter a new expression or modify the existing one and click **OK**.

To delete a watch:

- Select the expression in the Watches window, press the Delete key or right-click and choose **Remove Watch** from the context menu. You can also delete all the watches by choosing **Remove All Watches** from the context menu.

Caution: You cannot restore a deleted watch.

19.9.3 How to Modify Expressions in the Inspector Window

You can modify an existing expression in the inspector window.

To modify an expression in the Inspector window:

1. In the Inspector window, right-click and choose **Edit Expression** from the context menu.

   The Edit Expression dialog appears.
2. Enter a new expression.
3. Click **OK**.

19.9.4 How to Show and Hide Fields in the Filtered Classes List

While debugging, you can use filters to reduce the number of fields that are displayed when you expand an object in a data-related debugger window. You can perform this task in the Smart Data window, the Data window, the Inspector window, the Watches window, and the left-hand side of the Monitors window through the Object Preferences dialog. Displaying fewer fields narrows your focus when debugging and may make it easier to locate and isolate potential problems in your program.

For example, you can create filters for classes in the data windows so that the debugger displays only the fields of interest to you. This drastically reduces clutter and allows you to find the relevant data more quickly.

To show or hide fields in the filtered classes list:

1. Select an object in a data-related debugger window. Right-click and choose **Object Preferences** from the context menu.

   Choosing **Object Preferences** lets you go directly to the Object Preferences dialog for this specific object from which you can specify filters to control which fields are displayed and which fields are not displayed when you expand an object.

2. In the Object Preferences dialog, you can easily traverse the superclass hierarchy of the selected object, defining or updating the filters for each superclass. Select a class in the **Type Hierarchy** and choose the fields to hide or display in the Value column of the debugger window.

3. Click the arrows to shuttle filters from the **Fields to Show** list to the **Fields to Hide** list.

4. Click **OK** when you are done.

19.10 Debugging Remote Java Programs

In addition to debugging code locally in the JDeveloper IDE, you can also debug code which is located on a remote machine or running in a different VM instance. This means that you can use the debugger to debug code that has already been deployed. The debugger can simultaneously attach to multiple remote VMs, so you can seamlessly debug distributed applications, such as JSPs deployed to a web server accessing EJBs deployed to an application server.

The main difference between remote debugging and local debugging is how you start the debugging session. For local debugging, JDeveloper automatically launches the program you want to debug (called a debuggee process) and then attaches the debugger to that program. For remote debugging, you must manually launch the program you want to debug. Also, if you are debugging a JSP or a servlet, you must manually start a browser to invoke your JSP or servlet.

Once the debuggee is launched and the JDeveloper debugger is attached to it, remote debugging is very similar to local debugging. Remember that you can use remote debugging when the debuggee process is running on the same machine as JDeveloper or when the debuggee process is running on a different machine.

Unlike local debugging, you must choose which protocol to use before you start your remote debugging session. The remote debugging protocols are configured in **Debugger - Remote** page of the Edit Run Configuration dialog.

You can also debug Web pages such as JSPs or servlets using the HTTP Analyzer. For more information, see Chapter 8, "Auditing and Profiling Applications."

Attach to JPDA

Select to attach to the debugger application at a specified address. For more information about the Sun Java Platform Debugger Architecture (JPDA) Connection and Invocation, see
`http://java.sun.com/javase/6/docs/technotes/guides/jpda/conninv.html`

Listen for JPDA

Select to specify that the debugger listen for a debuggee to attach to the debugger. Also, choose this option if you are debugging remote PL/SQL programs.

19.10.1 How to Start a Java Process in Debug Mode

After you've configured a project for remote debugging, you can start your remote debugging session by issuing the appropriate command based on the debugging protocol and the environment.

To start the Java process, enter the following at the command line:

```
java [-client|server] -cp <project_directory>\classes
-agentlib:jdwp,<option1>[=<value1>],<option2>[=<value2>]... <java_main_class>
```

The available options are:

- `server(=n/y)`

  If set to *y*, then the Java process waits for a Debugger to attach. If set to n (default), the process attaches itself to the debugger application at the specified address.

- `address`

  Specifies the port for the connection. Defaults to 4000.

- `timeout`

  Time interval after which the connection attempt times out. Defaults to 2 seconds.

- `suspend =(y/n)`

  If set to *y* (default), the Java process runs after the debugger connects to it. If set to n, the debuggee process starts right away without waiting for the debugger to connect to it.

Command line examples:

- ```
  java -cp <project_directory>\classes
  -agentlib:jdwp=transport=dt_
  socket,server=y,suspend=n,address=4000
  ```

  Listen for a debugger connection on port 4000, but begin execution without waiting for the debugger. Timeout after 2s (default). Implement the Client VM (default).

- ```
  java -server -cp <project_directory>\classes
  -agentlib:jdwp=transport=dt_
  socket,server=n,suspend=y,timeout=3,address=8000
  ```

  Attach to a debugger connection on port 8000. Begin execution only after connecting to the debugger. Timeout after 3s. Implement the Server VM.

For more information about the Sun JPDA Connection and Invocation, see http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html.

19.10.2 How to Remote Debug Using the Javascript Debugger

JDeveloper allows you to remote debug a Javascript program in local instances and server instances.

Before you start remote debugging, install the Firefox plugin for Javascript debugging and configure JDeveloper for Javascript remote debugging.

To remote debug a Javascript program (HTML/JS files):

1. Enable remote debugging in JDeveloper.
   - Choose **Application** > **Project Properties**.
   - In the Project Properties dialog, select the **Run/Debug/Profile** node. From the **Run Configurations** list, select a run configuration and click **Edit**.
   - In the Launch Settings page of the Edit Run Configuration dialog, select **Remote Debugging**.
   - Select Remote node under **Tool Settings** > **Debugger**.
   - In the Remote page, select **Protocol** as *Attach to Mozilla/Firefox*.
   - Optionally, set host machine name, port, and timeout information. By default, JDeveloper uses port 4000 and 2 seconds timeout values.
   - Click **OK** to close the Edit Run Configuration dialog, and then close the Project Properties dialog.
2. Close all open instances of Firefox, if any.
3. In JDeveloper, set breakpoints in the Javascript program.
4. Open command window and start Mozilla Firefox with the following command:

   ```
   firefox AnHtmlFile -oraclejsdebugport=<port> <another browser argument>
   ```

   For example:

   ```
   C:\>firefox file://C:/Shopcart/Servlet/public_html/index.html -oraclejsdebugport=4000
   ```

   Firefox won't open, but its process will start in the background. Open Windows Task Manager and verify that firefox process is running.
5. In JDeveloper, start the remote debugger. In Application Navigator, select the project, right-click and choose **Start Remote Debugger**.
6. The program starts in Firefox browser and, in JDeveloper, the debugger stops at the first breakpoint you have set in your source code. Now, you may continue debugging your Javascript program using available debugger options.

To remote debug a Javascript program in a server instance (JSP/Servlets/HTML files):

1. Start JDeveloper Integrated WebLogic server. From the **Run** menu, choose **Start Server Instance**.
2. Deploy your project to the integrated WebLogic server.
3. Enable remote debugging in JDeveloper.
   - Choose **Application** > **Project Properties**.
   - In the Project Properties dialog, select the **Run/Debug/Profile** node. From the **Run Configurations** list, select a run configuration and click **Edit**.
   - In the Launch Settings page of the Edit Run Configuration dialog, select **Remote Debugging**.
   - Select Remote node under **Tool Settings** > **Debugger**.
   - the Remote page, select **Protocol** as *Attach to Mozilla/Firefox*.

- Optionally, set host machine name, port, and timeout information. By default, JDeveloper uses port 4000 and 2 seconds timeout values.
- Click **OK** to close the Edit Run Configuration dialog, and then close the Project Properties dialog.

4. Close all open instances of Firefox, if any.
5. In JDeveloper, set breakpoints in your program files.
6. Open command window and start Mozilla Firefox with the following command:

   `firefox` *`webaddress`* -oraclejsdebugport=<*`port`*> <*`another browser argument`*>

   For example:

   ```
   C:\>firefox http://130.35.102.18:7101/Shopcart/index.jsp
   -oraclejsdebugport=4000
   ```

   Firefox won't open, but its process will start in the background. Open Windows Task Manager and verify that firefox process is running.
7. In JDeveloper, start the remote debugger. In Application Navigator, select the project, right-click and choose **Start Remote Debugger**.
8. The program starts in Firefox browser and, in JDeveloper, the debugger stops at the first breakpoint you have set in your source code. Now, you may continue debugging your program using available debugger options.

19.10.3 How to Use a Project Configured for Remote Debugging

Any project can be configured to perform remote debugging.

To configure a project for remote debugging:

1. Click **Debug** from the toolbar.

   The appropriate Attach to dialog appears.
2. In the **Host** list box, enter or select the name or IP address of the machine where the remote debuggee has been started.
3. In the **Port** list box, enter or select the port number for the remote debuggee.
4. Click **OK**.

   In the Log window, once the debugger has connected, a successful connection message appears.
5. If you are remote debugging a JSP or servlet, you will want to access your JSP or servlet by launching your browser. If you are remote debugging an EJB, you will want to run an EJB client that will access your EJB.
6. Continue with your debugging session as usual.
7. To detach the debugger from the remote debugging process without terminating the debuggee process, choose the **Run** > **Detach** menu option. This option is appropriate for remote debugging an application server.
8. To terminate the remote debugging process, choose the **Run** > **Terminate** menu option, or select the **Terminate** icon.

19.10.4 How to Configure JPDA Remote Debugging

In the following steps, you will configure JDeveloper for Java Platform Debugger Architecture (JPDA) remote debugging.

To configure your project for remote debugging:

1. Make changes in the JSP section of global-web-application.xml as follows:

```
<init-param>
   <param-name>debug</param-name>
   <param-value>class</param-value>
</init-param>
```

2. Start commands for Integrated WebLogic Server (make sure -server is the first parameter).

```
value="-server -agentlib:jdwp=transport=dt_
socket,server=y,suspend=n,address=4000 -Xms512m
-Xmx750m -XX:PermSize=128m -XX:MaxPermSize=256m
-Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
-Djava.awt.headless=true -Dhttp.webdir.enable=false"/>
```

To configure JDeveloper for remote debugging:

1. Choose **Application** > **Project Properties**, select the **Run/Debug/Profile** node, select a run configuration and click **Edit**.
2. Select the **Remote Debugging and Profiling** check box.
3. On the Debugger - Remote page, verify that **Protocol** is set to *Attach to JPDA*.
4. Close the Preferences dialog.
5. Set breakpoints in your code and from the **Debug** button dropdown list select the desired run configuration. Complete the connection dialog and verify connection to the debuggee.
6. Access JSP previously deployed to server via a browser. The breakpoint should be hit and all work as expected.

20

Implementing Java Swing User Interfaces

This chapter describes how to use the tools and features that JDeveloper provides to help you develop Java Swing interfaces. It explains the fundamental tasks you perform as you work with components and JDeveloper's UI design tools. Included is a description of the UI debugger, which is used to debug user interfaces specifically for AWT and Swing-based client applications and applets.

This chapter includes the following sections:

20.1 About Implementing Java Swing User Interfaces

Using the Java Visual Editor in JDeveloper, you can quickly and easily assemble the elements of a user interface (UI) for a Java application or applet using Swing components. You construct the UI with JavaBeans selected from the component palette, such as buttons, text areas, lists, dialogs, and menus. Then, you set the values of the component properties and attach event-handler code to the component events.

20.2 Understanding the JDeveloper User Interface Design Tools

With JDeveloper tools, you can visually design and program Java classes to produce new compound or complex components.

JDeveloper UI design tools include the following:

- A Java Visual Editor in which you create and arrange panels and other UI components inside a frame or other UI container. You access the Java Visual Editor by right clicking the file in the Navigator and choosing **Open**. Click the **Design** tab to work the visual editor. For more information, see Section 18.3.1, "Editing Code with the Java Visual Editor."
- A Menu Editor in which you create and edit a menu bar, menus, menu items, and popup menu components. You access the Menu Editor when you have the Java Visual Editor open by dropping a menu component into your UI container from the Component Palette. You can also edit an existing menu component by clicking the menu component in the Structure window. For more information, see Section 20.9.2, "Using the Menu Editor."
- A Component Palette containing visual and nonvisual components. You display the Component Palette for components that you select in the Java Visual Editor. To display the Component Palette for an open file, choose **View** > **Component Palette**. For more information, see Section 11.1.4, "How to Use the Component Palette."
- A Structure window that displays a hierarchical view of all the components in your source file, and their relationships. You access the Structure window below the Navigator window or, if docked, by choosing **View** > **Structure**. For more information, see Section 3.11.6, "Structure Window."
- The Property Inspector, used to inspect and set component properties and to attach methods to component events. Changes made in the Inspector are reflected visually in the Java Visual Editor and source code. You display the Property Inspector for a particular file that you open in the Java Visual Editor. To display the Property Inspector for an open file, choose **View** > **Property Inspector**. For more information, see Section 11.1.3, "How to Use the Property Inspector."

If you want to use the UI design tools on a file, it must meet the following requirements:

- It must be a Java file.
- It must be free from syntax errors.
- It must contain a public class (the file name must be the same as the name of the public class).
- It must follow JDeveloper coding conventions:
  - All UI controls are declared as class members or as local variables within `jbInit()`.
  - All UI property settings done in `jbInit()`. This is necessary in order for the JDeveloper Java Visual Editor and Property Inspector to reflect the settings.
  - All property settings set as expressions involve only member UI controls or static values.
- It must have a default constructor.

Any file that meets the above requirements can be visually designed using the Java Visual Editor and the Property Inspector. You can also visually design a non-UI class.

Note: These requirements are satisfied when you create your files with any of the JDeveloper dialogs.

When you first add a component to your design, the JDeveloper Java Visual Editor ensures that your class has a default constructor, a private `jbInit()` method, and that this `jbInit()` method is called correctly from the default constructor. If JDeveloper does not find this code, it will add it. It will also add any imports needed by the component.

When you open the file in the Java Visual Editor, JDeveloper updates the Structure window tree. For example, if your class has a frame and a menu, there will be subtrees for UI and Menu. If you drop any other JavaBeans components into the Structure window, an 'Others' folder appears so you can select and edit these components in the Property Inspector.

20.3 Controlling the Look and Feel of a Swing Application

JDeveloper includes the Swing GUI components. The Swing classes allow you to specify a look and feel for a user interface. You can take advantage of this new Java pluggable look-and-feel to create applications that have the look and feel of a user's native desktop for Windows or Solaris machines. You can also ensure a uniform look and feel in your applications across platforms with the Java Metal look and feel.

There are four look and feel choices in JDeveloper:

- Oracle
- Metal
- CDE/Motif
- Windows

There are also several considerations when setting the look and feel using the UI Manager:

- The line of code for the look and feel statement is inside a **try/catch** block. This placement is necessary for it to compile.
- This code needs to be run before any components are instantiated. That is why it is placed in the <*Application*>`.java static main()` method.
- The class *UIManager* lives in the `com.sun.java.swing` package.

20.3.1 How to Change the Oracle Look and Feel

To change to the Oracle look and feel:

1. Open the Application file in the Code Editor.
2. In the Main method, add the following code:

```
try {
     UIManager.setLookAndFeel(new oracle.bali.ewt.olaf.OracleLookAndFeel());
}
catch (Exception e) {
}
```

It is necessary to have the Oracle look and feel (OLAF) `share.jar` and `jewt4.jar` files from the jlib directory in the classpath.

20.3.2 How to Change the Windows Look and Feel

To change to the Windows look and feel:

1. Open the Application file in the Code Editor.
2. In the Main method, add the following code:

```
try {
     UIManager.setLookAndFeel(new
      com.sun.java.swing.plaf.windows.WindowsLookAndFeel());
}
catch (Exception e) {
}
```

It is necessary to have the Oracle look and feel (OLAF) `share.jar` and `jewt4.jar` files from the jlib directory in the classpath.

20.3.3 How to Change the Metal Look and Feel

The Metal look and feel is default for Swing components. If you want to use the Metal look and feel, you do not have to change your code.

To change to the Metal look and feel:

1. Open the Application file in the Code Editor.
2. In the Main method, add the following code:

```
try {
     UIManager.setLookAndFeel(new
      javax.swing.plaf.metal.MetalLookAndFeel());
}
catch (Exception e) {
}
```

Note: It is necessary to have the Oracle look and feel (OLAF) `share.jar` and `jewt4.jar` files from the jlib directory in the classpath.

20.4 Working with Java Swing and AWT Components

Use Swing and AWT JavaBeans components to assemble the user interface (UI) for a Java application or applet. You construct the UI in the Java Visual Editor by selecting JavaBeans from the Component Palette, such as buttons, text areas, lists, dialogs, and menus. Then, you set the values of the component properties and attach event-handler code to the component events. Tools to visually design and program Java classes to produce new compound or complex component.

20.4.1 Using Swing JavaBeans Components

Swing components are lightweight components that are self-rendering and do not use Windowing resources as do AWT components. In many cases, these components have corresponding AWT components, but the Swing version is enhanced to allow greater flexibility and consistency between platforms.

Note: Swing components rely on underlying functionality in Swing containers. If you intend to use Swing JavaBeans, you must create your program using a JFrame, JPanel or other container that implements the basic Swing functionality.

Table 20–1 contains the Swing JavaBeans components.

Table 20–1 Swing JavaBeans Components

Component	Description
JButton	A simple push button. A JButton can have an embedded icon.
JCheckBox	A square box used to display boolean (true/false) values. When its value is set to true, the box displays a checkmark by default. You have the option of setting your own checkmark graphic.
JComboBox	Similar to the Choice control in AWT, displays a list of values that the user can select at runtime. JComboBox has the property editable which enables the user to type a new value at runtime.
JEditorPane	A specialized JTextComponent that displays text formatted with HTML 3.2 or RTF. It is intended to allow you to create help pages for your application or applet.
JLabel	A text component that enables you to display a text string and optional icon. Additional properties enable you to set the position of the text relative to the icon.
JList	Displays a list of objects. **Tip**: The JList, unlike the AWT List component, doesn't have a scrolling facility built into the component. To make a scrollable list, you need to drop the list component into a JScrollPane container.
JPasswordField	A JTextField that by default displays asterisks (*) in place of the characters entered by the user.
JProgressBar	Displays a progress bar that graphically depicts the percentage of completion for a process.
JRadioButton	This component is specifically designed to behave as a Radio Button. When grouped with other JRadioButtons in a ButtonGroup, only one of the buttons in the group can be selected at one time. The ButtonGroup is a non-visual component.
JScrollBar	A graphic control the user can use to set an integer value. This component can be displayed with either a horizontal or a vertical orientation.
JSeparator	A component that draws a straight line. It is intended to be used as a component in a JMenu, but since it is an actual component, you can use it to draw a line in your UI that separates one set of controls from another. The JSeparator can be displayed in vertical or horizontal orientation.
JSlider	Similar to a JScrollBar, this control allows the user to set an integer value using a graphic control. JSlider allows you to set major and minor tick marks, and to display a border around the control.

Table 20–1 (Cont.) Swing JavaBeans Components

Component	Description
JTable	Displays information in a two-dimensional grid, similar to a spreadsheet application. **Tip**: The JTable, unlike the AWT List component, doesn't have a scrolling facility built into it. To make a scrollable table, you need to drop the list component into a JScrollPane container.
JTextArea	An editable text area that displays a single string in multiple rows, each of which ends in a newline character. **Tip:** In order to make a JTextArea scrollable, it needs to be displayed in a JScrollPane container.
JTextField	An editable text area that displays a single string in a single row.
JTextPane	A full-featured text editor control that enables word wrap, image display, and style definition. **Tip**: In order to make a JTextPane scrollable, it needs to be dropped in a JScrollPane container.
JTree	Displays hierarchical information, such as file directories, in a tree format. **Tip**: The JTree, unlike the AWT List component, doesn't have a scrolling facility built into it. To make a scrollable tree, you need to drop the list component into a JScrollPane container.
JToggleButton	Toggle buttons are similar to JCheckBox controls. When they are pushed (set to true) they remain true until they are set to false by pressing them again. JToggleButtons can be placed in a ButtonGroup so that only one in the group can be true at one time. Toggle buttons are useful in toolbars to display a tool in an active state. The buttons in the Component Palette are examples of toggle buttons.

20.4.2 Using AWT JavaBeans

AWT JavaBeans components use the standard controls of the underlying platform to display the user interface. This architecture makes AWT components simple and straightforward to implement, but limits control over the final display of the UI and reduces flexibility. They also require more system resources to run, which is why they are sometimes referred to as heavyweight components. Swing components are implemented in Java to be self-rendering and therefore make less use of system resources, which is why they are referred to as lightweight components. Swing components can be more complex to implement, but offer greater flexibility and consistency between host platforms.

Table 20–2 contains the AWT JavaBeans Components.

Table 20–2 AWT JavaBeans Components

Component	Description
Button	Displays a simple push button

Table 20–2 (Cont.) AWT JavaBeans Components

Component	Description
Checkbox	True/false, on/off boolean control. As a standalone component, the checkbox appears as a square with an X to indicate when its value is set to true. When set as a member of a checkbox group, the checkbox becomes a circle (Radio Button): a dot in the circle indicates that its value is set to true. Add a checkbox component to a CheckboxGroup by setting its CheckboxGroup property to the name of the CheckboxGroup to which it belongs.
CheckboxGroup	Non-visual component used to integrate the behavior of a set of check box components. Only one item in a checkbox group can be set to true at one time. Checkbox components are added to the CheckboxGroup by setting the CheckboxGroup property to the name of the corresponding CheckboxGroup. The default value for a CheckboxGroup can be set by setting its SelectedCheckbox property to the name of the checkbox component. **Tip:** Depending on the order in which you add the Checkbox components and CheckboxGroup, the CheckboxGroup may be instantiated before the Checkboxes. To ensure that the correct default is set, you can move the SelectedCheckbox and current initialization statements to the bottom of the JbInit method so that all of its constituents are instantiated before the default is set.
Choice	Displays a popup menu. The label of the Choice control is the currently selected item. This is similar to a combobox.
Label	Displays a non-editable text label in the application, though its value may change programmatically.
List	Displays a scrolling list of data items. The user may be able to select one or more items depending on the property settings of the List.
MenuBar	Displays a menubar. Any container can add a MenuBar control as a child, however only Frame has the setMenubar() method which will parent the MenuBar at a specific location and will not interfere with the locations of any other children. Menubars are not available in Applets.
PopupMenu	Displays a popup menu with a list of commands. Popup menus can be attached to panels, and are available for use in Applets. For more information about creating menus, see Section 20.9, "Working with Menus".
Panel	A container which is used to display a group of controls. Each panel can have its own layout to give you control over the positioning of components in relation to one another.
Scrollbar	A slider control that allows the user to set an integer value. The scrollbar can be set to display horizontally or vertically.
ScrollPane	A type of panel that has horizontal and vertical scrollbars available to enable you to display a child component that is larger than the ScrollPane itself.
TextArea	A text component that displays a single string of text in multiple rows, each ending with a newline character. Text can be scrolled up and down, left and right.
TextField	An editable text component that displays a single string of text in a single row.

20.5 Working with Layout Managers

Use JDeveloper's layout managers to control how components are located and sized in the container each time it is displayed. A layout manager automatically arranges the components in a container according to a particular set of rules specific to that layout manager.

A Java program can be deployed on more than one platform. If you use standard UI design techniques of specifying absolute positions and sizes for your UI components, your UI might not look good on all platforms. What looks fine on your development system might be unusable on another platform. To solve this problem, Java provides a system of portable layout managers. Layout managers allow you to specify rules and constraints for the layout of your UI in a way that will be portable.

Layout managers give you the following advantages:

- Correctly positioned components that are independent of fonts, screen resolutions, and platform differences.
- Intelligent component placement for containers that are dynamically resized at runtime.
- Ease of translation with different sized strings. If a string increases in size, the components stay properly aligned.

A Java UI container uses a special object called a *layout manager* to control how components are located and sized in the container each time it is displayed. A layout manager automatically arranges the components in a container according to a particular set of rules specific to that layout manager.

The layout manager sets the sizes and locations of the components based on various factors such as:

- the layout manager's layout rules
- the layout manager's property settings, if any
- the layout *constraints* associated with each component (for more information, see Section 20.5.24, "Understanding Layout Constraints").
- certain properties common to all components, such as `preferredSize`, `minimumSize`, `maximumSize`, `alignmentX`, and `alignmentY`
- the size of the container

In Java, certain types of containers use specific layout managers by default.

- All panels (including applets) use *FlowLayout*.
- All windows (including frames and dialog boxes) use *BorderLayout*.

When you create a container in a Java program, you can accept the default layout manager for that container type, or you can override the default by specifying a different type of layout manager.

Normally, when coding your UI manually, you override the default layout manager before adding components to the container. When using the Java Visual Editor, you can change the layout whenever you like. JDeveloper will adjust the code as needed.

The Java Visual Editor in JDeveloper uses a default layout manager for each container, usually null layout. If you want to use a different layout manager than the default, you can do so by explicitly adding a layout manager to the source code for the container, or by selecting a layout from the container's *layout* property list in the Inspector.

> **Note:** If you want to change the properties for a layout manager using the Java Visual Editor, you must explicitly specify a layout for a container so its properties will be accessible in the Property Inspector.

Choose a layout manager based on the overall design you want for the container. Some layouts can be difficult to work with in the Java Visual Editor because they immediately take over placement and resizing of a component as soon as you add it to the container. To alleviate this problem during initial layout prototyping, JDeveloper provides a default layout called *null*, which leaves the components exactly where you place them and at the size you specify. Starting with an null makes prototyping easier in your container. Later, after adding components to the container, you can switch to an appropriate portable layout for your design.

In some designs, you might use nested panels to group components in the main Frame, using various different layouts for the Frame and each of its panels.

> **Note:** If you really want to design a panel without a layout manager, you can set the layout manager in the source code to *null*. However, you should not leave it this way for deployment.

Experiment with different layouts to see their effect on the container's components. If you find the layout manager you've chosen doesn't give you the results you want, try a different one, or try nesting multiple panels with different layouts to get the desired effect.

20.5.1 Understanding Sizing Properties

Layout managers use various pieces of information to determine how to position and size components in their container. Components provide a set of methods that allow layout managers to intelligently lay out components. All of these methods allow a component to communicate its desired sizing to the layout manager.

The methods in Table 20–3 are property getters and represent the following:

Table 20–3 Sizing Properties

Method	Description
`getPreferredSize()`	The size a component would choose to be, that is, the ideal size for the component to look best. Depending on the rules of the particular layout manager, the `preferredSize` may or may not be considered in laying out the container.
`getMinimumSize()`	How small the component can be and still be usable. The `minimumSize` of a component may be limited, for example, by the size of a label. For most controls, `minimumSize` is the same as preferredSize. Layout managers generally respect `minimumSize` more than they do `preferredSize`.
`getMaximumSize()`	The largest, useful size for this component. This is used so that the layout manager won't waste space on a component that can't use it effectively. For example, `BorderLayout` could limit the center component's size to its maximum size, and then either give the space to the edge components, or limit the size of the outer window when resized.
`getAlignmentX()`	How the component would like to be aligned along the x axis, relative to other components.

Table 20–3 (Cont.) Sizing Properties

Method	Description
`getAlignmentY()`	How the component would like to be aligned along the y axis, relative to other components.

To understand how each layout manager uses these pieces of information, see the individual layouts listed in Section 20.5.2, "Understanding Layouts Provided with JDeveloper".

20.5.2 Understanding Layouts Provided with JDeveloper

JDeveloper provides the following standard layout managers from the Java AWT `BorderLayout`, `FlowLayout`, `GridLayout`, `CardLayout`, and `GridBagLayout`. For Swing, `BoxLayout2` and `OverLayout2` have been included.

JDeveloper also provides FormLayout, a grid-based container that places components in a grid of columns and rows, allowing specified components to span multiple columns or rows. FormLayout is provided by JGoodies Forms, an open source framework included with JDeveloper.

Additionally, JDeveloper provides these custom layouts:

- `XYLayout` that keeps components you put in a container at their original size and location (x,y coordinates).
- `PaneLayout`, used by the `SplitPanel` control.
- `VerticalFlowLayout`, which is very similar to `FlowLayout` except that it arranges the components vertically instead of horizontally.

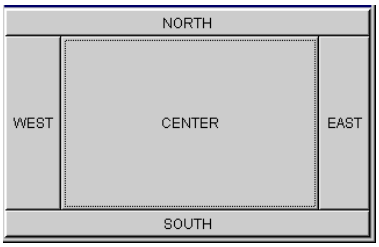
You can create custom layouts of your own, or experiment with other layouts like the ones in the `sun.awt` classes, or third-party layout managers, many of which are public domain on the Web. If you want to use a custom layout in the Java Visual Editor, you may have to provide a Java helper class file to help the Java Visual Editor use the layout.

20.5.3 Using BorderLayout

`BorderLayout` arranges a container's components in areas named North, South, East, West, and Center.

- The components in North and South are given their preferred height and are stretched across the full width of the container.
- The components in East and West are given their preferred width and are stretched vertically to fill the space between the north and south areas.
- A component in the Center expands to fill all remaining space.

Figure 20–1 BorderLayout



The BorderLayout that appears in Figure 20–1 is good for forcing components to one or more edges of a container, and for filling up the center of the container with a component. It is also the layout you want to use to cause a single component to completely fill its container.

You will probably find `BorderLayout` to be the most useful layout manager for the larger containers in your UI. By nesting a panel inside each area of the `BorderLayout`, then populating each of those panels with other panels of various layouts, you can achieve quite complicated UI designs.

Components are positioned in one of five areas within a `BorderLayout`, based on the constraints property. You can set the constraints property for the component in the Inspector to one of the following values: North, South, East, West, or Center.

For example, to put a toolbar across the top of a `BorderLayout` container, you could create a `FlowLayout` panel of buttons and place it in the North area of the container. You do this by selecting the panel and choosing North for its constraints property in the Inspector.

To set the constraints property:

1. Select the component you want to position, either in the Java Visual Editor or the Structure window.
2. Select the `constraints` property in the Inspector, and click its value field.
3. Click the down arrow on the constraints property dropdown list and select the area you want the component to occupy.
4. Press **Enter** or click anywhere else in the Property Inspector to commit the change to code.

If you use the Java Visual Editor to change the layout of an existing container to `BorderLayout`, the components near the edges automatically move to fill the closest edge. A component near the center may be set to Center. If a component moves to an unintended location, you can correct the constraints property in the Inspector, or drag the component to a new position in the Java Visual Editor.

Each of the five areas can contain any number of components (or panel of components), however, unless the topmost component is not opaque, any lower components in the same area will be covered by the topmost one.

The following are some general guidelines for working with multiple components and `BorderLayout`:

- Make sure the container has no more than five components.
- Use `XYLayout` first to move the components to their approximate intended positions, with only one component near each edge.
- Group multiple components in an area into a panel before converting.

Note: BorderLayout ignores the order in which you add components to the container.

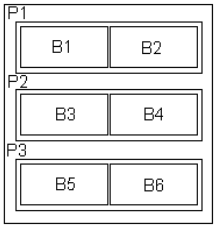
By default, a `BorderLayout` puts no gap between the components it manages. However, you can use the Inspector to specify the horizontal or vertical gap in pixels for a `BorderLayout` associated with a container.

To modify the gap surrounding `BorderLayout` components, select the BorderLayout object in the Structure window (displayed immediately below the container it controls), then modify the pixel value in the Property Inspector for the hgap and vgap properties.

20.5.4 Using BoxLayout2

`BoxLayout2` allows you to arrange a container's components either vertically or horizontally. By nesting components in containers, you can achieve complex layouts. Figure 20–2 shows three panels (P1, P2, P3) arranged vertically. Each panel contains two buttons arranged vertically.

Figure 20–2 BoxLayout2



When you use `BoxLayout2` for a component, you specify whether its major axis is the Y axis (top placement) or X axis (left to right placement). Components are arranged from left to right (or top to bottom), in the same order as they were added to the container.

`BoxLayout2` attempts to arrange components at their preferred widths (for left to right layout) or heights (for top to bottom layout). The components will not wrap if the container is resized.

If all the components are not the same height in a horizontal layout, BoxLayout2 attempts to make all the components as high as the highest component. If that's not possible for a particular component, then `BoxLayout2` aligns that component vertically, according to the component's Y alignment.

If the components are not all the same width in a vertical alignment, `BoxLayout2` attempts to make all components in the column as wide as the widest component; if that fails, it aligns them horizontally according to their X alignments.

20.5.5 Using CardLayout

`CardLayout` places components (usually panels) on top of each other in a stack like a deck of cards. You see only one at a time, and you can flip through the panels by using another control to select which panel comes to the top.

CardLayout is a good layout to use when you have an area that can contain different components at different times. This gives you a way to manage two or more panels that need to share the same display space.

20.5.5.1 How to Create a CardLayout Container

You can see an example of using CardLayout by looking at the wizards in JDeveloper. The **Cancel**, **Next**, and **Back** buttons control which panel is displayed next.

To create a CardLayout container:

1. Create a new Application.
2. Right-click the frame file in the Navigator and choose **Open**.
3. Add a panel to your UI in the Java Visual Editor and set its layout property to CardLayout. For more information, see Section 20.7.9, "How to Create a Panel".
4. Drop a new panel into the CardLayout panel. This new panel will completely fill up the CardLayout panel.

> **Note:** The first component you add to a CardLayout panel will always fill the panel.

5. Set the layout property for this new panel to XYLayout and add the desired components.
6. Click a panel on the Component Palette, then click on the CardLayout panel in the component tree in the Structure window to add it to the stack in CardLayout panel.
7. Set this second panel to XYlayout and add components to it.
8. Repeat steps 6 and 7 for each new panel you want to add to the stack.

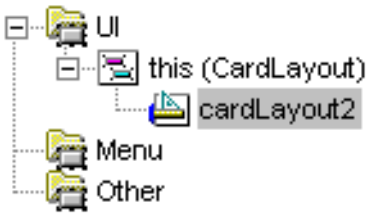
20.5.5.2 How to Specify the Gap Surrounding a CardLayout Container

Using the Structure window, you can specify the amount of horizontal and vertical gap surrounding a stack of components in a CardLayout.

To specify the gap surrounding a cardLayout container:

1. Select the cardLayout object in the Structure window, displayed immediately below the container it controls.

Figure 20–3 cardLayout



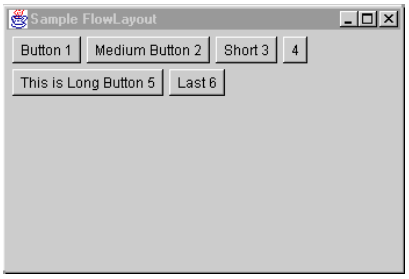
2. Click the hgap (horizontal gap) or vgap (vertical gap) property in the Inspector.
3. Enter the number of pixels you want for the gap.

4. Press **Enter** or click anywhere else in the Inspector to register the changes.

20.5.6 Using FlowLayout

FlowLayout arranges components in rows from left to right, and then top to bottom using each component's preferredSize. FlowLayout lines up as many components as it can in a row, then moves to a new row. Typically, FlowLayout is used to arrange buttons on a panel.

Figure 20–4 FlowLayout



Note: Note: If you want a panel that arranges the components vertically, rather than horizontally, see Section 20.5.21, "Using VerticalFlowLayout".

You can choose how to arrange the components in the rows of a `FlowLayout` container by specifying an alignment justification of left, right, or center. You can also specify the amount of gap (horizontal and vertical spacing) between components and rows. Use the Inspector to change both the alignment and gap properties when you're in the Java Visual Editor.

Alignment

- LEFT- groups the components at the left edge of the container.
- CENTER - centers the components in the container.
- RIGHT groups the components at the right edge of the container. The default alignment in a FlowLayout is CENTER.

  To change the alignment, select the **FlowLayout** object in the Structure window, then specify a value in the Property Inspector for the alignment property as follows:

  - 0=LEFT
  - 1=CENTER
  - 2=RIGHT

Gap

The default gap between components in a `FlowLayout` is 5 pixels.

To change the horizontal or vertical gap, select the `FlowLayout` object in the Structure window, then modify the pixel value of the `hgap` (horizontal gap) or `vgap` (vertical gap) property in the Inspector.

Order of Components

To change the order of the components in a FlowLayout container, drag the component to the new location, or right-click a component and choose **Move to First** or **Move to Last**.

20.5.7 Using FormLayout

`FormLayout` places components in a grid of columns and rows, allowing specified components to span multiple columns or rows. Columns and rows need not have the same width or height.

> **Note:** FormLayout is provided by JGoodies Forms, an open source framework included with JDeveloper. To add FormLayout layout manager, add JGoodies Forms library in the current project's library.

You can change the following column and row properties. Columns and rows are specified by three parts: a mandatory size, an optional default alignment, and an optional resize behavior. To edit these properties, when you're in the Java Visual Editor, drag the mouse and select the entire `FormLayout` grid, then right-click inside the selected grid and choose **Column Properties** or **Row Properties**.

Alignment

The default alignment of components in a column or row. The default alignment in a `FormLayout` is **Fill**.

Size

The width of a column or height of a row. You can specify a constant size, a component size (minimum or preferred), or a bounded size (minimum and maximum). Constant size is specified by a value plus a unit. Component sizes give a column or row the maximum size of its contained components and are measured by the component's minimum or preferred size. Bounded size lets you specify lower and upper bounds for the column and row start size (before resizing). This ensures a minimum or maximum column or row size. Note that the maximum size does not limit the column/row size if the column/row can grow (see resize behavior).

Resize

Controls whether or not resizing is allowed. Columns and rows can grow if the layout container becomes larger than the preferred size. The default is not to allow resizing.

Gap

The default places gaps between components.

> **Note:** Gap size is currently fixed and can not be edited in the Property Inspector.

Order of Components

To change the order of the components in a FormLayout container, drag the component to the new location.

20.5.8 Using GridLayout

`GridLayout` places components in a grid of cells that are in rows and columns. `GridLayout` expands each component to fill the available space within its cell. Each cell is exactly the same size and the grid is uniform. When you resize a `GridLayout` container, `GridLayout` changes the cell size so the cells are as large as possible, given the space available to the container.

Figure 20–5 GridLayout

Use `GridLayout` if you are designing a container where you want the components to be of equal size, for example, a number pad or a toolbar.

You can specify the number of columns and rows in the grid, but only one of the rows or columns can be zero. You must have a value in at least one so the GridLayout manager can calculate the other.

For example, if you specify four columns and zero rows for a grid that has 15 components, `GridLayout` creates four columns of four rows, with the last row containing three components. Or, if you specify three rows and zero columns, `GridLayout` creates three rows with five full columns.

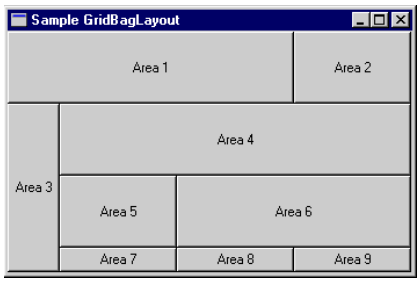
In addition to number of rows and columns, you can specify the number of pixels between the cells by modifying the horizontal gap (`hgap`) and vertical gap (`vgap`) properties. The default horizontal and vertical gap is zero.

To change the property values for a GridLayout container, select the GridLayout object in the Structure window, then edit the values for the `rows`, `cols`, `hgap`, or `vgap` properties in the Property Inspector.

20.5.9 Using GridBagLayout

`GridBagLayout` is an extremely flexible and powerful layout that provides more control than GridLayout in laying out components in a grid. `GridBagLayout` positions components horizontally and vertically on a dynamic rectangular grid. The components do not have to be the same size, and they can fill up more than one cell.

Figure 20–6 GridBagLayout



`GridBagLayout` determines the placement of its components based on each component's constraints and minimum size, plus the container's preferred size.

In the following discussion:

- A component's *cell* refers to the entire set of grid cells the component occupies.
- A component's *display area* refers to all the space of the cell that it occupies which is not taken up by the component's external padding (insets).

While `GridBagLayout` can accommodate a complex grid, it will behave more successfully (and more predictably) if you organize your components into smaller panels, nested inside the `GridBagLayout` container. These nested panels can use other layouts, and can contain additional panels of components if necessary. This method has two advantages:

- It gives you more precise control over the placement and size of individual components because you can use more appropriate layouts for specific areas, such as button bars.
- It uses fewer cells, simplifying the GridBagLayout and making it much easier to control.

On the other hand, GridBagLayout requires more containers, and therefore your program uses more memory, than if you used other layout managers.

20.5.9.1 Understanding GridBagLayout Constraints

`GridBagLayout` uses a `GridBagConstraints` object to specify the layout information for each component in a `GridBagLayout` container. Since there is a one-to-one relationship between each component and `GridBagConstraints` object, you need to customize the `GridBagConstraints` object for each of the container's components.

`GridBagLayout` components have the following constraints:

- anchor
- fill
- gridx, gridy
- gridwidth, gridheight
- ipadx, ipady
- weightx, weighty

`GridBagConstraints` give you control over:

- The position of each component, absolute or relative.

- The size of each component, absolute or relative.
- The number of cells each component spans.
- How each cell's unused display area gets filled.
- The amount of internal and external padding for each component.
- How much weight is assigned to each component to control which components utilize extra available space. This controls the component's behavior when resizing the container.

For a detailed explanation of each of the constraints, including tips for using them and setting them in the Java Visual Editor, see the individual constraint topics.

20.5.9.2 Setting GridBagConstraints Manually in the Source Code

When you use the Java Visual Editor to design a `GridBagLayout` container, JDeveloper always creates a new `GridBagConstraints2` object for each component you add to the container. `GridBagConstraints` is derived from `GridBagConstraints`, and has a constructor that takes all eleven properties of `GridBagConstraints` so the code generated by the Java Visual Editor can be simpler.

For example:

```
bevelPanel1.add(textFieldControl3, new GridBagConstraints2(0, 5, 6, 2, 1.0, 0.0,
GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(0, 24, 0, 0),
150, 0));
bevelPanel1.add(checkboxControl1, new GridBagConstraints2(7, 5, 4, 1, 0.0, 0.0,
GridBagConstraints.CENTER, GridBagConstraints.NONE, new Insets(8, 29, 0, 24), 18,
-11));
```

You can modify the parameters of the `GridBagConstraints2` constructor directly in the source code, or you can use the Constraints property editor to change the values.

When you create a `GridBagLayout` container by coding it manually, you only need to create one `GridBagConstraints` object for each `GridBagLayout` container. `GridBagLayout` uses the `GridBagConstraints` default values for the component you add to the container, or it reuses the most recently modified value. If you want the component you're adding to the container to have a different value for a particular constraint, then you only need to specify the new constraint value for that component. This new value will stay in effect for subsequent components unless, or until, you change it again.

> **Note:** While this method of coding GridBagLayout is the leanest (recycling constraint values from previously added components), it doesn't allow you to edit that container visually in the Java Visual Editor.

20.5.9.3 Modifying Existing GridBagLayout Code to Work in the Java Visual Editor

If you have a `GridBagLayout` container that was previously coded manually by using one `GridBagConstraints` object for the container, you will not be able to edit that container in the Java Visual Editor without making the following modifications to the code:

- You must create a `GridBagConstraints2` object for each component added to the container.

- The `GridBagConstraints2` object must have a large constructor with parameters for each of the eleven constraint values, as shown above.

20.5.9.4 Designing GridBagLayout Visually in the Java Visual Editor

`GridBagLayout` is a complex but useful layout manager. JDeveloper has additional features in the Java Visual Editor that make `GridBagLayout` much easier to design and control, such as a Constraints property editor, a grid, and a pop-up menu on selected components.

There are two approaches you can take to designing `GridBagLayout` in the Java Visual Editor. You can design it from scratch by adding components to a `GridBagLayout` panel, or you can prototype the panel in the Java Visual Editor using another layout first, such as `XYLayout`, then convert it to `GridBagLayout` when you have all the components arranged and sized the way you want them. This method can speed up your design work substantially.

Whichever method you use, it is recommended that you take advantage of using nested panels (for more information, see Section 20.5.33, "Working with Nested Containers and Layouts") to group the components, building them from the inside out. Use these panels to define the major areas of the `GridBagLayout` container to simplify your `GridBagLayout` design, which gives you fewer cells in the grid and fewer components that need `GridBagConstraints`.

20.5.10 Converting to GridBagLayout

When you prototype your layout in another layout first, the conversion to `GridBagLayout` will be much cleaner and easier if you are careful about the alignment of the panels and components as you initially place them, especially left and top alignment. Keep in mind that you are actually designing a grid, so try to place the components inside an imaginary grid, and use nested panels to keep the number of rows and columns as small as possible. Using `XYLayout` for prototyping gives you the advantage of component alignment functions on the components' popup menu (accessed by right-clicking a component in the Java Visual Editor).

As the Java Visual Editor converts the design to `GridBagLayout`, it assigns constraint values for the components based on where the components were before you changed the container to `GridBagLayout`. Often, only minor adjustments are necessary, if any.

Converting to `GridBagLayout` assigns weight constraints to certain types of components (those which you would normally expect to increase in size as the container is enlarged at runtime, such as text areas, fields, group boxes, or lists). If you need to make adjustments to your design after converting to `GridBagLayout`, you'll find the task much easier if you remove all the weight constraints from any components first (set them all to zero).

If even one component has a *weight* constraint value greater than zero, it is hard to predict the sizing behavior in the Java Visual Editor due to the complex interactions between all the components in the container.

You can easily spot a `GridBagLayout` whose components have weights because the components will not be clustered together in the center of the container. Instead, the components fill the container to its edges.

Tip: When you remove all the weights from the components in a GridBagLayout, one of two things will happen:

- If the container is large enough for the grid, the components will all cluster together in the center of the container, with any extra space around the edges of the grid.
- If the container is too small for the components, the grid will expand beyond the edges of the container and the components that are off the edges of the container will be invisible. Enlarge the size of the container until all the components fit. If the `GridBagLayout` container you are designing is a single panel in the center of the main UI frame, enlarge the size of the frame. You can resize this container to the final size after you have finished setting all the components' constraints.

20.5.11 Adding Components to a GridBagLayout Container

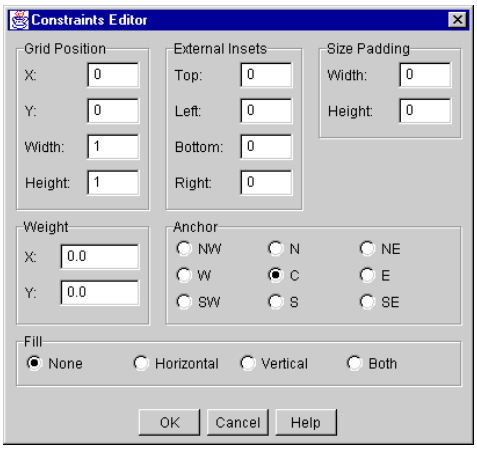
If you want to create your GridBagLayout by starting out with a new GridBagLayout container and adding all the components to it from scratch, there are certain behaviors you should expect.

- Since the default weight constraint for all components is zero, when you add the first component to the container, it will locate to the center of the container at its minimumSize. You will now have a grid with one row and one column.
- The next component you add will be placed in an adjacent cell, depending on where you click. If you click under the first component, it will go on the next row in that column. If you click to the right of the component, it will go on the same row in the next column. All subsequent components are added the same way, increasing the number of cells by one as you add each one.
- Once you have several components, or cells containing components, you can use the mouse to drag the components to new cell locations, or you can change the `gridx` and `gridy` constraints in the Constraints property editor.
- No matter how many components you add, as long as the grid stays smaller than the container, they will all cluster together in the middle of the container. If you need a bigger container, simply enlarge it in the Java Visual Editor.
- If after several rows, your design has been fitting nicely into a certain number of columns, then you suddenly have a row that requires an odd number of components, consider dropping a panel into that row that takes up the entire row, and use a different layout inside that panel to achieve the look you want.

20.5.12 How to Set GridBagConstraints in the Constraints Property Editor

By using the `GridBagLayout Constraints` property editor, some of the `GridBagConstraints` can be specified in the Java Visual Editor without having to edit the source code.

Figure 20–7 Constraints Editor



One big advantage to using the Constraints property editor for setting constraints is the ability to change constraints for multiple components at the same time. For example, if you want all the buttons in your GridBagLayout container to use the same internal padding, you can hold down the Shift key while you select each one, then open the Constraints property editor and edit the constraint.

To use the Constraints property editor:

1. Select the component(s) within the `GridBagLayout` container you want to modify, either in the Structure window or in the Java Visual Editor.
2. Select the constraints property in the Property Inspector and click its value field.
3. Set the desired constraints in the property editor, then press **OK**.

20.5.13 Displaying the Grid

The Java Visual Editor displays an optional grid that lets you see exactly what is happening with each cell and component in the layout.

- To display this grid, right-click on a component in the GridBagLayout container and select **Show Grid**.
- To hide the grid temporarily when **Show Grid** is selected, click on a component that is not in the GridBagLayout container (including the `GridBagLayout` container itself) and the grid will disappear. The grid is only visible when a component inside a `GridBagLayout` container is selected.
- To hide the grid permanently, right-click a component and select **Show Grid** again.

20.5.14 Using the Mouse to Change Constraints

The Java Visual Editor allows you to use the mouse for setting some of the constraints by dragging the whole component or by grabbing various sizing nibs on the component. Directions for setting constraints visually are included in the individual following constraint topics.

20.5.15 Using the GridBagLayout Popup Menu

Right-clicking a `GridBagLayout` component displays a context menu that gives you easy access to some of the properties of the Constraints property editor, and lets you quickly set or remove certain constraints.

Table 20–4 GridBagLayout Popup Menu Commands

Menu Command	Action
Remove Padding	Sets any size padding values (ipadx and ipady) for the selected component to zero.
Constraints...	Displays the Constraints popup editor for the selected `GridBagLayout` component.
Fill Horizontal	Sets (ors) the *fill* constraint value for the component to HORIZONTAL. The component expands to fill the cell horizontally. If the *fill* was VERTICAL, it sets the constraint to BOTH.
Fill Vertical	Sets (ors) the *fill* constraint value for the component to VERTICAL. The component expands to fill the cell vertically. If the *fill* was HORIZONTAL, it sets the constraint to BOTH.
Remove Fill	Changes the *fill* constraint value for the component to NONE.
Weight Horizontal	Sets the *weightx* constraint value for the component to 1.0.
Weight Vertical	Sets the *weighty* constraint value for the component to 1.0
Remove Weights	Sets both *weighty* and *weighty* constraint values for the component to 0.0.

20.5.16 GridBagConstraints

The following section lists each of the GridBagConstraints separately. It defines each one, explaining its valid and default values, and tells you how to set that constraint visually in the Java Visual Editor.

anchor

When the component is smaller than its display area, use the *anchor* constraint to tell the layout manager where to place the component within the area.

The *anchor* constraint only affects the component within its own display area, depending on the fill constraint for the component. For example, if the *fill* constraint value for a component is `GridBagConstraints`.BOTH (fill the display area both horizontally and vertically), the *anchor* constraint has no effect because the component takes up the entire available area. For the anchor constraint to have an effect, set the *fill* constraint value to `GridBagConstraints.NONE`, `GridBagConstraints.HORIZONTAL`, or `GridBagConstraints.VERTICAL`.

Valid values:

GridBagConstraints.CENTER

GridBagConstraints.NORTH

GridBagConstraints.NORTHEAST

GridBagConstraints.EAST

GridBagConstraints.SOUTHEAST

GridBagConstraints.SOUTH

GridBagConstraints.WESTGridBagConstraints.SOUTHWEST

GridBagConstraints.NORTHWEST

Default Value: GridBagConstraints.CENTER

To set the anchor constraint in the Java Visual Editor:

You can use the mouse to set the anchor for a component that is smaller than its cell. Simply click on the component and drag it, dragging the component toward the desired location at the edge of its display area, much like you would dock a movable toolbar. For example, to anchor a button to the upper left corner of the cell, click the mouse in the middle of the button, and drag it until the upper left corner of the button touches the upper left corner of the cell. This sets the `anchor` constraint value to NorthWest.

You can also specify the anchor constraint in the Constraints property editor:

1. Select the component in the Java Visual Editor.
2. In the Property Inspector click the constraints property to display the Constraints editor.
3. Select the desired anchor constraint value in the Anchor area, then press **OK.**

fill

When the component's display area is larger than the component's requested size, use the *fill* constraint to tell the layout manager which parts of the display area should be given to the component. As with the anchor constraint, the fill constraint only affects the component within its own display area. Fill tells the layout manager to expand the component to fill the whole area it has been given.

Valid values:

`GridBagConstraints.NONE`

(Don't change the size of the component.)

`GridBagConstraints.BOTH`

(Resize the component both horizontally and vertically to fill the area completely.)

`GridBagConstraints.HORIZONTAL`

(Only resize the component to fill the area horizontally.)

`GridBagConstraints.VERTICAL`

(Only resize the component to fill the area vertically.)

Default Value: `GridBagConstraints.NONE`

To specify the fill constraint in the Java Visual Editor:

The fastest way to specify the fill constraint for a component is to use the component's context menu in the Java Visual Editor.

1. Right-click the component in the Java Visual Editor to display the context menu.
2. Do one of the following:
   - Select **Fill Horizontal** to set the value to HORIZONTAL.
   - Select **Fill Vertical** to set the value to VERTICAL.
   - Select both **Fill Horizontal** and **Fill Vertical** to set the value to BOTH.

- Select **Remove Fill** to set the value to NONE.

You can also specify the fill constraint in the Constraints editor.

1. In the Property Inspector click the constraints property to display the Constraints editor.
2. Select the desired fill constraint value in the Fill area, then press OK.

gridwidth, gridheight

Use these constraints to specify the number of cells in a row (gridwidth) or column (gridheight) the component uses. This constraint value is stated in cell numbers, not in pixels.

Valid values:

`gridwidth=nn, gridheight=nn`

(Where nn is an integer representing the number of cell columns or rows.)

`GridBagConstraints.RELATIVE (-1)`

Specifies that this component is the next to last one in the row (*gridwidth)* or column (*gridheight*.) A component with a GridBagConstraints.RELATIVE takes all the remaining cells except the last one. For example, in a row of six columns, if the component starts in the third column, a gridwidth of RELATIVE will make it take up columns three, four, and five.

`GridBagConstraints.REMAINDER (0)`

Specifies that this component is the last one in the row (`gridwidth`) or column (`gridheight`).

Default Value: `gridwidth=1, gridheight=1`

To specify gridwidth and gridheight constraints in the Java Visual Editor:

You can specify gridwidth and gridheight constraint values in the Constraints property editor.

1. In the Property Inspector click the constraints property to display the Constraints editor.
2. In the Grid Position area, enter a value for gridwidth in the Width field, or a value for `gridheight` in the **Height** field. Specify the number of cells the component will occupy in the row or column.
   - If you want the value to be RELATIVE, enter a -1.
   - If you want the value to be REMAINDER, enter a 0.

You can use the mouse to change the `gridwidth` or `gridheight` by sizing the component into adjacent empty cells.

gridx, gridy

Use these constraints to specify the grid cell location for the upper left corner of the component. For example, gridx=0 is the first column on the left, and gridy=0 is the first row at the top. Therefore, a component with the constraints gridx=0 and gridy=0 is placed in the first cell of the grid (top left).

`GridBagConstraints.RELATIVE` specifies that the component be placed relative to the previous component as follows:

- When used with `gridx`, it specifies that this component be placed immediately to the right of the last component added.
- When used with `gridy`, it specifies that this component be placed immediately below the last component added.

Valid values:

```
gridx=nn, gridy=nn
```

```
GridBagConstraints.RELATIVE (-1)
```

A maximum value of 512 rows and columns is supported by GridBagLayout; therefore, values you specify for X and Y must be not exceed 512.

Default value:

```
gridx=0, gridy=0
```

To specify the grid cell location in the Java Visual Editor:

You can use the mouse to specify which cell the upper left corner of the component will occupy. Click near the upper left corner of the component and drag it into a new cell. When moving components that take up more than one cell, be sure to click in the upper left cell when you grab the component. Sometimes, due to existing values of other constraints for the component, moving the component to a new cell with the mouse may cause changes in other constraint values, for example, the number of cells that the component occupies might change. To more precisely specify the gridx and gridy constraint values without accidentally changing other constraints, use the Constraints property editor.

1. In the Property Inspector click the `constraints` property to display the Constraints editor.
2. In the Grid Position area, enter the column number for `gridx` value in the X field, or the row number for `gridy` value in the Y field. If you want the value to be RELATIVE, enter a -1.

> **Note:** When you use the mouse to move a component to an occupied cell, the Java Visual Editor ensures that two components never overlap by inserting a new row and column of cells so the components will not be on top of each other. When you relocate the component using the Constraints property editor, the Java Visual Editor does not check to make sure the components don't overlap.

insets

Use insets to specify the minimum amount of external space (padding) in pixels between the component and the edges of its display area. The inset says that there must always be the specified gap between the edge of the component and the corresponding edge of the cell. Therefore, insets work like brakes on the component to keep it away from the edges of the cell. For example, if you increase the width of a component with left and right insets to be wider than its cell, the cell will expand to accommodate the component plus its insets. *`fill`* and *`padding`* constraints never steal any space from insets.

Valid values:

```
insets = new Insets(n,n,n,n)
```

Top, left, bottom, right (where each parameter represents the number of pixels between the display area and one edge of the cell).

Default values:

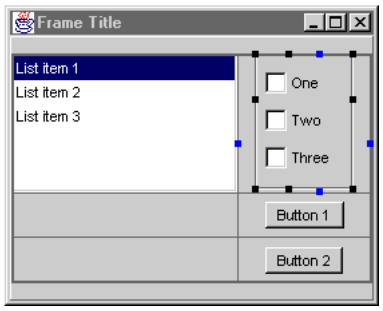
```
insets = new Insets(0,0,0,0)
```

To set inset values in the Java Visual Editor:

The Java Visual Editor displays blue sizing nibs on a selected *GridBagLayout* component to indicate the location and size of its insets. Grab a blue nib (sizing handle) with the mouse and drag it to increase or decrease the size of the inset.

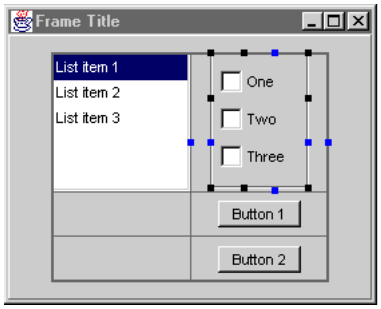
When an inset value is zero, you will only see one blue nib on that side of the cell, as shown in Figure 20–8.

Figure 20–8 GridBagLayout with Inset Value Set to 0



As shown in Figure 20–9, when an inset value is greater than zero, the Java Visual Editor displays a pair of blue nibs for that inset: one on the edge of the cell and one on the edge of the display area. The size of the inset is the distance (number of pixels) between the two nibs. Grab either nib to change the size of the inset.

Figure 20–9 GridBagLayout with Inset Value Set to Greater Than 0



For more precise control over the inset values, use the Constraints property editor to specify the exact number of pixels.

1. In the Property Inspector click the `constraints` property to display the Constraints editor.
2. In the External Insets area, specify the number of pixels for each inset: top, left, bottom, or right.

Note: Although negative inset values are legal, they can cause components to overlap adjacent components, and are not recommended.

ipadx, ipady

These constraints specify the internal padding for a component. Use *ipadx* and *ipady* to specify the amount of space (in pixels) to add to the minimum size of the component for internal padding. For example, the width of the component will be at least its minimum width plus *ipadx* in pixels. The code only adds it once, splitting it evenly between both sides of the component. Similarly, the height of the component will be at least the minimum height plus *ipady* pixels.

1. *ipadx* specifies the number of pixels to add to the minimum width of the component.
2. *ipady* specifies the number of pixels to add to the minimum height of the component.

For example, when added to a component that has a preferred size of 30 pixels wide and 20 pixels high:

- If *ipadx= 4*, the component will be 34 pixels wide
- If *ipadx= 2*, the component will be 22 pixels high.

Valid values:

`ipadx=nn, ipadx=nn`

Default value:

`ipadx=0, ipady=0`

To set the size of internal padding constraints in the Java Visual Editor:

Setting the size of internal padding constraints in the Java Visual Editor. If you drag the sizing nib beyond the edge of the cell into an empty adjacent cell, the component will occupy both cells (the `gridwidth` or `gridheight` values will increase by one cell).

Figure 20–10 Before Dragging the Sizing Nib Beyond the Edge of a Cell

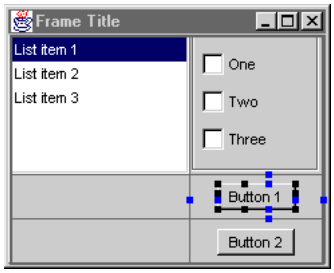
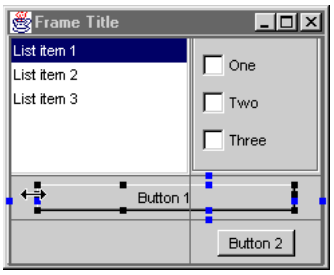


Figure 20–11 After Dragging the Sizing Nib Beyond the Edge of a Cell



For more precise control over the padding values, use the Constraints property editor to specify the exact number of pixels to use for the value.

1. In the Property Inspector click the `constraints` property to display the Constraints editor.
2. Enter the number of pixels for the Width and Height values in the Size Padding area, then press **OK**.

To quickly remove the padding (set it to zero), right-click the component in the Java Visual Editor and choose **Remove Padding**. You can also select multiple components and use the same procedure to remove the padding from all of them at once.

Negative values are valid. They will make the component smaller than its preferred size.

Note: Padding, much like `XYLayout`, may not be accurate on different computer systems or in different languages.

---

weightx, weighty

Use the weight constraints to specify how to distribute a GridBagLayout container's extra space horizontally (*`weightx`*) and vertically (*`weighty`*) when the container is resized. Weights determine what share of the extra space gets allocated to each cell and component when the container is enlarged beyond its default size.

Weight values are of type double and are specified numerically in the range 0.0 to 1.0 inclusive. Zero means the component should not receive any of the extra space, and 1.0 means the component gets a full share of the space.

- The weight of a row is calculated to be the maximum *`weightx`* of all the components in the row.
- The weight of a column is calculated to be the maximum *`weighty`* of all the components in the column.

Valid Values:

`weightx=n.n, weighty=n.n`

Default Value:

`weightx=0.0, weighty=0.0`

To set weightx and weighty constraints in the Java Visual Editor:

To specify the weight constraints for a component in the Java Visual Editor, right-click the component and choose **Weight Horizontal** (*`weightx`*), or **Weight Vertical** (*`weighty`*). This sets the value to 1.0. To remove the weights (set them to zero),

right-click the component and choose **Remove Weights**. You can do this for multiple components: hold down the Shift key when selecting the components, then right-click and choose the appropriate menu item.

If you want to set the *weight* constraints to something other than 0.0 or 1.0, you can set the values in the Constraints editor.

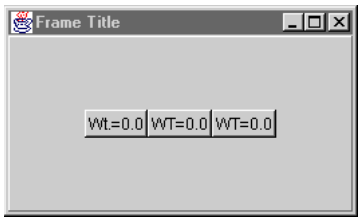
1. In the Property Inspector click the `constraints` property to display the Constraints editor.
2. Enter a value between 0.0 and 1.0 for the X (*weightx*) or Y (*weighty*) value in the Weight area, then press **OK**.

> **Note:** Because weight constraints can make the sizing behavior in the Java Visual Editor difficult to predict, setting these constraints should be the last step in designing a `GridBagLayout`.

Examples of How Weight Constraints Affect Components' Behavior

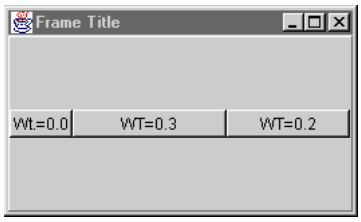
- As shown in Figure 20–12, if all the components have weight constraints of zero in a single direction, the components will clump together in the center of the container for that dimension and won't expand beyond their preferred size. `GridBagLayout` puts any extra space between its grid of cells and the edges of the container.

Figure 20–12 Weight Constraints Set to Zero



- As shown in Figure 20–13, if you have three components with *weightx* constraints of 0.0, 0.3, and 0.2 respectively, when the container is enlarged, none of the extra space will go to the first component, 3/5 of it will go the second component, and 2/5 of it will go to the third.

Figure 20–13 Three Components with Weight Constraints Set to 0.0, 0.3, and 0.2

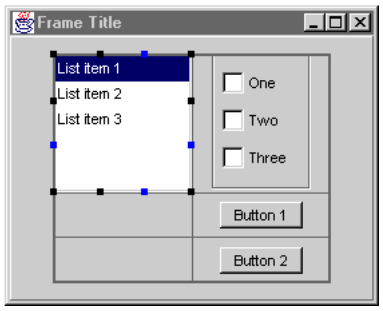


- You need to set both the weight and fill constraints for a component if you want it to grow. For example, if a component has a weightx constraint, but no horizontal fill constraint, then the extra space goes to the padding between the left and right edges of the component and the edges of the cell. It enlarges the width of the cell without changing the size of the component. If a component has both weight and

fill constraints, then the extra space is added to the cell, plus the component expands to fill the new cell dimension in the direction of the fill constraint (horizontal in this case).

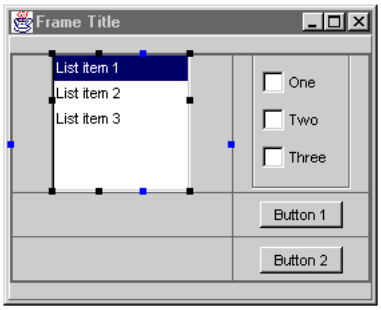
In Figure 20–14, all the components in the *GridBagLayout* panel have a weight constraint value of zero. Because of this constraint, the components are clustered in the center of the *GridBagLayout* panel, with all the extra space in the panel distributed between the outside edges of the grid and the panel. The size of the grid is determined by the preferred size of the components, plus any insets and padding (*ipadx* or *ipady*).

Figure 20–14 All Components with Weight Constraint Value Set to Zero



In Figure 20–15, a horizontal weight constraint of 1.0 is specified for the *ListControl*. Notice that as soon as one component is assigned any weight, the UI design is no longer centered in the panel. Since a horizontal weight constraint was used, the *GridBagLayout* manager takes the extra space in the panel that was previously on each side of the grid, and puts it into the cell containing the *ListControl*. Also notice that the *ListControl* did not change size.

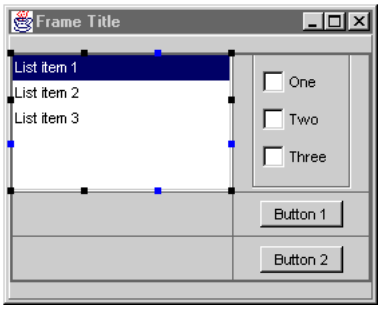
Figure 20–15 Horizontal Weight Constraint Set to 1.0 For ListControl.



Tip: f there is more space than you like inside the cells after adding weight to the components, decrease the size of the UI frame until the amount of extra space is what you want. To decrease the size of the frame, select the frame in the Java Visual Editor or the Structure window, `this(BorderLayout)`, then click its black sizing nibs and drag the frame to the desired size.

As shown in Figure 20–16, if a horizontal `fill` is then added to the *ListControl*, the component expands to fill the new horizontal dimension of the cell.

Figure 20–16 Horizontal Fill Added to ListControl



- If one component in a column has a *weightx* value, GridBagLayout gives the whole column that weight. Conversely, if one component in a row has a *weighty* value, the whole row is assigned that weight.

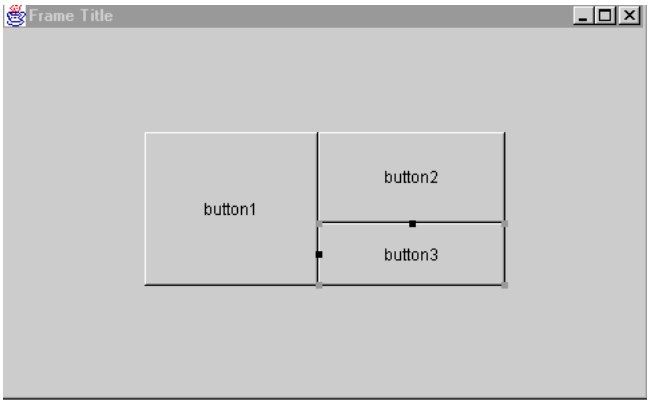
20.5.17 Using OverlayLayout2

OverlayLayout2 arranges components over the top of each other. The requested size of the container will be the largest requested size of the children, taking alignment needs into consideration. The alignment is based upon what is needed to properly fit the children in the allocation area. The children will be placed such that their alignment points are all on top of each other. Unlike BorderLayout's centering, OverlayLayout2 does not expand the component to fill the available space, but instead leaves it at its preferred size.

20.5.18 Using PaneLayout

As shown in Figure 20–17, PaneLayout allows you to specify the size of a component in relation to its sibling components. PaneLayout applied to a panel or frame lets you control the percentage of the container the components will have relative to each other, but does not create moveable splitter bars between the panes.

Figure 20–17 PaneLayout



In a PaneLayout, the placement and size of each component is specified relative to the components that have already been added to the container. Each component specifies a PaneConstraints object that tells the layout manager from which component to take space, and how much of its existing space to take. Each component's PaneConstraints object is applied to the container as it existed at the

time the component was added to the container. The order in which you add the components to the container is very important.

The constraint for a PaneConstraints component that is being added to a container consists of four variables:

- **String name**

  The name for this component (must be unique for all components in the container - as in `CardLayout`).

- **String *splitComponentName***

  The name of the component from which space will be taken to make room for this component.

- **String *position***

  The edge of the `splitComponentName` to which this component will be anchored.

  Table 20–5 shows valid values.

Table 20–5 Valid values for String position

Select this	To do this
PaneConstraints.TOP	This component will be above `splitComponentName`.
PaneConstraints.BOTTOM	This component will be below `splitComponentName`.
PaneConstraints.RIGHT	This component will be to the right of `splitComponentName`.
PaneConstraints.LEFT	This component will be to the left of `splitComponentName`.
PaneConstraints.ROOT	This component is the first component added.

- **float *proportion***

  The proportion of `splitComponentName` that will be allocated to this component. A number between 0 and 1.

20.5.19 How Components are Added to PaneLayout

Components are added to PaneLayout as follows:

- The first component will always take all the area of the container. The only important variable in its `PaneConstraint` is its *name*. Other components will use this value, specifying it as their `splitComponentName`.
- The second component must specify its `splitComponentName` as the *name* of the first component.
- The `splitComponentName` of subsequent components may be the *name* of any component that has already been added to the container.

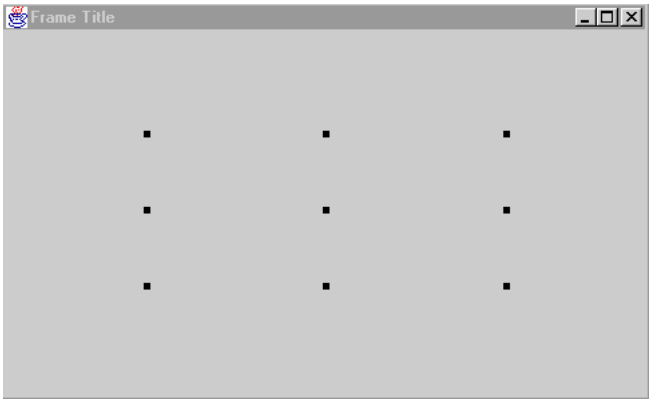
20.5.20 How to Create a PaneLayout Container in the Java Visual Editor

You can create a PaneLayout container in the Java Visual Editor.

To create a PaneLayout Container:

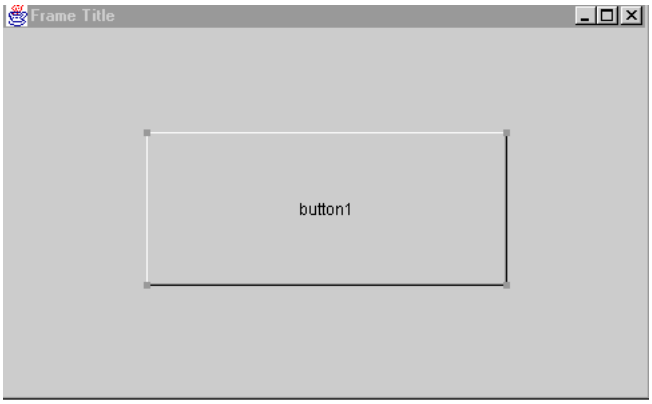
1. Add a container to your UI in the Java Visual Editor. This can be any kind of a frame or panel, such as the one shown in Figure 20–18.

Figure 20–18 UI Container in Java Visual Editor



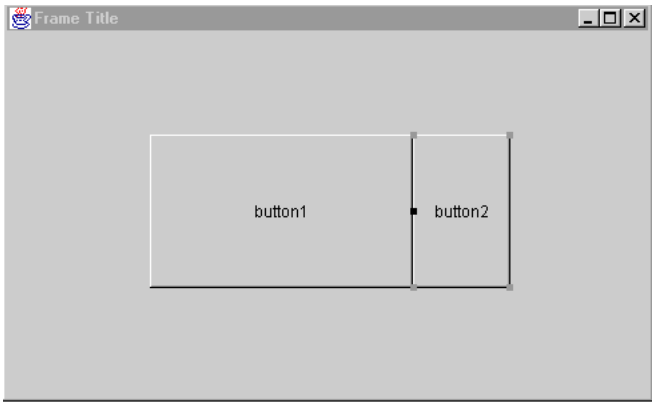
2. Change the container's layout property to `PaneLayout`. This allows you to access the `PaneLayout` properties in the Inspector.
3. From the Component Palette, select the first component and drop it into the `PaneLayout` container. This component will completely fill the container until you add another component.

Figure 20–19 Component Dropped in PaneLayout Container



4. Select the second component and drag it to the desired size in the desired position.

Figure 20–20 Second Component Dragged into PaneLayout

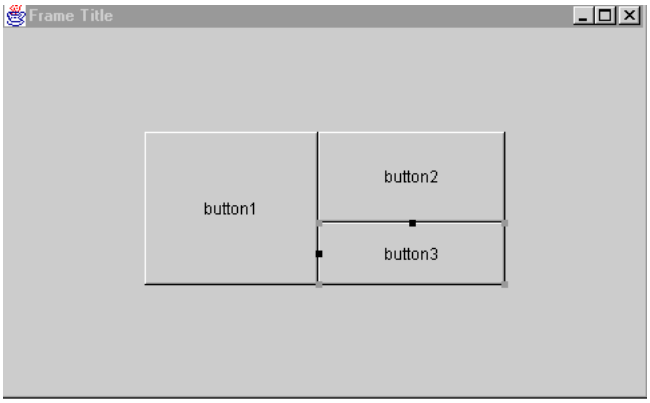


Note: f the first component you added to a `PaneLayout` container was itself a container, the Java Visual Editor assumes you are trying to add the second component to the outer container instead of to the `PaneLayout` container. Use the component tree to specify to the containers that you want the component to be placed in.

5. To add a third component to the `PaneLayout`, draw it similarly to define its position relative to the other components.

   For example, to split the right half of the container, begin drawing the third component starting from the middle of the right edge of the panel to the bottom right corner of the first component.

Figure 20–21 Third Component Dragged into PaneLayout



6. Use the same method to add subsequent components.

20.5.21 Using VerticalFlowLayout

`VerticalFlowLayout` arranges components in columns from top to bottom, then left to right using each component's `preferredSize`. `VerticalFlowLayout` lines up as many components as it can in a column, then moves to a new column. Typically, `VerticalFlowLayout` is used to arrange buttons on a panel.

Figure 20–22 VerticalFlowLayout

You can choose how to arrange the components in the columns of a `VerticalFlowLayout` container by specifying an alignment justification of top, middle, or bottom. You can also specify the amount of gap (horizontal and vertical spacing) between components and columns. It also has properties that let you specify the components should fill the width of the column, or the last component should fill the remaining height of the container. Use the Inspector to change these properties when you're in the Java Visual Editor.

Alignment

- TOP - groups the components at the top of the container.
- MIDDLE - centers the components vertically in the container.
- BOTTOM - groups the components so the last component is at the bottom of the container.

The default alignment in a `VerticalFlowLayout` is MIDDLE.

To change the alignment, select the `verticalFlowLayout` object in the Structure window, then specify a value in the Inspector for the alignment property as follows:

- 0=TOP
- 1=Middle
- 2=BOTTOM

Gap

The default gap between components in a `VerticalFlowLayout` is 5 pixels.

To change the horizontal or vertical gap, select the `VerticalFlowLayout` object in the Structure window, then modify the pixel value of the hgap (horizontal gap) or vgap (vertical gap) property in the Inspector.

Order of Components

To change the order of the components in a `VerticalFlowLayout` container, drag the component to the new location.

Horizontal Fill

`horizontalFill` lets you specify a fill to edge flag which causes all the components to expand to the container's width.

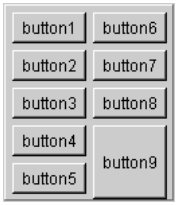
Figure 20–23 horizontalFill

WARNING: Your program can become unstable if the main panel has less space than it needs. This property also prohibits multi-column output.

Vertical fill

`verticalFill` lets you specify a vertical fill flag that causes the last component to fill the remaining height of the container.

Figure 20–24 verticalFill



The default value for `verticalFill` is `False`.

20.5.22 Using XYLayout

`XYLayout` is a JDeveloper custom layout manager. XYLayout puts components in a container at specific (x,y) coordinates relative to the upper left corner of the container. Regardless of the type of display, the container will always retain the relative (x,y) positions of components. However, when you resize a container with an `XYLayout`, the components do not reposition or resize.

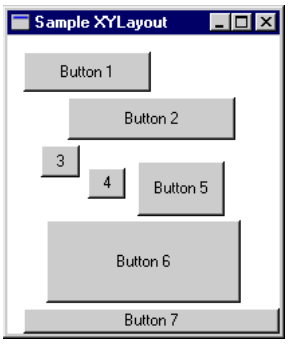
For `XYLayout` containers, the preferredSize of the container is defined by the values specified in the width and height properties of the `XYLayout`. For example, if you have the following lines of code in your container initialization,

```
xYLayoutN.setWidth(400);

xYLayoutN.setHeight(300);
```

and if `xYLayoutN` is the layout manager for the container, then its `preferredSize` is 400 x 300 pixels.

If one of the nested panels in your UI uses `XYLayout`, that panel's `preferredSize` will be determined by the layout's `setWidth()` and `setHeight()` calls. This value will be used for the panel in computing the preferredSize of the next (outer) container.

Figure 20–25 XYLayout



You'll discover that `XYLayout` is very convenient to use in prototyping a user interface. When you design more complicated user interfaces with multiple nested panels, `XYLayout` can be used for the initial layout of the panels and components, after which you can choose from one of the standard layouts for the final design.

> **Note:** `XYLayout` uses absolute x,y values when positioning objects on the screen, and does not adjust to different screen resolutions. To ensure your layout adjusts to other resolutions, don't leave any containers in `XYLayout` in your final design.

You can use the UI design tools to specify the container's size and its components' x,y coordinates.

- To specify the size of a container using `XYLayout`, select the `XYLayout` object in the Structure window and enter the pixel dimension for the height and width properties in the Property Inspector. This setting specifies the size of the `XYLayout` container.
- To change the (x,y) values for a container using **XYLayout**, do one of the following:
  - In the Java Visual Editor, drag the component to a new size. JDeveloper automatically updates the constraint values in the Property Inspector.
  - Select the component in the Structure window, then click the `constraints` property edit field and enter coordinates for that component.

  the alignment options available from the context menu.

Table 20–6 Alignment Options

Select this	To do this
Move to first	Moves the selected component to the top of the Z-order.
Move to last	Moves the selected component to the bottom of the Z-order.
Align left	Lines up the left edges of the selected components with the left edge of the first selected component.
Align center	Horizontally lines up the centers of the selected components with the center of the first selected component.
Align right	Lines up the right edges of the selected components with the right edge of the first selected component.

Table 20–6 (Cont.) Alignment Options

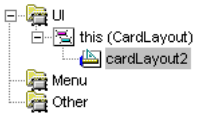
Select this	To do this
Align top	Lines up the top edges of the selected components with the top edge of the first selected component.
Align middle	Vertically lines up the centers of the selected components with the middle of the first selected component.
Align bottom	Lines up the bottom edges of the selected components with the bottom edge of the first selected component.
Even space horizontal	Evenly spaces the selected components horizontally between the first and last selected components.
Even space vertical	Evenly spaces the selected components vertically between the first and last selected components.
Same size horizontal	Makes all the selected components the same width as the first selected component.
Same size vertical	Makes all the selected components the same height as the first selected component.

A simpler approach to XYLayout is to prototype the UI directly in the Java Visual Editor by setting the layout to null. UI components are set where you place them at the size you create them. For more information, see Section 20.6, "Prototyping Your UI with Layout Properties".

20.5.23 Understanding Layout Properties

Each container normally has some kind of layout manager attached to its layout property. The layout manager has properties that can affect the sizing and location of all components that are added to the container. You can view and edit these properties in the Inspector when the layout manager is selected in the Structure window. The layout manager displays as an item in the Structure window just below the container to which it is attached.

Figure 20–26 Layout Properties



20.5.24 Understanding Layout Constraints

For each component you add to a container, JDeveloper may instantiate a constraints object, or produce a constraint value, which provides additional information about how the layout manager should size and locate this specific component. The type of constraint object or value created depends upon the type of layout manager being used. The Inspector displays the constraints of each component as if they were properties of the component itself, and allows you to edit them.

20.5.25 Determining the Size and Location of Your UI Window at Runtime

If your UI class is a `Frame` or `Dialog`, you can control its size and location at runtime. The size and location are determined by what the code does when the UI window is created and what the user does to resize or reposition it.

When the UI window is created, and various components are added to it, each component that is added affects the `preferredSize` of the overall window, typically making the preferredSize of the window container larger as additional components are added. This effect on `preferredSize` depends on the layout manager of the outer container, as well as any nested container layouts. For more details about the way that preferred`LayoutSize` is calculated for various layouts, see the sections in this document on each type of layout.

The size of the UI window, as set by your program (before any additional resizing that may be done by the user), is determined by which of the following container methods is called last in the code:

- `pack()`
- `setsize()`

The location of your UI at runtime will be at 0,0 unless you override this by setting the location property of the container (for example by calling setLocation() before making it visible).

20.5.26 Sizing a Window Automatically with pack()

The `pack()` method computes a window's `preferredSize`, based upon the components it contains, and sizes itself accordingly. `pack()` creates the smallest possible window, while still respecting the preferredSize of the components that are placed within it.

> **Note:** The Application.java file created by the New Application dialog calls `pack()`, packing the frame to its `preferredSize` before making it visible.

20.5.27 How the preferredSize is Calculated for a Container

`preferredSize` is calculated differently for containers with different layouts

20.5.28 Portable Layouts

Portable layouts, such as `FlowLayout` and `BorderLayout`, calculate their preferredSize based on a combination of the layout rules and the preferredSize of each component that was added to the container. If any of the components are containers (such as a Panel), then the `preferredSize` of that Panel is calculated according to its layout and components. The layout calculation is recursed into as many layers of nested containers as necessary. For more information about `preferredSize` calculation for particular layouts, see the individual layout descriptions.

20.5.29 Explicitly Setting the Size of a Window Using setSize()

If you call `setSize()` on the container (rather than `pack()` or subsequent to calling `pack()`), the size of the container will be set to a specific size, in pixels. `setSize()` overrides the effect of pack() and `preferredSize` for the container.

> **Note:** Because different screens have different pixel sizes, if you use `setSize()` you must call `validate()` in order for child containers to be properly laid out. Note that `pack()` automatically calls `validate()`.

20.5.30 Making the Size of your UI Portable to Various Platforms

To make your UI portable, either use pack() or calculate an appropriate size to use with `setSize()` based on the pixel sizes of the various screens your application will be deployed on.

For example, you might want the UI to appear at 75% of the width and height of the screen. For the UI to appear at this sizing, you can add the following lines of code to your application class instead of calling `pack()`:

```
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
frame.setSize(screenSize.width * 3 / 4, screenSize.height * 3 / 4);
```

Note: To ensure portability, change all XYLayout containers to a portable layout after prototyping. For more information, see Section 20.6, "Prototyping Your UI with Layout Properties".

20.5.31 Positioning a Window on the Screen

If you don't explicitly position your UI, it will appear in the upper left corner of the screen.

To center the UI on the screen, obtain the width and height of the screen, subtract the width and height of your UI, divide the difference by two (in order to create equal margins on opposite sides of the UI), and use these figures for the location of the upper left corner of your UI.

The code in Example 20–1 is generated when you select the **Center Frame on Screen** option in the New Application dialog, and performs this calculation for you.

Example 20–1 Code Generated When You Select Center Frame on Screen Option

```
//Center the window
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = frame.getSize();
if (frameSize.height > screenSize.height)
  frameSize.height = screenSize.height;
if (frameSize.width > screenSize.width)
  frameSize.width = screenSize.width;
frame.setLocation((screenSize.width- frameSize.width) / 2,
(screenSize.height - frameSize.height) /2);
```

20.5.32 Placing the Sizing and Positioning Method Calls in your Code

The calls to `pack()`, `validate()`, `setSize()`, or `setLocation()` can be made from inside the UI container class for example, `this.pack()`. These calls can also be made from the class that creates the container, for example, `frame.pack()`, after invoking the constructor, before `setVisible()`. The latter is what the New Application dialog-generated code does; the calls to `pack()` or `validate()`, and `setLocation()` are placed in the Application class, after the frame is constructed (after the call to `jbInit()`).

If you are constructing the UI from various places within your application, and you want it to come up in the same size and place, place your calls in the constructor of your UI container class (after the call to `jbInit()`. If your application instantiates the UI from only one place, such as in the New Application dialog generated application,

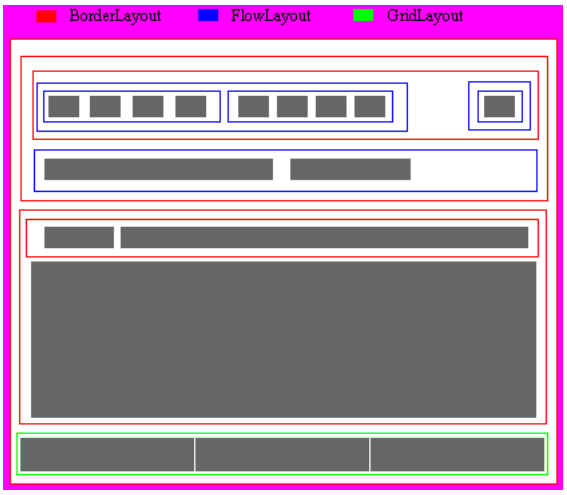
put the sizing and positioning code in the place where the UI is created, in this case the Application class.

20.5.33 Working with Nested Containers and Layouts

Most UI designs in Java use more than one type of layout to get the desired component positions by nesting multiple panels with different layouts in the main container. By creating a composite design, and by using the appropriate layout manager for each panel, you can group and arrange components in a way that is both functional and portable.

For example, Figure 20–27 demonstrates the use of nested panels with different layouts. The solid gray objects are the buttons and other visible components in the UI which are nested in various levels of panels.

Figure 20–27 Nested Panels



20.5.33.1 How to Create Nested Panels

You can nest panels within other panels to gain more control over the placement of components.

To create nested panels:

1. Add a panel to the UI design of your application. For more information, see Section 20.7.9, "How to Create a Panel."
2. In the Property Inspector, change the Layout property to null.

   When you add a panel to the UI design, it uses `FlowLayout`. Change the layout to `null` initially because it is the easiest layout to work with during the design process. After you have all the panels and other components in place in your UI, you will set it to its final layout.

   To locate a specific panel in the Java Visual Editor, you can select the panel in the Structure window.
3. Place panels within panels to logically group components.
   - Make sure components are fully nested inside their panels. Examine the Structure window to make sure each component is indented under the correct panel in the tree outline.

- The components will probably be irregular sizes and shapes. Don't worry about getting it perfect at this point, because when you change the panel layouts, the layout manager will realign the components.

4. Set the layout property for each panel to the appropriate layout. For more information, see Section 20.5.23, "Understanding Layout Properties."
   - Be aware that when you change a layout manager for one panel, the effects may change again when you change the layout manager for the panel that holds it.
   - Select the layout object in the Structure window to change its settings.

20.5.34 Adding Custom Layout Managers

JDeveloper supports the integration of other layout managers with its Java Visual Editor. Users will be able to work with the layout manager using both the Java Visual Editor and the Property Inspector when you:

1. Write an `oracle.jdevimpl.uieditor.LayoutAssistant` implementation to be associated with the `LayoutManager`.
2. Register the `LayoutAssistant` implementation with the IDE.
3. (Optionally) Register a `java.beans.PropertyEditor` with the IDE to handle any special constraints type.

To create a custom layout manager assistant:

Each `LayoutManager` must be associated with a `LayoutAssistant` implementation. The class `oracle.jdevimpl.uieditor.assistant.BasicLayoutAssistant` provides a minimal implementation of the interface `oracle.jdevimpl.uieditor.LayoutAssistant` and will be used for any `LayoutManager` for which no explicit registration has been made. There are several required methods in the interface, but it is beyond the scope of this topic to describe them. Integrators may wish to subclass their `LayoutAssistant` implementations from this minimal implementation. Refer to the Javadoc for the `LayoutAssistant` interface for details.

To register the layout assistant:

To register the new layout assistant, you need to add a key-value definition to `oracle.jdevimpl.uieditor.UIEditorAddin` section in the `JDeveloper\lib\addins.xml` file as follows:

```
<property>
  <key>PREFIX.LAYOUT_MANAGER_CLASS_NAME</key>
  <value>LAYOUT_ASSISTANT_CLASS_NAME</value>
</property>
```

where:

- PREFIX is `jdeveloper.uiassistant`
- LAYOUT_MANAGER_CLASS_NAME is the fully-qualified classname of the layout manager for which an assistant is being registered
- LAYOUT_ASSISTANT_CLASS_NAME is the fully-qualified classname of the layout manager assistant to register

For example, if you were to register a `LayoutAssistant` implementation `oracle.jdevimpl.uieditor.assistant.GridBagLayoutAssistant` for the `java.awt.GridBagLayout` layout manager, the property to add would look like:

```
<property>
  <key>jdeveloper.uiassistant.java.awt.GridBagLayout</key>
  <value>oracle.jdevimpl.uieditor.assistant.GridBagLayoutAssistant</value>
</property>
```

Note that in order for the layout assistant to be available from within the IDE, so that it will appear in the Property Inspector's layout property list, the layout assistant must be added to the IDEClasspath as a directive in `JDeveloper\bin\jdev.conf` file. For example:

```
AddJavaLibFile <myAssistant.jar>
```

where:

`myAssistant.jar` contains the compiled class file for your `LayoutAssistant` implementation.

To register a constraints property editor for the layout manager:

If your custom layout manager uses a constraint class, you may want to register a class `implementing java.beans.PropertyEditor` for editing the constraints. You register the property editor class under `Inspector.PropertyEditor` in the ide.properties file, where:

- `Inspector.PropertyEditor.count=#` provides the total number of property editors that are registered in ide.properties. You must increment the count for each new property editor you wish to add.
- `<#>` is the zero-based index, in the list of registered editors, corresponding to the editor you are registering.
- `<fully qualified type>` is the fully-qualified type of the constraints for which a PropertyEditor is being registered.
- `<Fully qualified class>` is the fully-qualified classname of the `java.beans.PropertyEditor` implementation being registered for editing instances of the given constraints type.

For example, if you were to register the property editor for the GridBagLayout layout manager's GridBagConstraints, your entry in the ide.properties file would look like:

```
Inspector.PropertyEditor.count=1

Inspector.PropertyEditor.editor1.type=java.awt.GridBagConstraints
Inspector.PropertyEditor.editor1.editor=oracle.jdevimpl.uieditor.assistant.GridBag
ConstraintsEditor
```

Note: Your property editor class also needs to be in the IDEClasspath as described above for the layout manager assistant.

20.6 Prototyping Your UI with Layout Properties

Before you start creating your UI, it is useful to sketch your UI design on paper to get an idea of the general strategy you'll use for placing various panels and components, and for assigning layouts. You can also prototype your UI directly in the Java Visual

Editor. JDeveloper makes this easy by providing a layout called `null` that leaves the components where you place them at the size you create them.

20.6.1 Using null Layout for Prototyping

To make this approach simpler, JDeveloper provides the null layout. When you start a project using the New Application dialog, JDeveloper generates a UI container class (usually one that extends Frame or JFrame) that uses `null`. You can open the frame in the Java Visual Editor and do your design work directly on the frame.

When you initially add a new panel of any type to the Java Visual Editor, you'll notice that the layout property in the Inspector says `<default layout>`, which means that the Java Visual Editor will automatically use the default layout for that container. However, you may want to change the layout property to the layout manager you want to use so it is visible in the Structure window and that component constraints can be modified in the Property Inspector. You cannot edit layout properties for `<default layout>`.

20.6.2 Designing the Big Regions First

Design the big regions of your UI first, then, using null, work down into finer details within those regions. Once the design is right, work systematically from the inner regions outward, converting the panels to more portable layouts such as `FlowLayout`, `BorderLayout`, or `GridLayout`, making minor adjustments if necessary.

In most cases, you will place a container in your design first, then add components to it. You can also draw a new container around existing components. However, these components will not automatically nest into the new panel. After drawing the container, you must move each component in the container. You may even need to move a component out of the container, then back in. Check the Structure window to make sure each component nests properly. Each component inside a container should be indented in the Tree under its container.

20.6.3 Saving Before Experimenting

When designing in JDeveloper, expect to work by trial and error, especially when changing the layouts. Be sure to save your work before experimenting with a layout change.

You may discover that a particular layout you planned to use doesn't work as you expected. You may need to reexamine your design process and use a different configuration of containers, components, and layouts. For this reason, you may want to copy the container file (for example `Frame1.java`) to a different name and safe location at critical points so that, when you need to back up in your work, you don't need to start over completely.

20.6.4 Selecting a Final Layout Manager

If you've used the `null` or `XYLayout Manager` to design your Java program, you'll need to change the layout manager to one of the standard layout managers that is supported across different platforms.

To change the layout manager, see Section 20.8.3, "How to Change the Layout for a Container".

20.7 Working with Containers and Components

JDeveloper provides easy tools to help you generate your containers. Containers hold and manage other components. When you lay out a form to design the UI in JDeveloper you use containers. There are two general classes of containers:

- Heavyweight Swing containers, extends their AWT equivalents and are thus heavyweight implementations, and include JFrame, JDialog, and JApplet.
- Lightweight Swing containers, which don't contain operating system-specific code, and include, for example, JTabbedPane, JPanel, JSplitPane, and JScrollPane.

Containers are also components; you interact with them by getting and setting their properties, calling their methods, and responding to their events as with any other component.

20.7.1 Using Windows

A Window is a stand-alone top-level container component with no borders, title bar, or menu bar. Although a Window could be used to implement a popup window, you normally use a subclass of Window in your UI such as one of those listed in Table 20–7.

Table 20–7 Windows Subclasses

Window	Description
Frame	A top-level window with a border and a title. A Frame has standard window controls such as a control menu, buttons to minimize and maximize the window, and controls for resizing the window. It can also contain a menu bar.
Dialog box	A popup window, similar to a Frame, but it needs a parent. Dialog boxes are used for getting input from the user or to present warning messages. It can also contain a menu bar. Dialog boxes are usually intended to be transient, or temporary, and can be one of the following types: ■ *Modal*: Prevents user input to any other windows in the application until that dialog box is dismissed. ■ *Modeless:* Lets the user enter information in both the dialog box and the application.

20.7.2 Using Panels

A Panel is a simple UI container, without border or caption, used to group other components, like buttons, checkboxes, and text fields. Panels are embedded within some other UI, such as in a frame or dialog. They can also be nested within other panels.

Table 20–8 Panels

Panel	Description
Applet	A subclass of Panel used to build a program intended to be embedded in an HTML page and run in an Internet browser or applet viewer. Since Applet is a subclass of Panel, it can contain components, but does not have a border or caption.

20.7.3 Using Lightweight Swing Containers

The lightweight Swing containers available from JDeveloper's Component Palette pages include JMenuBar, JPopupMenu, JSplitPane, JScrollPane, JTabbedPane, and JToolbar. All are a subclass of JComponent. You can add other containers and components to these containers, combining components and their containers in various ways to get the interface you want.

Table 20–9 Lightweight Swing Containers

Lightweight Container	Description
Menu Bar	The lightweight Swing containers available from the Component Palette pages include JMenuBar, JPopupMenu, JSplitPane, JScrollPane, JTabbedPane, and JToolbar. All are a subclass of JComponent. You can add other containers and components to these containers, combining components and their containers in various ways to get the interface you want.
Popup Menu	A small window which pops up and displays a series of choices, which you create in JDeveloper using the Menu Editor. A JPopupMenu is used for the menu that appears when the user selects an item on the menu bar. It is also used for "pull-right" menu that appears when the selects a menu item that activates it. Finally, a JPopupMenu can also be used anywhere else you want a menu to appear -- for example, when the user right-clicks in a specified area.
Split Pane	Manages two panes that are separated horizontally or vertically by a divider that can be repositioned by the user. You can choose which pane to add a component to. You can specify the components with the properties `leftComponent` and `rightComponent`, or `topComponent` and `bottomComponent`. In these properties, "Left" is equivalent to "Top" and "Right" is equivalent to "Bottom" -- so if you change the arrangement, your existing code still works. Subsequent adds to the same pane replace its contents with the new object.
Scroll Pane	Manages two panes that are separated horizontally or vertically by a divider that can be repositioned by the user. You can choose which pane to add a component to. You can specify the components with the properties leftComponent and rightComponent, or topComponent and bottomComponent. In these properties, "Left" is equivalent to "Top" and "Right" is equivalent to "Bottom" -- so if you change the arrangement, your existing code still works. Subsequent adds to the same pane replace its contents with the new object. Along with its scroll bars and viewport, a JScrollPane can have a column header and a row header. Each of these is a JViewport object that you specify with `rowHeader` and `columnHeader` properties. The column header `viewport` automatically scrolls left and right, tracking the left-right scrolling of the main viewport. (It never scrolls vertically, however.) The row header acts in a similar fashion.
Tabbed Pane	Manages multiple panels that completely overlap each other. The user can select a panel to view by clicking on a "tab" attached to the panel (like the tab on a file folder). You add tabs in JDeveloper by dropping a JPanel onto the tabbed pane from the Component Palette. The tabPlacement property lets you position tabs on the top, bottom, left side, or right side of the container.

Table 20–9 (Cont.) Lightweight Swing Containers

Lightweight Container	Description
Toolbar	Provides a component which is useful for displaying commonly used Actions or controls. It can be dragged out into a separate window by the user (unless the `floatable` property is set to false). In order for drag-out to work correctly, it is recommended that you add JToolBar instances to one of the four 'sides' of a container whose layout manager is a BorderLayout, and do not add children to any of the other four 'sides'.

20.7.4 Understanding Component Properties in the Property Inspector

A property is a named attribute of a class that can affect its appearance or its behavior. A property can be:

- **Readable**: These properties have a "get" method, which enables you to read the property's value. If it is a Boolean property, it can also use "is" to read the value.
- **Writable**: These properties have a "set" method, which enables the property's value to be changed.
- **Both readable and writable**: These properties have both "get" and "set" methods.

20.7.5 Setting Property Values in the Property Inspector

Properties are attributes that define how a component appears and responds at runtime. In JDeveloper, you set a component's initial properties during design time, and your code can change those properties at runtime.

The Property Inspector window displays the properties of the selected component(s) and is where you set the property values at design time for any component in your design. By setting properties at design time, you are defining the initial state of a component when the UI is instantiated at runtime.

> **Note:** To modify the initial property values at runtime, you can put code in the body of the methods or event handlers which you can create on the **Events** section of the Property Inspector.

20.7.6 Setting Shared Properties for Multiple Components

When you open the Property Inspector with more than one component selected in the Java Visual Editor, by default the Property Inspector displays all the common properties of the selected components.

Despite the fact that you may have selected multiple components in the Java Visual Editor, the Property Inspector still only displays property values for one component at a time. When the properties are shared among the selected components, sometimes the values will be the same, and in other cases, the property values may differ among the components. The Property Inspector helps you to identify which shared properties have differing values by representing the value in italics.

When the values for shared properties differ among the selected components, the property value that is displayed always belongs to the anchor selection. The anchor selection is the one that appears with hollow selection handles in the Java Visual Editor. You may alter the anchor selection by holding down the Shift key and clicking one of the other currently selected components. Altering the anchor selection changes which components' properties are shown in the Inspector, as well as altering which

component the Java Visual Editor context menu operations apply to (for example, the Align context menu item).

When you change any of the shared properties in the Property Inspector, the property value changes to the new value in all the selected components.

20.7.7 Laying Out Your User Interface

This section explains the fundamental tasks you perform as you work with components and JDeveloper's UI design tools to create a user interface. If you're comfortable using controls in a graphical user interface environment, much of the material discussed here, such as selecting, sizing, and deleting components might be familiar to you.

Before you begin actually creating your UI, you may want to prototype your UI. For more information, see Section 20.6, "Prototyping Your UI with Layout Properties".

To design a user interface in JDeveloper:

- Create containers such as frames, panels, or dialogs. These containers hold specific types of components.
- Add and arrange components in the containers. You can add components to a container and then arrange them using layout managers. For more information, see Section 20.8.1, "How to Add Components to Your User Interface".
- Set component properties. You can set properties for each component using the Property Inspector. For more information, see Section 20.7.4, "Understanding Component Properties in the Property Inspector".
- Attach event-handling code to a component event. Events are the actions a component takes when they are triggered by a user or another component. For more information, see Section 20.10.2, "How to Attach Event-Handling Code to a Component Event".
- Set container layouts and component constraints. A layout constraint tells the layout manager how to size and position a component. For more information, see Section 20.8.4, "How to Modify Component Layout Constraints".

To start your project:

These instructions assume that you have already created a project that includes a designable container class. If not, you will need to:

1. Create or open a JDeveloper Project.
2. Create an applet or an application.

20.7.8 How to Create a Frame

A frame is a top-level window with a border and a title. It has standard window controls such as a control menu, buttons to minimize and maximize the window, and controls for resizing the window.

The New Frame dialog adds a new class to the active project. It adds necessary import statements, creates a default constructor, and it creates a `jbInit()` method in which JDeveloper sets properties and other initialization code used by the Java Visual Editor.

To add a frame:

1. Open or create a project.

2. Choose **File** > **New** to locate the New Frame dialog in the New Gallery.
3. In the **Categories** list, expand **Client Tier** and select **Swing/AWT**.
4. In the **Items** list, select **Frame** and click **OK** to launch the New Frame dialog.
5. In the New Frame dialog, enter the name of the package and class.
6. Choose which frame type to extend.
7. Type the frame title.
8. If there are any options (such as Menu Bar) select any you feel are appropriate.
9. Click **OK** to create the frame and its source code.

   The frame is displayed as a `.java` source file in the Navigator.

To view the frame in JDeveloper:

- Right-click the file in the Navigator and choose Open and click the Design tab to use the interactive UI design tools, such as the Component Palette and the Property Inspector.
- Click the Source tab begin editing the source code directly.

20.7.9 How to Create a Panel

A *panel* is a UI container that groups components such as buttons, checkboxes, and text fields. A panel has a border and may have a title if the border selected is a TitledBorder. Typically, a panel is embedded within a dialog box or frame.

The New Panel dialog adds a new class to the opened project that extends a panel you select. It creates a default constructor, and a `jbInit()` method in which JDeveloper puts property setters and other initialization code used by the Java Visual Editor.

To create a panel:

1. Open or create a project.
2. Choose **File** > **New** to locate the New Panel dialog in the New Gallery.
3. In the **Categories** list, expand **Client Tier** and select **Swing/AWT**.
4. In the **Items** list, select **Panel** and click **OK** to launch the New Panel dialog.
5. In the New Panel dialog, enter a name of the panel's class and package.
6. Choose the base class from which the panel is derived.

   You can choose from any of the base classes installed with JDeveloper. If you prefer, you can search for another class that isn't from an installed package using the browse button. By default, the selection is limited to the panel container provided with the core J2SE (Java 2, Standard Edition) and Swing classes.
7. Click **OK** to create the panel and its source code.

   The panel is displayed as a `.java` source file in the Navigator.

To view the panel in JDeveloper:

1. Right-click the file in the Navigator and choose **Open** and click the **Design** tab to use the interactive UI design tools, such as the Component Palette and the Property Inspector.
2. Click the **Source** tab begin editing the source code directly.

20.7.10 How to Create a Dialog Box

A dialog box is a popup window with a border and a title. Dialog boxes are typically used to collect user input.

The New Dialog dialog creates a new class that extends Dialog or JDialog and adds it to the current project. It adds the necessary import statement. It also creates a `jbInit()` method in which JDeveloper puts property setters and other initialization code used by the Java Visual Editor. The `jbInit()` method will be invoked when using any of the constructors.

After adding the dialog box, you can design the dialog directly using the Java Visual Editor. This is how you add buttons and other controls to your new dialog box.

To create a dialog box:

1. Open or create a project.
2. Choose **File** > **New** to locate the New Panel dialog in the New Gallery.
3. In the **Categories** list, expand **Client Tier** and select **Swing/AWT**.
4. In the **Items** list, select **Dialog** and click **OK** to launch the New Dialog dialog.
5. In the New Dialog dialog, enter the name of the dialog box's package and class. The file name is automatically filled in for you; it is assigned the same name as the class and is saved to the package directory.
6. Choose whether to extend `java.awt.Dialog` or `javax.swing.JDialog`.
7. Click **OK** to create the dialog box and its source code.

   The dialog box is displayed as a `.java` source file in the Navigation window.

To view the dialog box in JDeveloper:

- Right-click the file in the Navigator and choose Open and click the Design tab to use the interactive UI design tools, such as the Component Palette and the Property Inspector.
- Click the **Source** tab begin editing the source code directly

20.7.11 How to Use a Dialog Box That is Not a Bean

Once the dialog box has been created and its UI designed, you will want to test or use your dialog box from some UI in your program.

To test or use your dialog box:

1. Instantiate your dialog class from someplace in your code where you have access to a Frame which can serve as the parent Frame parameter in the dialog constructor. A typical example of this would be a Frame whose UI you are designing, which contains a Button or a MenuItem which is intended to bring up the dialog box. In applets, you can get the Frame by calling `getParent()` on the applet.

   For a modeless dialog box (which we are calling dialog1 in this example), you can use the form of the constructor that takes a single parameter (the parent Frame) as follows:

   ```
   Dialog1 dialog1=new Dialog1(this);
   ```

   For a modal dialog box, you will need to use a form of the constructor that has the boolean modal parameter set to true, such as in the following example:

```
Dialog1 dialog1=new Dialog1(this, true);
```

You can either place this line as an instance variable at the top of the class (in which case the dialog box will be instantiated during the construction of your Frame and be reusable), or you can place this line of code in the actionPerformed event handler for the button that invokes the dialog box (in which case a new instance of the dialog box will be instantiated each time the button is pressed.) Either way, this line instantiates the dialog box, but does not make it visible yet.

(In the case where the dialog is a bean, you must set its `frame` property to the parent frame before calling `show()`, rather than supplying the frame to the constructor.)

2. Before making the instantiated dialog box visible, you should set up any default values that the dialog box fields should display. If you are planning to make your dialog into a Bean, you need to make these dialog box fields accessible as properties. You do this by defining getter and setter methods in your dialog class.
3. Next, you have to cause the dialog box to become visible during the actionPerformed event by entering a line of code inside the event handler that looks like this:

   ```
   dialog1.show();
   ```

4. When the user presses the **OK** button (or the **Apply** button on a modeless dialog box), the code that is using the dialog box will need to call the dialog's property getters to read the user-entered information out of the dialog, then do something with that information.

   - For a modal dialog box, you can do this right after the `show()` method call, because `show()` doesn't return until the modal dialog box is dismissed. You can use a result property to determine whether the **OK** or **Cancel** button was pressed.
   - For a modeless dialog box, show() returns immediately. Because of this, the dialog class itself will need to expose events for each of the button presses. When using the dialog box, you will need to register listeners to the dialog's events, and place code in the event handling methods to use property getters to get the information out of the dialog box.

20.7.12 How to Create a Tabbed Pane

A *tabbed pane* is a UI container that groups components such as buttons, checkboxes, and text fields on multiple panels. Each panel has a title and a tab that the end user clicks to view the panel contents.

To create a tabbed pane:

1. Create a frame or other container. For more information, see Section 20.7.8, "How to Create a Frame."
2. In the `Swing Containers` page of the Component Palette, click the J**TabbedPane** component.
3. Click inside the container in the Java Visual Editor to drop the tabbed pane with its default size.
4. Resize the tabbed pane as desired.
5. To add the first tab to the tabbed pane, click the **JPanel** component in the Swing Containers page of the Component Palette, then click on the **JTabbedPane** inside the Java Visual Editor.

6. To add additional tabs to the tabbed pane, after you click the **JPanel** component in the Component Palette, you must click specifically on the tab itself of a previously added tab panel.
7. To add a layout panel (one that has no tabs) to any of the tabbed pane tabs, click the **JPanel** component in the **Swing Containers** page of the Component Palette, then click the content area of the **JTabbedPane** inside the Java Visual Editor (in this case, do not click the tab itself).

   To work with a panel that is not currently the top most tab in the Java Visual Editor, click directly on the tab. When you select tab, you also raise the panel to the top of the stacking order. Alternatively, you can select tab panels by choosing the desired panel in the Structure Window. To view the panels you have added to the tabbed pane, expand the `UI` folder, expand the **dataPanel** node, and finally expand the **JTabbedPane** node to see the list of JPanels.

20.8 Working with Components in a Container

You can create and manage your container components using the JDeveloper tools.

20.8.1 How to Add Components to Your User Interface

There are several ways to insert a component from the JDeveloper Component Palette into a UI container you create.

- Using the mouse to select, insert, and position a component
- Using keyboard bindings to select and insert a component

When you visually add a component in the Java Visual Editor, JDeveloper generates an instance variable for the component and adds it to the source code. When you delete a component, the Java Visual Editor deletes the associated lines from the code.

To add a component to your UI using the mouse:

1. Create a container component.
2. Choose the desired component list from the Component Palette dropdown menu.
3. Click the desired component in the palette.
4. To insert the selected component into the container, do one of the following:
   - Click inside the container to insert the component at its default size, or
   - Drag the mouse in the Java Visual Editor to form a bounding box from the initial mouse click to a final point which represents the desired dimensions of the object to be created. Dragging to a specific size is only appropriate when layouts for a container that consider individual component dimensions. Ultimately, the layout manager for each container in your UI will determine its components' size and position, or
   - Drop the component onto the desired container in the Structure window. Be aware that this method gives you no control over where the component appears in the container.

To add a component to your container using only keystokes:

1. Create a container component. For more information, see Section 20.8, "Working with Components in a Container."
2. Choose the desired component list from the Component Palette dropdown menu.

3. To select a component from the palette, press the **tab** key to focus on the desired component, then press the **spacebar** to select the component.
4. To insert the selected component into the container, press **Enter.**

   The inserted component appears in the top, lefthand corner of the UI container. You may use the Property Inspector to size and position the inserted component.

To add multiple instances of a component:

Press the **Shift** key while clicking the component in the Component Palette. You may release the **Shift** key and the palette will still remain in multiple creation mode.

Click the arrow tool in the Component Palette to turn off multiple object creation.

20.8.2 How to Set Component Properties at Design Time

The Property Inspector window displays the properties of the component selections you make in the Java Visual Editor. Use the Property Inspector window to set the property values at design time for any component in your UI design.

To set a component's properties at design time:

1. Select a component in the Java Visual Editor or in the Structure window.
2. To display the Property Inspector with the values for the selected component, choose **View,** then **Property Inspector** or right-click the component in the Java Visual Editor and choose Property Inspector. The Property Inspector window is highlighted.
3. To display the Property Inspector with the values for the selected component, choose **View Property Inspector** or right-click the component in the Java Visual Editor and choose Property Inspector. The Property Inspector window is highlighted. OR

   In the Property Inspector toolbar, type the name of the property in the **Find** text field, and press Enter to display the property in the Inspector. If you entered a partial name or more than one property exists by the same name, you can use the Up or Down arrow buttons to jump to properties matching the entered name.
4. Enter the value in the right column one of the following ways:
   - When there is only a text field, you simply type the string value for that property, for example a text value or a number value, then press **Enter**.
   - When the value field is displayed with a down arrow, click the down arrow and choose a value from the list, then press **Enter**.
   - When the value field shows an ellipses (...) button, click it to display a property editor for that property, for example, a color or font selector. Set the values in the property editor, then press **OK**.

To set properties for multiple components:

1. Hold down the **Ctrl** key or **Shift** key, and select the desired components.
2. The Property Inspector displays the properties that are shared by selected components. Select and edit the desired property in the Property Inspector. Editing the value of a shared property will cause all selected components to have the same value. If the value is shown in italic font, that means the value belonging to the anchor component's value differs from the other selected components.

Note: The **Ctrl** key causes the selection state of the selected object to be toggled (from either not selected to selected, else from selected to not selected). Using the **Shift** key to make selections also changes the object's anchor usage:

- If the object is not yet selected, it will become selected and become the anchor.
- If the object is already selected, it will just become the anchor.

Using either the **Ctrl** or **Shift** key, if a control goes from being not selected to selected, it will become the new anchor until the user changes the anchor using the Shift select action.

20.8.3 How to Change the Layout for a Container

JDeveloper provides a layout property in the Inspector, in which you can choose a new layout for any container in the Java Visual Editor.

To select a new layout:

1. Select the container in the Structure window.
2. Click the **Properties** tab in the Inspector, select the layout property, and click its value field.
3. Click the down arrow in the layout property's value field and choose a layout from the dropdown list.

JDeveloper does the following:

- Substitutes the new layout manager in the Structure window.
- Changes the source code to add the new layout manager and updates the container's call to `setLayout`.
- Changes the layout of components in the Java Visual Editor.
- Updates the layout constraints for the container's components in the Inspector and in the source code.

20.8.4 How to Modify Component Layout Constraints

When you drop a component into a container, JDeveloper creates an appropriate constraint object or value for that container's layout manager. JDeveloper automatically inserts this constraint value or object into the constraint property of that component in the Property Inspector. It also adds it to the source code as a parameter of the `add()` method call in the `jbInit()` method.

To edit a component's layout constraints:

1. Select the component in the Java Visual Editor or the Structure window.
2. Select the `constraints` property in the Property Inspector and click its value field.

   When working with a `null` layout manager, there is no constraints property on the children, set the layout constraints on the bounds property instead.
3. Use the Property Editor to modify the constraints, or

Click the desired toolbar constraint button in the Java Visual Editor and, when available, choose the value from the dropdown list.

20.8.5 How to Select Components in Your User Interface

Before attempting to select an existing component in your UI, be sure the selection arrow in the Component Palette is depressed. Otherwise, you may accidentally place a component on your UI.

To select a single component, do one of the following:

- Click the component in the Java Visual Editor.
- With focus on the Java Visual Editor, tab to the component (Tab = forward; Shift+Tab = backward).
- Select the component in the Structure window

To select multiple components, hold down the Ctrl key and do one of the following:

- Click the components in the Java Visual Editor one at a time.
- Click and drag around the outside of the components you want to select.

Note: We recommend using the **Ctrl** key to perform multiple selections because the **Shift** key changes which of the selected objects is the 'anchor' of the selection. The selection anchor is the object whose property values will be displayed in the Property Inspector; it is also the object to which context menu actions apply (i.e., alignment).

As you drag, you surround the components with a rectangle, or "lasso." When this rectangle encloses all the components you want to select, release the mouse button. If necessary, you can then use **Ctrl+click** to individually add or remove components from the selected group.

Hold down the **Ctrl** key and select the components in the Structure window.

20.8.6 How to Size and Move Components

For many layouts, the layout manager determines the size of the components by constraints, making sizing in the Java Visual Editor have no effect. However, when working with `null`, `XYLayout`, or `GridBagLayout`, you can size components when you first place them in your UI, or you can resize and move components later.

Note: `GridBagLayout` currently ignores size on creation, but you can then resize components after creation.

To size a component as you add it:

1. Select the component in the Component Palette.
2. Place the cursor where you want the component to appear in the UI.
3. Drag the mouse pointer before releasing the mouse button. As you drag, an outline appears to indicate the size and position of the control.

4. Release the mouse button when the outline is the size you want.

To move a component one pixel at a time:

1. Make sure that the snap-to-grid feature is turned off (see **Tools** > **Preferences, Java Visual Editor**).
2. Select the component in the Component Palette.
3. Hold down the **Ctrl + shift** keys, then use the direction arrow keys to move the object one pixel at a time

To resize or move a selected component:

1. Click the component in the Java Visual Editor or in the Structure window to select it.

   When you select a component, sizing handles or nibs appear on the perimeter of the component. For some containers, a move handle appears in the middle of the component.
2. Click the appropriate outer handle and drag to resize.
3. Click anywhere in the component and drag it any direction to move it. If the component is a container that is covered with other components, use the center move handle to drag it.

To resize or move a group of selected components:

1. Do one of the following to select the group of components to be changed:
   - Hold down the **Ctrl** key and select each of the components.
   - Hold down the left mouse button and draw a "lasso" around the group of components you want to change.

   When you select a component, sizing handles or nibs appear on the perimeter of the component. For some containers, a move handle appears in the middle of the component.
2. Click the appropriate outer handle and drag to resize.
3. Click anywhere in the component and drag it any direction to move it. If the component is a container that is covered with other components, use the center move handle to drag it.
4. Right-click and choose **Size and Space**, then choose the desired operation.

   OR

   Click the desired size button in the Java Visual Editor toolbar. Operations which are invalid for the currently selected components appear disabled in the toolbar and context menu.

20.8.7 How to Group Components

JDeveloper provides a number of container components for grouping components together so they behave as a single component at design time.

For instance, you might group a row of buttons in a Panel to create a toolbar. Or you could use a container component to create a customized backdrop, status bar, or checkbox group.

When you place components within containers, you create a relationship between the container and the components it contains. Design time operations you perform on the

containers, such as moving, copying, or deleting, also affect any components grouped within them.

To group components by placing them into a container:

1. Add a container to the UI.

   If you are working in a layout that considers size such as the `GridBagLayout`, `null` layout or `XYLayout`, you can drag to size it.

2. Add each component to the container, making sure the mouse pointer falls within the container's boundaries. (The status bar at the bottom of JDeveloper displays which container your mouse is over.) You can add a new component from the Component Palette, or drag an existing component into the new container.

   As you add components, they appear inside the selected container in the Java Visual Editor, and under that container in the Structure window.

   Tip: If you want the components to stay where you put them, change the container's layout to null before adding any components. Otherwise, the size and position of the components will change according to the layout manager used by the container. You can change to a final layout after you finish adding the components.

20.8.8 How to Change Component Z-Order

JDeveloper lets you modify the order in which components, which have been visually stacked one on top of another, appear in the UI design. The topmost component specified by Z-order is the one which the user sees at runtime by default.

Note: Although the Java Visual Editor only displays the topmost component in a stack, you can view the Z-order of the stack in the Structure window: the topmost component appears first in the list of nodes, followed by the next component in the stack, and so on. When you change the Z-order, the Structure window is updated to display the new order.

To change the Z-order of the topmost component:

1. Select the component in the Java Visual Editor from that is currently on top of the stack.
2. Right-click and choose **Order** > **Send to Back**.

To change the Z-order of a component that is not visually on top:

1. Select the stacked component in the Structure window for which you want to change the order.
2. In the Java Visual Editor, right-click and choose **Order**, then choose either **Bring to Front** or **Send to Back**.

20.8.9 How to Cut, Copy, Paste and Delete Components

JDeveloper supports Cut, Copy, Paste and Delete functionality in the Java Visual Editor. You can perform these operations between files of the same project or different projects.

Note: When you cut, copy, paste from one file to a file in another project, you may be required to update your project properties on the target project to include additional libraries. If the target project does not define the right libraries for the pasted object, the paste will fail since the Java Visual Editor will not recognize the class of the incoming objects.

20.8.10 How to Copy a Component

Visual components copied in the Java Visual Editor are not copied to the system clipboard, and cannot be transferred into other applications. Use a screen capture utility to create an image of a control, or copy the source text to use the Java code in another Java development environment.

When you copy a component that has defined event methods, the event listener is copied with the component, but not the event handler. This is because in most cases it's the format of the control, and not the behavior, that you want to copy. If your goal is to copy the control and its behavior to a different Java class file, you need to separately copy the handler: open the file in the Code Editor, locate the handler code, select it, and choose **Edit** > **Copy**.

To copy one or more components:

1. Select the components you want to copy. For more information, see Section 20.8.5, "How to Select Components in Your User Interface."
2. Choose **Edit** > **Copy**.

The components are copied to a local clipboard in JDeveloper.

20.8.11 How to Cut a Component

Before you cut your component be sure to paste the previously cut object since cutting the event code will overwrite the contents of the clipboard.

To cut a component from your user interface:

1. Select the components you want to cut.
2. Choose **Edit** > **Cut**.

The component is removed from the Java Visual Editor and placed into a local clipboard only accessible by JDeveloper (not to the system clipboard). If you quit JDeveloper without pasting the control into a container, the cut version of the control will be lost.

The cut command is the first step in a cut and paste action. If you just want to remove a component, see Deleting a component, below. Deleting a component removes it without changing the contents of your clipboard. If you get in the habit of using the cut command to remove items permanently, there is a chance that one day you will inadvertently replace something in the clipboard that you would rather have kept.

When you cut a component that has defined event methods, the event listener is cut with the component. The event handler is not removed from the source code, nor is it placed on the clipboard. There are two reasons for this:

- In most cases, it isn't the behavior of the control, but the format that you want to retain.

- More than one component may use the same event handler, so removing it from the code may impact other parts of your program.

To cut an event handler in the Code Editor, locate the handler code, select it, and choose **Edit,** then **Cut**.

20.8.12 How to Paste a Component

The components you copy or cut from a JDeveloper Design window can be pasted into any other designable class file.

To paste a component:

1. Open the file to which you want to paste the component in the Java Visual Editor.
2. Select the container to which you want to paste the component.
3. Choose **Edit,** then **Paste**.

The JavaBeans components you paste will add any existing event listener code from the original component to your source code. The event handler code does not get copied or cut with the component: if you want to use the same event handler, you need to copy and paste the handler separately using the Code Editor.

20.8.13 How to Delete a Component from your UI

Delete a component when you want to remove it from your Java program without affecting the contents of the clipboard.

To delete a component:

1. Select the component in the Java Visual Editor or the Structure window.
2. Press the **Delete** key.

When you delete a JavaBeans component from the Java Visual Editor, the event listener methods, if any, are deleted from the source code, but the event handler methods are not. If you want to remove the event handler methods, you need to delete them in the Code Editor.

20.9 Working with Menus

The basic parts of a menu are referred to using the following terms:

- A *menu bar* is displayed at the top of a frame. It is composed of one or more top-level menus, such as File, Edit, or Help. A JMenuBar may have any component as a child, such as a JComboBox or JToggleButton, for example.
- A *menu* is a child of menu bar and contains a collection of menu items, submenus, and separators.
- A *submenu* is a menu whose parent is another menu instead of the menu bar.
- A *menu item* is an individual element on a menu which can invoke a command. Menu items can have attributes such as being disabled (gray) when not allowed, or checkable so their selection state can be toggled.
- An *accelerator,* also known as an keyboard shortcut, allows an alternative way to invoke a menu item. When a menu item has an accelerator, it is displayed at the right of the menu item.

- The *separator bar* helps to visually group related items. It does not invoke a command.

20.9.1 Understanding Menu Components

There are four types of menu component on the Component Palette: a `MenuBar`, `JMenuBar`, `PopupMenu`, and `JPopupMenu`.

- A `MenuBar` or a `JMenuBar` is attached to the main UI Frame, and appears at the top of the frame.
- A `PopupMenu` or a `JPopupMenu` appears when the user right-clicks in your UI. At runtime, popup menus do not appear on the menu bar, instead they are displayed where the user invokes them.

All of these controls can be edited in the Property Inspector.

The first `MenuBar` or `JMenuBar` control dropped onto the UI container is considered the current menubar for your UI. However, you can create more than one menubar for an application; they are displayed in the Inspector in the frame's `MenuBar` property. Select a menu from the `MenuBar` property dropdown list to make it active.

Note: Menu components are only editable at design time in the Menu Editor, not the Java Visual Editor. The menu bar and its top-level menus display in the Java Visual Editor, but they are not selectable and cannot be edited from there. However, you can always see and select them in the Structure window. To see how the menu looks in your UI, run your application.

20.9.2 Using the Menu Editor

You access the Menu Editor by opening the Java file in the Java Visual Editor, which makes the Structure window visible. Then, in the Structure window, when you click on a menu, menu item, or menu root node, the Menu Editor appears.

There are multiple ways that you work with the Menu Editor to create menus:

- Keyboard Arrow, Esc, and Enter keys let you change the focus of the editable menu component
- Drag and drop operations let you move menus or menu items
- Menu Editor toolbar or menu item context menus let you insert menu components

For instance, you can type labels directly into the menu component which has the current focus in the Menu Editor as indicated by the highlighted box. Pressing Enter validates the label and lets you type the next menu component label. You can also use the keyboard arrow keys to move the current focus in the Menu Editor to another position in the menu you want to edit. The Esc key changes the focus from anywhere inside a menu to the menu bar.

In addition to labels that you specify, you can insert various menu components either through the Menu Editor toolbar or the commands duplicated by right-clicking on a menu component. These operations are supported by the toolbar and context menu:

- Insert MenuItem
- Insert Separator
- Insert Submenu

- Enable (or disable) menu item
- Make menu item checkable

Additionally, you can rearrange entire menus or single menu items using drag and drop operations by clicking along the menus in the menu bar or inside the menus in their menu items.

20.9.3 Interacting with the Code Editor and the Property Inspector

JDeveloper synchronizes your changes as you work. As you edit menu items in the Menu Editor, all changes are reflected in the source code by the Code Editor and the Property Inspector. When you make changes to the menus in the source code or the Property Inspector, those changes are reflected in the Menu Editor.

For example, when you add a `Menu` to a `MenuBar` component, this `Menu` appears in the Structure window as a child of the `MenuBar`. Also, when you change properties for `Menu` or `MenuItem` (like text or enabled), those changes are reflected in the code, Menu Editor, and Property Inspector.

Since JDeveloper also maintains synchronization with the Code Editor, there is no need to save your menu design manually. JDeveloper generates the code which you can view and edit in the Code Editor as you use the Menu Editor. The generated code is saved when you save your Java source file. The next time you open the Java file and select a `MenuBar` component in the Structure window, the Menu Editor will open and reload everything for that component.

Once you add a menu component to the UI design, you can use the Menu Editor to design the menu structure. To activate the menus in the user interface, you must use the Property Inspector to attach the menu items to events, or enter the code manually in the Code Editor.

20.9.4 How to Add a Menu Component to a Frame

Since a non-popup menu can only be attached to container, such as a JFrame or a JDialog, you must first open or a container file. You can open one in one of the following ways:

- Open an existing Frame or Dialog file.
- Use the New Gallery to create a new frame or create a dialog. For more information, see Section 20.7.8, "How to Create a Frame".

To add a menu component to the UI:

1. Right-click the UI frame file in the Navigator and choose **Open**.
2. Select your main UI frame in the Java Visual Editor or in the Structure window.
3. Click a menu component on the Component Palette and drop it anywhere in the Java Visual Editor.

   You can choose either a menu bar or a popup menu.

   - A menu bar is attached to the main UI frame or dialog, and is displayed at the top of the application.
   - A popup menu is displayed when the user right-clicks in your UI. Popup menus do not have menu bars.

   Alternatively, you can open a file that already contains a menu component.

At this point, nothing is visible on the UI. The added menu component is displayed in the Structure window and opens in the Menu Editor.

20.9.5 How to Add a Popup Menu

Popup menus can be created so they open on a particular UI container. You add the popup menu to the container and create an event handler to specify the user's action that triggers the popup.

To add a popup menu:

1. Open your UI class in the Java Visual Editor.
2. Drop a popup menu from the AWT or Swing Containers Component Palette into the Structure window. The Menu Editor appears.
3. Add one or more menu items to the popup menu. For more information, see Section 20.9.8, "How to Add a Menu Item".
4. Expand the UI folder in the Structure window and select the panel or other component whose event you want the popup menu attached to so you can see that component in the Property Inspector. For the following example, `panel1` was selected.
5. In the Property Inspector, click the **Events** tab and click the desired event value field.
6. Type the stub name of the event into the event value field and press **Enter** to create an event-handling method stub in the source code with the supplied name. For the following example, the `MouseClicked` event was selected and the name `panel1_mouseClicked` entered.
7. Edit your event-handler stub to resemble the following:

Example 20–2 Event Handler Stub

```
void panel1_mouseClicked(java.awt.event.MouseEvent e) {
  panel1.add(popupMenu1);
   if (e.isPopupTrigger()) {
    // Make the PopupMenu visible relative to the current mouse
    // position in the container.
   opupMenu1.show(panel1, e.getX(), e.getY());
   }
}
```

8. Add event handlers to the popup menu's menu items as needed for your application.

20.9.6 How to Create a Submenu

Submenus can appear on menus to provide additional, related commands. Such nested lists are displayed with the menu text followed by an arrow. JDeveloper supports as many levels of submenus as you want to build into your menu.

Organizing your menu structure with submenus can save vertical screen space. However, for optimal design purposes you probably want to use no more than two or three menu levels in your UI design.

When you move a menu off the menu bar into another menu, it becomes a submenu. Similarly, if you move a menu into an existing submenu, it forms another submenu under the submenu.

To create a submenu:

1. Select the menu item to which you want to add a submenu.
2. Right-click and choose **Insert Submenu**.

   Alternatively, you can create a submenu by selecting the menu item and pressing **Ctrl** and the right arrow key.
3. Click the new submenu item and type a name for the nested menu item, or drag an existing menu item into this placeholder.
4. Press **Enter**, or the **Down** arrow, to create the next placeholder.
5. Repeat steps 3 and 4 for each item you want to create in the submenu.
6. Press Esc to return to the previous menu level.

20.9.7 Customizing Menus with the Menu Editor

Use the JDeveloper Menu Editor to customize and manage your menu items.

20.9.8 How to Add a Menu Item

When you first open the Menu Editor, it displays the menu bar or popup menu that you opened with any defined menu items. There is also a blank menu to the right of the last menu in the menubar and a placeholder at the end of each menu, indicated by a dotted rectangle.

To add menu items to an existing menu:

1. In the Menu Editor, select the position on the menu bar where you want to add a new menu, or on the menu select where you want to add a new menu item, separator or submenu.
2. Right-click and choose **Insert MenuItem**.
3. While the menu item appears highlighted in the Menu Editor, type the text for the new menu component's label.

   As you start to type, the highlighted dotted rectangle changes to a normal text edit field containing a cursor. The text field will scroll as you type to accommodate labels longer than the edit field.
4. When you're finished typing, press **Enter**.

   The width of the list expands if necessary to display all the labels in the list, and a placeholder for the next menu item is automatically selected.
5. Enter a label for each new item you want to create in the list, or press Esc to return to the menu bar.

   In addition to directly selecting items in the Menu Editor, you can use the arrow keys to move from the menu bar into a menu, and to move between items in the list; press Enter to complete an action.

20.9.9 How to Disable a Menu Item

You can prevent users from accessing certain menu commands based on current program conditions without removing the command from the menu. For example, if no text is currently selected in a document, the **Cut**, **Copy**, and `Delete` items on the Edit menu are disabled and are displayed dimmed.

Use the `enabled` property to disable a menu item. As with most properties, you can specify an initial value for enabled using the Inspector. The default enabled state of a menu item is True; this may change when an event occurs.

To disable a menu item:

1. Select the menu item in the Menu Editor or in the Structure window.
2. Right-click and choose **Enabled**.
3. Alternatively, in the Property Inspector, set the enabled property for the menu item to `false`.

   In contrast to the visible property, the enabled property leaves the item visible. A value of `false` dims the menu item.

20.9.10 How to Specify Accelerators

Accelerators enable the user to perform an action without accessing the menu directly by typing in an accelerator key combination. For example, a commonly used accelerator for **File** > **Save** is **Ctrl+S**.

To specify an accelerator for a menu item:

1. Select the menu item in the Design view, or in the Structure window.
2. In the Property Inspector window, select the `accelerator` property from the Model section, and choose `KeyStroke` from the dropdown menu. Use the accelerator dialog to supply the key combination.

20.9.11 How to Insert a Separator Bar

A separator bar inserts a line between menu items and between sibling menu components, including menu items and submenus. You can use separator bars to indicate groupings within the menus.

To insert a separator bar on a menu:

1. Select the menu item before which you want a separator, or choose the blank item at the end of a menu.
2. Right-click and choose **Insert Separator**.

   The separator bar is inserted above the selected menu item.
3. Alternatively, you can type a hyphen (-) for the menu item label.

   Using the right-click menu to insert the separator will result in the line addSeparator() to be generated, whereas using the hyphen for the label will use the more memory intensive creation of an additional class member whose label is a hyphen.

20.9.12 How to Create Checkable Menu Items

To make a menu item checkable, you need to change the menu item from a regular MenuItem component to a `CheckboxMenu`Item. A `CheckboxMenuItem` has a State property (boolean) that allows an event-handler to determine how the associated event, or behavior, should be executed.

- The `state` property for a checked menu item is set to `true`.
- The `state` property for an unchecked menu item is set to `false`.

To change a regular menu item to a CheckboxMenuItem:

1. Select the menu item.
2. Right-click and choose **Checkable**.

20.9.13 How to Insert and Delete Menus and Menu Items

To insert a new, blank menu or menu item, place the cursor on an existing menu item and right-click and choose **Insert** (Menu or Menu Item).

Menus are inserted to the left of the selected item on the menu bar, and menu items are inserted above the selected item in the menu.

To delete a menu item, select the menu item and press the **Delete** key.

> **Note:** A default placeholder (which you cannot delete) appears after the last menu on the menu bar and below the last item on a menu. This placeholder does not appear in your menu at runtime.

20.9.14 How to Move a Menu Item

In the Menu Editor, you can move menus and menu items by dragging and dropping them. When you move a menu item with submenu items, the submenu items move as well.

You can move menu items and submenu items:

- Within a menu
- To other menus

You move entire menus along the menu bar.

To move a menu item:

1. Click and drag the menu item or submenu item to the new location.

   If you are dragging the menu item to another menu, drag it along the menu bar until the cursor points to the new menu.This action causes the menu to open, enabling you to drag the item to its new location.
2. Drop the menu item or submenu item at the new location.

To move a menu:

1. Click menu label in the menu bar and drag to the new location across the menu bar until the cursor points to the location where you want the menu to appear.
2. Drop the menu at the new location.

20.10 Working with Event Handling

Use UI design tools in JDeveloper to attach event handler code to component and menu events.

In building your Java program, you can think of your code as being divided into two categories: initialization code and event-handling code.

- Initialization code is executed when the UI components are created. You can think of this primarily as "start up" code for the components. This initialization code includes anything in the **jbInit()** method that all JDeveloper-designed GUI classes

have. JDeveloper generates this code based on your UI design. For example, JDeveloper generates a `button1.setLabel`("OK") method call because you set the label property of a button, using the Inspector, to "OK".

- Event-handling code is the code that is executed when the user performs an action, such as pressing a button or using a menu item. JDeveloper creates the stub (empty) event-handling method for you when you enter an event name in the Inspector for that component and press Enter. In that stub, you write code to handle the actual action caused by the event.

Your entire program consists of the initialization code, which says how things should look when they first appear, and the event-handling code, which says what should happen in response to user input.

There are some JDeveloper components, such as dialogs, which normally appear only when event-handling code is executed. For example, a dialog isn't part of the UI surface you are designing in the Java Visual Editor; instead it is a separate piece of UI which appears transiently as a result of a user selecting a menu item or pressing a button. Therefore, some of the code associated with using the dialog, such as a call to its `show()` method, might be placed into an event-handling method.

20.10.1 How to Attach Event Handling Code to Menu Events

In Swing, a menu item has `actionPerformed` events and `CheckboxMenuItems` have `itemStateChanged` events. Code that you add to the `actionPerformed` event for a menu item is executed whenever the user chooses that menu item or uses its accelerator keys.

To add code to a menu item's event:

1. Open the Java Visual Editor for your UI frame.
2. Add a menubar to your UI frame and insert menus and menu items into the menubar. Alternatively, you can open a file that already contains a menu.
3. Select a menu item in the Menu Editor or the Structure window.
4. In the Property Inspector, click the **Events** tab and click the desired event value field.
5. Type the stub name of the event into the event value field and press Enter to create an event-handling method stub in the source code with the supplied name.

   When you enter a name in the event value field, JDeveloper open the Code Editor and displays the source code in the Structure window. The cursor is positioned in the body of the newly created event-handling method, ready for you to enter code.
6. Inside the open and close braces, enter the code you want to have executed when the user clicks this Menu command.

20.10.2 How to Attach Event-Handling Code to a Component Event

Using the Events page of the Inspector, you can attach handlers to component events and delete existing event handlers.

To attach event-handling code to a component event:

1. Select the component in the Java Visual Editor or in the Structure window.
2. In the Property Inspector, select the **Events** tab to display the Events for that component and click the desired event value field.

3. Type the stub name of the event into the event value field and press Enter to create an event-handling method stub in the source code with the supplied name.

   JDeveloper creates an event handler with the new name and switches to that event handler in the source code. JDeveloper also inserts some additional code into your class, called an Adapter, to make the connection from the event to your event handling method.

4. Inside the stub of the event handler write the code that specifies the response to that component event.

> **Note:** To find out what methods and events a component supports, right-click that component in the Code Editor and choose **Go to Declaration** to open the class in the Code Editor. You can also right-click the component and choose **Browse Javadoc** to view the documentation for that class.

To quickly create an event handler for a component's default event:

1. Select a component on the Component Palette and add it to your UI.
2. Double-click the component in the Java Visual Editor. An event stub is created and focus switches to that event handler in the source code.
3. Add the necessary code to the event handler to complete it.

> **Note:** The default event is defined by beanInfo, or as `actionPerformed` if none was specified.

20.11 Working with Applets

Use JDeveloper's UI design tools to create an applet class and applet HTML file. You can also convert any HTML files that contain applets to a format that can be used with the Java Plug-in.

20.11.1 How to Create an Applet

In JDeveloper, you can easily create a skeleton Java applet and then edit it with the Code Editor.

To create a Java applet:

1. In the Navigator, select the project in which you want to create the new applet.
2. Choose **File** > **New** to open the New Gallery.
3. In the **Categories** tree, expand **Web Tier** and select **Applet**.
4. In the **Items** list, double-click **Applet** to open the Create Applet dialog.

   This will open the Create Applet dialog that will create the applet for you based on information you specify, including the name, package and class it extends.

When you are finished, you will have a skeleton `.java` file containing the applet class, based on the details you entered. You can edit this file in the Code Editor. Using JDeveloper you will also be able to embed your applet within an HTML page, which you can create with the Applet HTML File wizard. You can also run the applet standalone in order to test it from JDeveloper.

20.11.2 How to Create an Applet HTML File

In JDeveloper, you can easily create a Java applet HTML file that acts as a container for your applet.

To create a Java applet HTML file:

1. In the Navigator, select the project that contains your applet.
2. Choose **File** > **New** to open the New Gallery.
3. In the **Categories** tree, expand **Client Tier** and select **Swing/AWT**.
4. In the Items list, double-click **Applet HTML Page** to open the Applet HTML File wizard.

The Applet HTML File wizard will create the file for you based on information you specify, including the file location, code attributes, positioning attributes, and applet parameters. You can also create an optional deployment profile.

20.11.3 How to Convert an HTML Page that Contains an Applet

JDeveloper includes the Java Plug-in HTML Converter; it can convert any HTML files that contain applets to a format that can be used with the Java Plug-in.

To convert HTML files that contain applets:

1. Choose **Tools** > **Convert HTML** to open the Java Plug-in HTML Converter.
2. Specify a file or the directory path of the files to be converted.
3. Specify matching file names if you entered a directory path, and optionally select **Include Subfolders**.
4. Change or accept the default folder for backup files.
5. Select the template file to use from the dropdown list.
6. Click **Convert**.

   This plug-in places copies of your original files in the backup folder and converts all the files you specified in their original location. Once you have run the converter, your files will be setup to use the Java Plug-in.
7. Click **Done** when the Progress window shows that all files have been processed.

20.11.4 Deploying Applets

Deploying your applet, or any other Java EE web modules in Oracle Application Server with JDeveloper is a completely automated process.

20.11.4.1 How to Configure an Applet for Deployment

A standalone applet is packaged as a web archive (WAR) file which contains the applet, the Applet HTML file, as well as the standard Java EE web deployment descriptor, web.xml and possibly target-specific deployment descriptors, as well. After you have created the deployment profile and the appropriate deployment descriptor files, you can deploy the application to an application server, or as an archive file.

To configure a web application for deployment:

1. Create a WAR file deployment profile for your project.

A profile may have already been created for your project. If you wish to deploy to multiple targets, create a separate profile for each.

2. Add a `web.xml` deployment descriptor to your project, if it is not already present.

   Normally, this file is created with the WAR file deployment profile.

Notes: If you encounter problems when deploying a Swing applet (JApplet), for example, the error "Class not found" is displayed, this may indicate that JDeveloper cannot locate the Swing libraries. Your clients may need to use Sun's Java SE browser plugin or bundle the Swing libraries for JVMs version 1.1 with your applet.

Deployed applet files must reside in a separate location from any other web application files you have deployed.

20.11.4.2 How to Deploy an Applet as a WAR File

You can deploy web application components including applets as a WAR or EAR file to the target application server.

To deploy an applet as a WAR file:

1. If not already done, configure the applet for deployment.
2. If not already done, create an application server connection.
3. In the Navigator, right-click the project and choose **Deploy** > ***deployment profile***.
4. In the Deploy dialog, select one of the deployment options:
   - **Deploy to *application server connection*** to create the archive type specified in the deployment profile, and deploy it to the application server connection you select or create on the Select Server page of the Deploy dialog.
   - **Deploy to EAR file** to deploy the project and any of its dependencies (specified in the deployment profile) to an EAR. JDeveloper puts the EAR file in the default directory specified in the deployment profile.
   - **Deploy to WAR file** to deploy the project to a WAR. JDeveloper puts the WAR file in the default directory specified in the deployment profile.

Notes: The deployed applet files must reside in a separate location from any other web application files you have deployed.

If you encounter problems when deploying a Swing applet (JApplet), for example, the error "Class not found" is displayed, this may indicate that JDeveloper cannot locate the Swing libraries. You may need to force your clients to use Sun's Java SE browser plugin or bundle the Swing libraries for JVMs version 1.1 with your applet.

20.12 Working with the UI Debugger

In addition to JDeveloper's standard Java and PL/SQL debugger facilities, JDeveloper also provides support for debugging graphical user interfaces (GUIs) specifically for AWT and Swing-based client applications and applets.

The UI Debugger offers an alternative way of debugging a GUI application. Traditional debuggers let you examine the data structure and track program flow. Instead, the UI Debugger lets you examine the GUI structure and the event sequences.

The UI debugger helps you to see the relationship between UI components displayed on the screen with the actual data. It will also show you the events that are fired by the UI components, and the listeners that receive the events.

To use the UI Debugger, you need to first download it by choosing **Help** > **Check for Updates** and following the instructions in the wizard. For more information on how to install an Oracle JDeveloper Extension, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

There are no additional special prerequisites for the using the UI Debugger beyond those requirements for using the JDeveloper debugger, other than ensuring that the JDeveloper Runtime library, `jdev-remote.jar`, is selected in the **Project Properties - Libraries** page.

Debugging a GUI application can be a challenge since most traditional debuggers do not let you easily examine the tree structure of a GUI application, nor do they display the details of what is displayed by your application.

To start debugging, select a project and choose **Run** > **UI Debug <*projectname*>.jpr** to start debugging.

20.12.1 Working with UI Debugger Windows

You can use the UI Debugger features which are exposed in JDeveloper via three dockable windows. The UI Tree and the UI Outline windows appear automatically when the UI Debugger is started. The Events window appears the first time you track events. You can toggle all three windows by choosing **View** > **UI Debugger - <*UI_debugger_window*>**.

> **Note:** No information is displayed in the UI Debugger windows until you take a snapshot. Click the **Snapshot (F5)** button to populate the UI Tree and the UI Outline windows.

- **UI Tree:** Displays a hierarchical structure of your application's components and sub-components and their parent-child relationships. Select a component from the tree and right-click to display the context menu options. You will notice that the component is also selected in the UI Outline window.
- **UI Outline:** Displays an image or outline image of the application's GUI. Select a component from the graphical representation of the GUI application and right-click to display the context menu options.

  > **Note:** Since AWT components may not be painted correctly, Oracle recommends that you work in Outline mode for non-Swing based applications.

- **Events:** Displays information about those events you've selected to listen to from the Listeners dialog. The Listener dialog displays when you choose the Events context menu option from either the UI Tree or UI Outline windows. When you select an event in this window, its source component is selected in the tree and outline windows.

20.12.2 How to Start the UI Debugger

Before performing any UI Debugger task, you'll need to first start the UI Debugger.

To start the UI debugger:

1. Select the project in the navigator that you want to debug.
2. Select a run configuration. For more information, see Section 19.3, "How to Configure a Project for Running."
3. Choose **Run** > **UI Debug** <*projectname*>.**jpr** to start the project's default target and to run the application.

   JDeveloper starts the UI Debugger. The application is launched and the UI Tree and UI Outline windows automatically appear. However, no information is displayed in the UI Debugger windows yet.
4. After the application is completely launched, go to the dialog or window you want to debug and select it.
5. From either UI Debugger windows, click the **Snapshot (F5)** button.

   JDeveloper displays a hierarchical structure of the application in the UI Tree window and displays a graphical representation of the application's user interface in the UI Outline window.

20.12.3 Examining the Application Component Hierarchy

The information in the UI Tree and the UI Outline windows and the relationship between them are always synchronized. Since the information in the UI Tree and the UI Outline windows is identical (only the way they are presented is different), whenever you select a component in the UI Tree hierarchy, JDeveloper locates and highlights the same object in the UI Outline window, and vice versa.

Before examining the application component hierarchy, you must start the UI Debugger and take a snapshot. Whenever the UI of the application is updated, you must click **Snapshot** again to update the information displayed by the UI Debugger windows.

Ways to examine the application component hierarchy:

- Use the tree of the UI Tree window to explore the hierarchical structure of the components or use the UI Outline window to locate the components visually.
- Use the **Image** and **Outline** checkboxes at the top of the UI Outline window to toggle respectively the image and the borders of the components.
- You can use the icons at the bottom of the UI Outline window to zoom in or zoom out of the application image. If the image is larger than the window, you can pan across by clicking and dragging the image.
- The components that are not selected in the UI Outline window are shaded red.
- Hidden components are represented by gray text in the UI Tree.
- You can right-click a component in either windows to display the context menu options. See UI Tree or UI Outline for more information.

20.12.4 How to Display Component Information in the Watches Window

To examine the data associated to a component, you can choose to watch the component in the JDeveloper Watches window. A *watch* enables you to monitor the changing values of variables or expressions as your program runs.

To display component information in the Watches window:

1. If not already done, start the UI Debugger and take a snapshot.
2. Right-click a component either in the UI Tree or the UI Outline window and choose **Watch** from the context menu.

   The Watches window opens as a tab in the Smart Data window (if it is not already open), and a tree representing the component's structure is displayed in it.

20.12.5 How to Inspect a UI Component in an Inspector Window

You can view the state of a UI component in a JDeveloper Inspector window.

To display a UI component in an Inspector window:

1. If not already done, start the UI Debugger and take a snapshot.
2. Right-click a component either in the UI Tree or the UI Outline window and choose **Inspect** from the context menu.

   The Inspect window opens as a tab in the Smart Data window (if it is not already open), and a tree representing the component's structure is displayed in it.

20.12.6 How to Trace Events Generated by Components

Use the event tracing feature to monitor the firing of selected events generated by UI components. Use this information to determine the content of events, and their sequence.

To trace events generated by components:

1. If not already done, start the UI Debugger and take a snapshot.
2. Right-click a component either in the UI Tree or the UI Outline window and choose Trace Events from the context menu.

   The Trace Events dialog opens, displaying a list of the listeners that receive the event types fired by the component.

   Note: Event listeners are listed only for UI components that were visible when the snapshot was taken. If subsequent execution have added or removed UI components, the change will not be seen in the list.

3. (Optional) Select **Include Children** to also show additional event types fired by the children of the selected component.
4. In the Listeners dialog, select which event listener(s) you want to trace. For example, if you select `FocusListener`, all focus events will be traced.
5. Click **OK**.
6. The events fired by the selected listeners are displayed in the Events window. Right-click in the window to **Clear** the contents of the window or to **Remove** a specific Listener.

20.12.7 How to Show Event Listeners

Use the show listeners feature to find the recipients of events fired by UI components. Use this information to determine the extent of UI events.

Caution: Event listeners are listed only for UI components that were visible when the snapshot was taken. If subsequent execution have added or removed UI components, the change will not be seen in the list.

To trace events generated by components:

1. If not already done, start the UI Debugger and take a snapshot.
2. Right-click a component either in the UI Tree or the UI Outline window and choose Show Listeners from the context menu.

   The Listeners dialog opens for the selected component, displaying a list of listener types informed by the component, the classes of the registered listeners for each listener type, and the event methods implemented by each class.

   Note: The debugger's tracing filter is applied to the listener's list. A listener whose class is excluded by the filter will not be shown.

3. Select a method.
4. Click **Go To Source**.

   An edit window opens, showing the source code for the selected method.

20.12.8 How to Remote Debug GUI Applications

JDeveloper supports remote debugging GUI applications via the command line. To achieve this, you must manually launch the program you want to debug. Once the program is launched and the JDeveloper debugger is attached to it, remote debugging is very similar to local debugging.

Performing remote UI debugging is similar to remote debugging any application. Just make sure that the following requirements are met first:

- Add the JDeveloper runtime, `jdev-remote.jar`, to the libraries
- Specify the UI Debugger agent's main class before your application's main class

To remote debug GUI applications:

1. Configure your project for debugging, making sure to enable it for remote debugging.
2. Start your application manually as follows by executing:

   ```
   java -XXdebug -cp ...\jdev\lib\jdev-remote.jar
   oracle.jdevimpl.runner.uidebug.debuggee.Debuggee <MainClass>
   ```

   where

   - `...\jdev\lib\jdev-remote.jar` is the JDeveloper Runtime Library classpath which you must add to the command.
   - `oracle.jdevimpl.runner.uidebug.debuggee.Debuggee` is the name of the main class of the UI Debugger's agent.

3. A message similar to the following is printed in the command window:

   ```
   *** Port is 4000 ***
   *** Waiting for JVM debugger connection. ***
   ```

4. The UI Debugger uses a socket to interact with your application. The default port number for the socket is 4030 but you can specify another port number by inserting `-uidport,<port>` before the application's main class as follows:

   ```
   java -XXdebug -cp ...\jdev\lib\jdev-remote.jar
   oracle.jdevimpl.runner.uidebug.debuggee.Debuggee -uidport,5678
   mypackage1.Application1
   ```

   In this case, you will also have to specify the port number when you start the UI Debugger in the JDeveloper IDE.

To Start JDeveloper IDE for Remote UI Debugging:

1. Select a run configuration that has been set up for remote debugging (**Run** > **Choose Active Run Configuration)**.
2. Choose **Debug,** then **UI Debug** <***project_name***>**.jpr**.

   The main method of your Java application is started.
3. The Attach to JPDA dialog appears, prompting you to specify a host name and a UI debugger port.

   Unless you have used the -uidport option, you should leave this value to the default 4030.
4. Your UI debugging session will now behave as if it were performing local UI debugging. You can begin performing any UI debugger task.

20.12.9 Automatic Discovery of Listeners

The list of events that can be tracked by the UI Debugger is not hard-coded but is dynamically discovered at runtime. It is therefore possible to track events fired by any listener, provided that they adhere to the following guidelines:

- The component class must have public methods to add and remove a listener.
- The name of the methods must start with add or remove and end with `Listener`.
- The return type must be `void`.
- The methods must have only one argument.
- The type of the argument must be an interface that extends `java.util.EventListener`.
- The name of the method must be equal to the name of the interface preceded by `add` or `remove`.
- The return type of each method in the specified interface must be `void`.
- The method can only have one argument (the event).
- The type of the argument must be a class accessible as a bean.
- The return values of the getters can be anything except void. If the type is a non-primitive type, the value that will be shown in the UI Debugger will be the string obtained by calling the object's `toString()` method.

Examples

- For example, if you want to define a new event listener of type `Xxx`, your component must have methods with the following signatures:

  ```
  public void addXxxListener(XxxListener);
  ```

```
public void removeXxxListener(XxxListener);
```

- An example of an XxxListener interface could be:

```
public interface XxxListener extends java.util.EventListener
{
   public void methodOne(XxxEvent xxxEvent);
   public void methodTwo(XxxEvent xxxEvent);
   public void methodThree(XxxEvent xxxEvent);
}
```

- An example of a XxxEvent class could be:

```
public class XxxEvent
{
   public int getA(){...}
   public String getB(){...}
   public OtherType getC(){...}
```

Part V

Developing Applications Using Modeling

This part describes how to use Oracle JDeveloper diagrams and related diagramming tools and technologies to model the various components of your application.

- Chapter 21, "Getting Started With Application Modeling Using Diagrams"

  This chapter introduces you to the modeling features.

- Chapter 22, "Creating, Using and Managing Diagrams"

  This chapter walks you through how to use and manage your diagrams.

- Chapter 23, "Developing Java EE and Java Applications Using Modeling"

  This chapter details the modeling features related to your Java EE application modeling.

21

Getting Started With Application Modeling Using Diagrams

This chapter describes how to get started modeling your applications, and various application sub-systems, and databases using the diagrams and related diagramming tools and technologies included in Oracle JDeveloper.

This chapter includes the following sections:

21.1 About Modeling with Diagrams

JDeveloper supports four standard UML diagrams types, and four additional diagram types to model and collaborate the software and systems development for your applications. You can use the diagrams in JDeveloper to model your typical business applications, including custom applications.

21.2 Diagram Types

JDeveloper provides New Gallery wizards to create the diagrams for your modeling projects. You can easily create any of these diagrams and related UML and non-uml components using the New Gallery wizards. Select **File > New** > **General** > **Diagrams** from the Menu bar.

Select your diagram type then double-click, or click **OK** to start the wizard. The wizard lets you choose the package for your new diagram, as well as select the optional components you want available for that diagram. The component selection feature enables you to choose the diagram-related components you want to show up in the Component Palette while you are editing your diagram. Figure Figure 21–1 shows an example create diagram dialog for a class diagram.

Choose from a variety of different tools, elements, and UML-compliant objects to model your application systems on the diagrams, for example, those available in the Component Palette for a typical class diagram, as shown in Figure 21–2.

Figure 21–1 Create Class Diagram Example

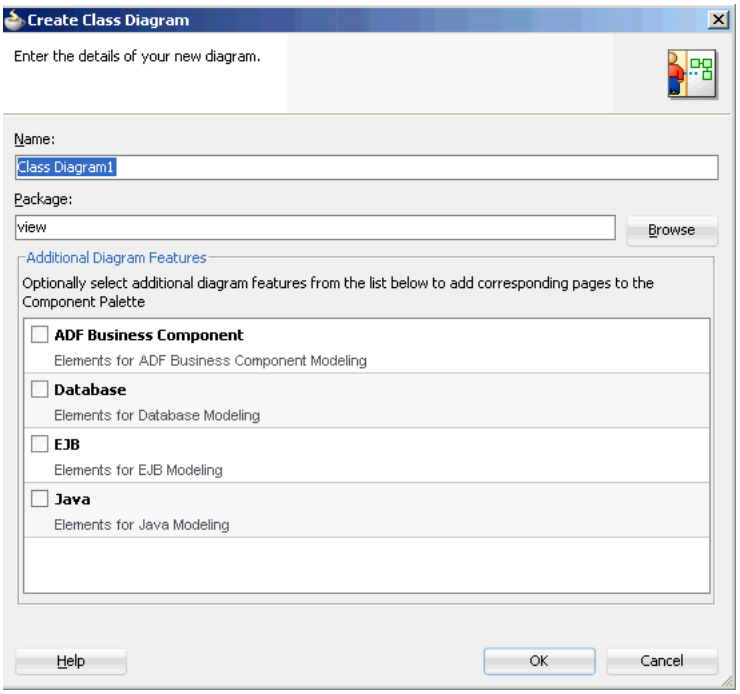
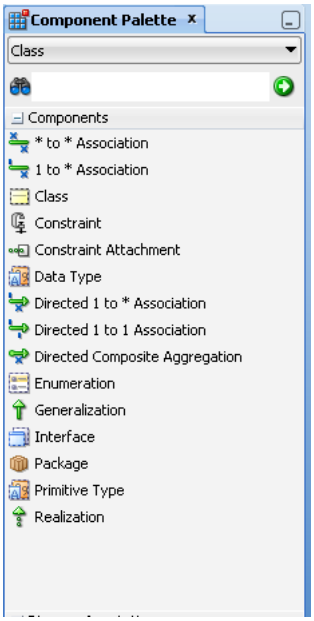


Figure 21–2 Class Diagram Component Palette



21.2.1 UML Diagrams

JDeveloper offers four standard UML diagram types to model your Java classes. You can use all of the standard UML objects, and class and diagram transformation features for UML to easily transform classes to online and offline database objects, or vice-versa. For more information, see Chapter 22.7.1, "How to Transform UML and Offline Databases".

Available UML diagram types include the following:

- **Activity Diagram**. Model system behavior as coordinated actions. You can use activities to model business processes, such as tasks that achieve specific business goals, like shipping, or order processing.
- **Class Diagram**. Model the structure of your system. Create new or inspect the architecture of existing class models, interfaces, attributes, operations, associations, generalizations and interface realizations.
- **Sequence Diagram**. Model sequence of event occurrences. Sequence diagrams are organized according to time, and show the calls between the different objects in the sequence. Use the diagram to create or inspect interactions between events, lifelines, messages and combined fragments.
- **Use Case Diagram**. Visually model what a system is supposed to do. A use case diagram is a collection of actors, use cases, and their communications.

21.2.2 Business Services Diagrams

You can model your business services and entities and their relationships in your applications using both regular and UML objects. Using JDeveloper transformation features for UML objects, you can transform online or offline database tables to classes, or classes to database tables. All of these objects can be modeled on any of the following four diagrams:

- **Business Components Diagram.** Diagram your business component interrelationships, entity classes, interactions, and public interfaces.
- **Database Diagram**. Model your online and offline database tables and their relationships. Use transformation features to create a diagram model that represents your database schema, or transform your classes to database tables online or offline.
- **EJB Diagram**. Create, edit, and model the entity objects, session and message-driven beans inside a system, and the relationships between them.
- **Java Class Diagram**. Model the relationships and the dependencies between Java classes. Use Java Class Diagrams to visually create or inspect objects like interfaces, enums, fields, methods, references, inheritance relationships, and implementation relationships.

21.3 How to Set Paths for a Modeling Project

You can configure the settings of a JDeveloper project to specify the root locations of the package hierarchies for modeled elements available to that project. The model path is configured by default. Change it if you want to include model element files to your model that are stored somewhere else, or to store new model element files somewhere else.

Modeled elements can be shared between projects by adding their file system location to the model path for a project.

To set the model path for a project:

1. Right-click the project whose model path you want to specify.
2. Choose **Project Properties**.
3. Open the **Project Source Paths** node and select the **Modelers** node.

4. In the **Model Path** area, use **Add** to enter the file system location for your project's model elements. Note that the order in which file system locations are entered in the **Model Path** signifies the order in which the folders are searched for a model element. These folders are the 'roots' of the package hierarchies used by a model. The first location specified in the model path is also the location in which new model elements are stored.
5. To complete the setting of the model path, click **OK.**

22

Creating, Using and Managing Diagrams

This chapter describes how to create and manage diagrams using the latest tools and technologies included in Oracle JDeveloper.

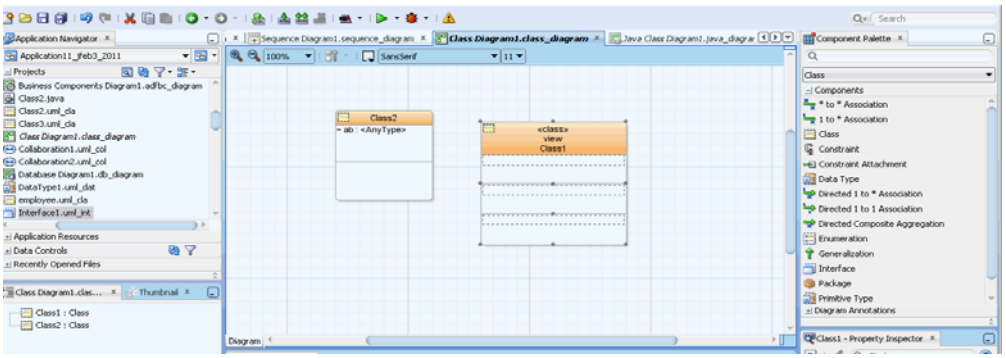
This chapter includes the following sections:

22.1 About Creating, Using, and Managing Diagrams

Oracle JDeveloper provides you with a wide range of tools and diagram choices to model your application systems. There are handy wizards to walk you through creating your diagrams and elements, as well as a Component Palette and Property Inspector to make it easy to drag and drop, and to edit a variety of elements without leaving your editing window. In addition there are four UML diagram options, including features to create, manage and transform your classes to UML and UML to classes, as well as XMI import and export capabilities.

Figure 22–1 shows the diagram editor window, with a class diagram, as well as the Application Navigator and Component Palette. You can open diagrams by double-clicking them in the Navigator, and once open, drop-and-drag components onto the diagram editor. Notice the standard formatting icons at the top of the editor, such as color, font, and zooming features.

Figure 22–1 Class Diagram Showing Objects in Navigator and Available Components



22.2 How to Use the Basic Diagramming Commands

You can perform many of the basic diagramming tasks and commands in a few clicks using JDeveloper menu options. Diagram files contain graphical properties such as positions, sizes and colors. The visual elements are generally stored in separate files. Changes you make to the diagram updates all related files. To ensure these are consistent, select **File** > **Save All**.

To create a new diagram:

1. In the Application Navigator, select your project, then choose **File > New** > **General** > **Diagrams**.
2. Select a diagram type, lick **OK**.
3. You might need to change the default name and package for the diagram. The default package for a diagram is the default package specified in the project settings. An empty diagram is created in the specified package in the current project, and opened in the content area. Click **OK**.

To publish a diagram as a image:

1. Right-click on the diagram that you want to publish as an image, then choose **Publish Diagram**.

   Or

   Click on the surface of the diagram that you want to publish as an image, then choose **Diagram > Publish Diagram**.
2. Using the location drop-down list, select the destination folder for the image file.
3. In the File name box, enter a name for the image file, including the appropriate file extension (.svg, .svgz, .jpg, or .png).
4. From the File type drop-down list, select the file type for the image file (SVG, SVGZ, JPEG, or PNG).
5. Click **Save**.

To rename a diagram:

1. In the Application Navigator, select the diagram to rename.
2. Choose **File > Rename**.
3. Choose **Rename the file only do not update references**. References updates are not applicable to diagrams in this case.

To set up the page before printing:

1. Click on the surface of the diagram you want to print, then choose **File > Page Setup**.
2. Make changes to the settings on the tabs of the Page Setup dialog.

To set the area of the diagram to print:

1. Choose **File > Print Area > Set Print Area**.
2. On the diagram, drag the mouse pointer to enclose the objects on the diagram to print. The area to print is shown with a dashed outline. If you do not set an area, then the whole diagram is printed.

To clear a previously set print area:

- Choose **File > Print Area > Clear Print Area**.

To see a preview of the page before printing:

- Choose **File > Print Preview**.

To delete a diagram:

1. In the navigator, select the diagram to remove.
2. Choose **Edit > Delete.** These commands remove the diagram file from the system and close the editing window for that diagram. The elements for the deleted diagram remain in the navigator and on the file system.

   You can also delete a diagram from the Application Navigator. In the Application Navigator, right-click on the diagram name and choose **Delete**.

To Zoom In and Out of a Diagram:

Use **Ctrl+scroll** to zoom in and out of diagrams. When using the thumbnail view, use **scroll** to zoom.

To display the diagram at original size:

In the zoom drop-down list, located on the diagram toolbar, choose **100%**, or click the diagram, then choose **Diagram** > **Zoom** > **100%**.

To display the entire diagram:

In the zoom drop-down list, located on diagram toolbar, choose **Fit to Window,** or click the diagram, then choose **Diagram > Zoom > Fit to Window**.

To display the selected elements at the maximum size:

In the zoom drop-down list, located on the diagram toolbar, choose **Zoom to Selected**, or click the diagram, then choose **Diagram > Zoom > Zoom to Selected**.

22.3 Working with Diagram Nodes and Elements

Use nodes and elements to represent the various elements and related resources of your system architecture.

22.3.1 How to Work with Nodes

A node is shape on a diagram.

To create a node on a diagram:

1. Select the node type you want to create from those listed in the Component Palette for your diagram.
2. Click the diagram where you want to create the node. This adds the node at its default size.

   Or

   Click and hold down the mouse button where you want to place one of the corners of the node and drag the node outline to the opposite node corner and release the mouse button.
3. Enter the name for the node when the default element name is highlighted on the new node.

   Or

   Complete the wizard.

> **Note:** To add properties to nodes on a diagram, double-click the node or right-click the node and choose **Properties**. Then add the properties using a dialog or editor. You can also create the property on the diagram using in-place creation.

Nodes for certain elements can also be created inside other nodes.

To Create Internal Nodes on a Diagram Element:

Elements can be represented on a diagram as internal nodes on other diagram elements.

Internal nodes can be used to create the following:

- Inner classes and inner interfaces in modeled UML classes and interface.
- Inner classes and inner interfaces in modeled Java classes and interfaces.
- Relation usages in modeled database views.

Figure 22–2 Symbolic Diagram Class View

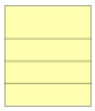
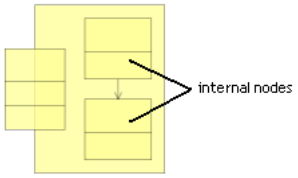


Figure 22–3 Expanded Diagram Class View Showing Internal Nodes



To create an internal node on a diagram element:

1. Select the node on the diagram to create an internal node.
2. Choose **Diagram > View As Expanded** to display an expanded view of the node.
3. Create the node for the internal node inside the expanded box, or drag the appropriate node from the navigator, or diagram, and drop it in the expanded node to create an inner node.

To change the way nodes are shown on a diagram:

Select the diagram element(s) and choose one of the following:

- **Diagram > View As > Compact**.
- **Diagram > View As > Symbolic**.
- **Diagram > View As > Expanded**.

To connect two nodes on a diagram:

Relationships between node elements are identified using connectors. A connector is the end point of a diagram edge.

1. Select the connector type from those listed in the Component Palette for the type element you are connecting.
2. Click the originating end node. Connectors with an implied or actual direction will traverse from the originating to the destination end.
3. Click the node to be the destination end of the connector.

To optimize the size of nodes on a diagram:

1. Select the nodes to resize.
2. Right-click the selected nodes then choose **Optimize Shape Size > Height and Width**, as one option.

22.3.2 How to Work with Diagram Elements

There are many options available for changing the properties on your diagrams and managing the elements.

To select all elements of the same type:

1. Select an object of the type you want.
2. From the context menu, choose **Select All This Type**.

To select all elements on the active diagram:

Choose **Edit** > **Select All**.

To select specific diagram elements on the active diagram:

Press and hold down the Ctrl key, then click the elements on the diagram to select.

To select all elements in a given area of the active diagram:

1. Position the pointer at the corner of the area on the diagram to select the elements, then press and hold down the mouse button.
2. Drag the mouse pointer over the area.
3. Release the mouse button when the objects are entirely enclosed within the selection area.

To deselect a selected element in a group of selected elements:

1. Press and hold down the Ctrl key.
2. Click the element(s) on the diagram to deselect.

To group elements on a diagram:

1. In the Component Palette, click **Group**.
2. Position the pointer at the corner of the area on the diagram to group the elements, then press and hold down the mouse button.
3. Drag the mouse pointer over the area.
4. Release the mouse button when the objects are entirely enclosed.

To manage grouped elements on a diagram:

Use the Manage Group feature to move elements in and out of groups, move elements to other groups, or move groups in and out of other groups.

1. Select the group to manage.
2. Right-click and select **Manage Group**.

   You can also move elements in and out of groups by Shift+dragging the element to the desired position.

To display related elements on a diagram:

Select the diagram item, then choose **Diagram > Show Related Elements**.

Or

Right-click the diagram item, then choose **Show Related Elements**.

To change the properties of a diagram element using the Properties dialog:

Open the Properties dialog in one of the following ways:

- On the diagram, double-click the element

  Or

- Select the element on the diagram, then, from its context menu, choose **Properties**.

To change certain properties of a diagram using in-place creation and editing:

You can create new secondary elements directly and change properties in place on modeled diagram elements.

To change the properties of diagram elements using the Property Inspector:

1. Open the Property Inspector by selecting **View > Property Inspector**.
2. In a navigator, select the element (s).
3. In the Property Inspector, find the property value to change.
4. On the right of the Property Inspector, select the control and change the value. (The control may be an edit box, a drop-down list, a checkbox, etc.).

> **Note:** These options are not all valid for all elements.

You can toggle the diagram display between symbolic mode, where all elements are displayed, and compact mode, where only basic information about the element is displayed.

To find an element on a diagram:

Click on the element name in the structure pane. The element is selected in the diagram. You can also use the thumbnail view of the diagram to find an element. To display a thumbnail view of a diagram, select the diagram either in the navigator or by clicking on the background of the diagram, then choose **View > Thumbnai**l. You can grab the view area box and move it over elements on the thumbnail view of the diagram. The corresponding elements are brought into view on the main diagram.

To change the color or font of one or more elements that are already on a diagram:

1. Select the element or elements on the diagram.
2. Then in the property inspector **(View >Property Inspector**), on the Graphical Options tab, select the current color (or the box for font type), make the required change(s), then press Enter.

   Or

   On the tool bar, select the font type, font size, or color box, then make the required change.

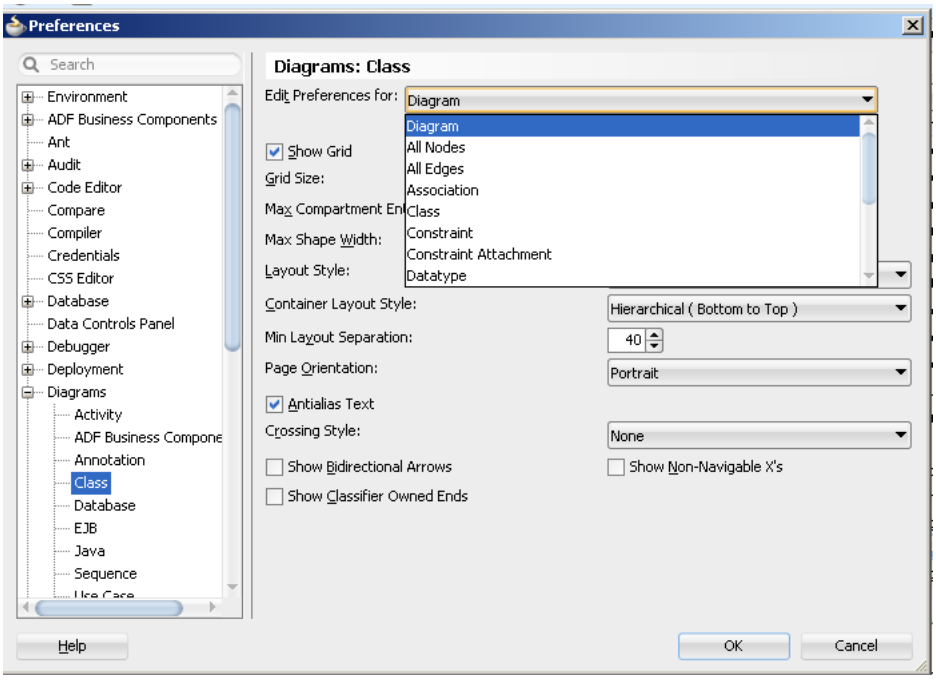
   Or

   Choose **Visual Properties** from the context menu, then make the required change.

To change the color or font of diagram elements to be added to a diagram:

1. Choose **Tools > Preferences**, select **Diagrams**, select the diagram type, and then (from the Edit Preferences For drop-down box), select the element type to change, as shown in Figure 22–4.
2. On the Color/Font tab, make the required changes.

Figure 22–4 Diagram Preferences Dialog



To copy the diagram graphical options of fill color, font, font color, and line color to one or more elements:

1. Select the element with the properties you want to copy.
2. Right-click and select **Visual Properties**. (You can also go to **Tools > Preferences > Diagrams** to customize all of the visual and graphical properties.)
3. Select one or more elements with the properties to change. (Press **Shift+select** to select a group of elements.)
4. Right-click and select **Paste Visual Properties**.

To copy elements from a diagram and paste them into another diagram:

1. Select the diagram elements, then choose **Copy** on the context menu, or choose the Copy icon on the toolbar, or press Ctrl-C.
2. Open the destination diagram.
3. Place the pointer where you want the diagram elements to be added, then choose Paste from the context menu (or choose the Paste icon on the toolbar, or press Ctrl-V.

To copy elements from a diagram and paste them into another application:

1. Select the diagram elements, then choose Copy on the context menu, or choose the Copy icon on the JDeveloper toolbar, or press Ctrl-C.
2. Open the destination application.
3. Use the clipboard paste facility in the destination application to place the diagram elements where you require them.

22.3.2.1 How to Resize and Move Diagram Elements

Resize an element by dragging the grab bars until the item is the size you want. Some diagram elements cannot be resized, such as initial and final pseudo states.

Certain element types also have internal grab bars, that are displayed when an element is selected. These internal grab bars are for resizing the compartments of those diagram elements.

Whenever a diagram element is resized or moved towards the visible edge of the diagram, the diagram automatically scrolled. New diagram pages are added where an element is resized or moved off the diagram surface.

Dragging selected diagram elements on the diagram surface is the easiest way of moving elements incrementally on a diagram. To move elements over a larger diagrams, cut and paste them using the clipboard.

To resize a diagram element:

1. Select the element to resize.
2. Position the pointer on any grab bar on the element and hold down the mouse button. The pointer is displayed as a double-headed arrow when it is over a grab bar.
3. Drag the grab bar until the element is resized, then release the mouse button.

To move diagram elements:

1. Select the element, or elements to move.
2. Position the pointer on the elements, then press and hold down the mouse button.
3. Drag the selected elements to their new position.
4. Release the mouse button. If an element overlaps another element they are displayed on top of one another. Right-clicking the element and choose Bring to Front to view.

22.3.2.2 How to Delete Diagram Elements

A diagram element can be removed from the current diagram without removing the file, or files where that element is stored. It can also be removed from the file system.

Diagram nodes and inner elements are removed the following ways:

- Nodes (excluding nodes inside an expanded node) are the only elements on diagrams that can be removed using **File** > **Cut**.
- Inner UML and Java classes and interfaces can be deleted using Erase from Disk. (If dragged outside the parent element they become primary nodes and can be removed using **File** > **Cut**.
- Other inner nodes (view object instances, entity object usages and application module instances) cannot exist outside of their parent elements. They can be deleted by selecting using **Edit** > **Cut**.

To remove an element from a diagram:

1. Select the element.
2. Choose **Edit > Delete** or press the **Delete** key. Using delete on elements such as associations and attributes completely removes them from the system.

To bring elements to the front or back of a diagram:

Right-click the diagram element and choose **Bring to Front** or **Send to Back.**

22.3.2.3 How to Undo the Last Action on a Diagram

You can undo and redo your most recent graphical actions. Graphical actions change the appearance of elements on the diagram surface and include the following:

- Cutting and pasting elements on class diagrams.
- Altering the position and size of diagram elements.
- Changing the font, color, and line width of diagram elements.

  Any change to a diagram element that is not graphical, such as changing the properties of that element, cannot be undone. Changing the properties of an element also prevents any previous graphical changes from being undone.

To undo the last graphical action on a diagram:

Choose **Edit >Undo [...]** or click the undo icon.

To redo a previously undone graphical action on a diagram:

Choose **Edit > Redo [...]** or click the redo icon.

22.3.2.4 How To Create UML Elements Independently of a Diagram

You can create UML elements without having to create any type of UML diagram in the New Gallery. The UML elements that you create in the New Gallery are listed in the navigator and can be dropped onto your diagrams.

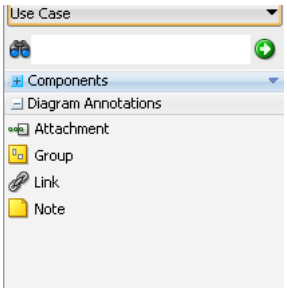
To create a UML element using the New Gallery:

1. Select the project in the navigator.
2. Select **File > New**. The New Gallery opens.
3. In the Categories panel, open the General node and select the UML node. The UML elements are listed in the Items panel.
4. In the Items panel, select the UML element to create, then click **OK**. The properties dialog opens for the selected UML element.
5. Complete the properties dialog, then click OK. The UML element is added to the navigator. If the UML element has a use case template associated with it, a use case form is opened in the default editor.

22.4 How to Work with Diagram Annotations

A note or annotation is a graphical object on a diagram containing textual information. Notes are used for adding comments to a diagram or the elements on a diagram. A note can be attached to one or more elements. A note is stored as part of the current diagram, not as a separate file system element. Note options are available in the Component Palette, as shown in Figure 22–5.

Figure 22–5 Annotations in the Component Palette



To add a note to a diagram:

1. Click the Note icon in the Diagram Annotations section of the Component Palette.
2. To create the note at the default size, click the diagram to create the note.

   Or

   To create the note at a different size, click the diagram, drag the note box to the desired size, and release the mouse button.
3. Enter the text for the note, then click the diagram surface.

To attach a note to an element on a diagram:

1. Click the Attachment icon in the Diagram Annotations section of the Component Palette.
2. Click the note.
3. Click the element that you want to attach the note to.

To change the font size, color, bolding, or italics on an element:

1. Click the note element. The text editing box appears.
2. Select the text to edit.
3. Select your text format.

22.5 Changing the Way a Diagram is Viewed

Change the way your diagram is shown by right-clicking an element, or using the many options available under **Tools > Preferences** for diagrams.

22.5.1 How to Hide, Show, and Layout Connectors on Diagram

Choose to hide a single connector or any number of connectors on your diagrams. Connectors that are hidden on a diagram continue to show in the Structure window, with 'hidden' appended. If there are any hidden connectors, you can bring them back into view individually or all at once.

To hide one or more connectors on a diagram:

1. Select the diagram connector or connectors to hide. (To select all connectors of a particular type, right-click a connector, then choose Select All This Type.)
2. Right-click and choose **Hide Selected Shapes**.

You can also go to the Structure window, select the connector or connectors to hide, right-click and choose **Hide Shapes.**

To show one or more connectors that are hidden on a diagram:

In the Structure window, select the connector or connectors to show, right-click and choose **Show Hidden Shapes**.

To show all hidden connectors on a diagram:

Right-click any part of the diagram and choose **Show All Hidden Edges** .

To list all hidden connectors together in the Structure window:

Right-click an object listed in the Structure window and choose **Order By Visibility.**

22.5.1.1 How to Show and Hide Page Breaks

You can display or hide page breaks on your diagrams. Page breaks are displayed as dashed lines on the diagram surface.

To display page breaks on new diagrams:

1. Choose **Tools > Preferences.**
2. Click **Diagrams** in the left pane of the dialog.
3. Click the diagram type in the left pane of the dialog. Note that class diagrams do not have pages.
4. Select the Show Page Breaks checkbox to display page breaks on new diagrams.
5. Click **OK**.

22.5.1.2 How to Lay Out Connectors on a Diagram

Connectors are laid out in oblique or rectilinear line styles.

Oblique lines can be repositioned at any angle. Rectilinear lines are always shown as a series of right angles. If the line style for a diagram is set to oblique, you can subsequently move the line (or portions of it) into a new position at any angle.

You can set the default line style for each type of connector element on a diagram. ("Line style" is one of the diagram preferences that you can set for each type of element that is a connector.)

You can also set the line style for a particular diagram, overriding the default line style for the diagram type:

- If you use this method to change the line style from oblique to rectilinear, lines already on the diagram that were drawn diagonally will be redrawn at right angles.
- If you use this method to change the line style from rectilinear to oblique, no change will be apparent on the diagram, but you will subsequently be able to move any of the lines on the diagram into a new position at any angle.

Whatever the line style for a drawing, you can select individual lines on the drawing and change their line style. If you change an individual line from oblique to rectilinear, the line will be redrawn using right angles. If you change an individual line from rectilinear to oblique, no change will be made to the line, but you can subsequently reposition it (or portions of it) at any angle.

There is an option to straighten all lines that are already on a diagram. If you choose this option, all lines will be redrawn along the shortest route between their start and end points, using diagonal lines if necessary. Using this option will also set the line style for the current diagram to oblique. After this, setting the line style for the diagram back to rectilinear will redraw all the lines at right angles.

You can also choose the crossing styles for your lines to be bridge or tunnel style. Selecting bridge style creates two parallel lines where the lines intersect. Selecting tunnel style creates a semi-circle shape on the intersection. The default style is a regular crossing over of the two lines where the lines intersect.

To set the default line style for a diagram (or for an element type that is a connector):

1. Select **Tools >Preferences,** then open the Diagrams node and choose the diagram type.
2. From the Edit Preferences For drop-down box, choose All Edges. If, instead of choosing All Edges, you choose a particular element type that is a connector, the default line style will be set only for that element type.
3. In the Display Options section, click the current line style. The current line style name becomes a drop-down box.
4. Select the required line style (either Oblique or Rectilinear) from the drop-down box.

To change the line style for particular connectors on the current diagram:

With the connector or connectors selected on the diagram, choose **Diagram > Line Style,** then either **Oblique** or **Rectilinear**.

To add a new elbow to a connector:

Shift+click on the connector where you want to create a new elbow. (New elbows can be added and used to change the route of a connector only if the line style is set to oblique.)

To remove an elbow from a connector:

Shift+click the elbow that you want to remove.

To straighten connectors:

Select the connector or connectors that you want to straighten on a diagram, then choose **Diagram > Straighten Lines.** (This will remove all intermediate elbows.)

To change the crossing style of all lines:

Select **Tools > Preferences > Diagrams > Crossing Style**. You can also change the crossing style in the Property Inspector for that diagram. The default style is a regular crossing over of the two lines where the lines intersect.

22.6 Laying out Diagrams

There are a number of ways to change the layout of a diagram. These include applying a layout style to any or all of the elements on a diagram, changing the height of nodes on a diagram so that they display all the properties of those elements, and aligning and distributing elements.

Diagrams can be laid out according to one of the predefined styles: Hierarchical, Symmetrical, Grid, and Row.

22.6.1 How to Use Diagram Layout Styles

You can lay out diagrams, or just selected diagram elements according to one of several layout styles, depending on your requirements: hierarchical (top to bottom, bottom to top, left to right, right to left), symmetric, grid, orthogonal, and row.

22.6.1.1 Hierarchical UML Diagram Layout

This lays out the diagram elements in hierarchies based on generalization structures and other connectors with a defined direction. Connectors between the nodes are laid out using the most direct route. Nodes on a diagram that are not connected to any other nodes are laid out in a grid layout. Hierarchical layout is available in four orientations: top to bottom, bottom to top, left to right and right to left.

Figure 22–6 Hierarchical UML Diagram Layout

22.6.1.2 Symmetrical Diagram

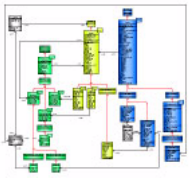
This lays out the diagram elements in the most symmetrical way based on the connectors between the nodes. Under certain circumstances, a symmetrical layout will position nodes in a radial layout around a central node. Nodes on a diagram that are not connected to any other nodes are laid out in a grid layout.

Figure 22–7 Symmetrical Diagram Layout

22.6.1.3 Orthogonal UML Layout

Orthogonal diagrams show hierarchical and non-hierarchical elements where the aligned edges of a component all follow the same direction. For UML class diagrams, a layout is created which represents each generalization hierarchy in an aligned fashion.

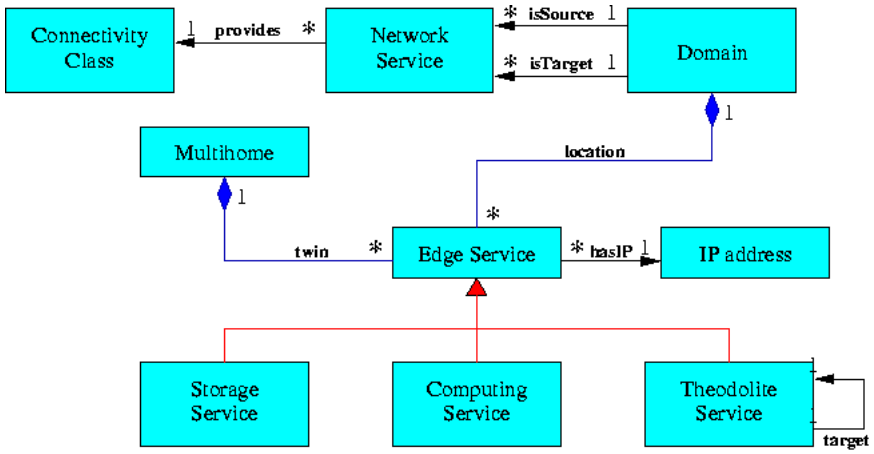
Figure 22–8 Orthogonal UML Diagram Layout



22.6.1.4 Grid Diagram

This lays out the diagram elements in a grid pattern with nodes laid out in straight lines either in rows from left to right, or in columns from top to bottom. Nodes are laid out in the grid pattern starting with the top left node.

Figure 22–9 Grid Diagram Layout



22.6.1.5 How to Use the Diagram Grid to Lay Out Diagrams

Diagram elements that are created or moved on a diagram can be automatically 'snapped' to the grid lines that are nearest to them, even if the grid is not displayed on the diagram. Grid cells on the diagram are square, so only one value is required to change both the height and width of the grid cells. By default, elements are not snapped to the grid on activity diagrams.

To define diagram grid display and behavior for the current diagram:

1. Click the surface of the diagram for which you want to define diagram grid display and behavior.
2. In the Property Inspector (**View > Property Inspector**), select the current value of the property that you want to change, then enter or select the new value for that property.

To define diagram grid display and behavior for new diagrams:

1. Choose **Tools > Preferences**, select **Diagrams** then the required diagram type.
2. Select from the following options:
   - Select the **Show Grid** checkbox to display the grid.
   - Select the **Snap to Grid** checkbox to snap elements to the grid. The grid does not have to be displayed for elements to be snapped to it.
   - Enter the grid size in the **Grid Size** field.
3. Click **OK**.

22.6.2 How to Align and Distribute Diagram Elements

You can align elements both vertically and horizontally on your diagrams. You can also change the location of elements so that they have equal vertical and horizontal spacing.

When you are distributing elements, the outermost selected elements on the vertical and horizontal axes are used as the boundaries. To fine tune the distribution, move the outermost elements in the selection, then redistribute the element.

To align and size elements on a diagram:

1. Select two or more elements in the diagram. Choose **Diagram > Align**.
2. Choose from the following:
   - Select the horizontal alignment.
   - Select the vertical alignment.
3. Use the Size Adjustments checkboxes to set the size of the selected elements:
   - Select the **Same Width** checkbox if you want all the selected elements to have the same width. The new element width is the average width of the selected element.
   - Select the **Same Height** checkbox if you want all the selected elements to have the same height. The new element height is the average height of the selected elements.
4. Click **OK**.

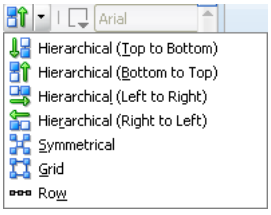
To distribute elements on a diagram:

1. Select three or more diagram elements and choose **Diagram > Distribute**.
2. Select the distribution for the elements.
   - Select the horizontal distribution: None, Left, Center, Spacing, or Right.
   - Select the vertical distribution: None, Top, Center, Spacing, or Bottom.
3. Click **OK.**

22.6.3 How to Layout Diagram Elements

Layout styles are available by opening the context menu for a diagram and choosing Lay Out Shapes, or by using the Diagram Layout Options Dropdown as shown in Figure 22–10,

Figure 22–10 Diagram Layout Options Dropdown



To layout elements on a diagram:

1. Choose one of the following:
   - Select the individual elements on the diagram.
   - Click the surface of the diagram to layout all the elements on a diagram.
   - Select the container element to layout all the elements within a container element.

2. On the diagram tool bar, choose the required layout style from the dropdown list, shown in Figure 22–10.

After the selected elements have been laid out they remain selected to be moved together to any position on the diagram.

To set the layout for new elements on the current diagram:

In the Property Inspector (**View > Property Inspector**), select the layout style.

To set the default layout of elements on a diagram:

1. **Choose Tools > Preferences > Diagrams**, then select the diagram type.
2. Select your layout style and click **OK**.

22.7 Transforming Java Classes and Interfaces

JDeveloper supports UML transformations on your Java classes and interfaces as well as XMI import and export features. You can use these transformation features to produce a platform-specific model like Java from a platform-independent model like UML. You can perform transformations multiple times using the same source and target models. There are also some reverse transforms to UML from Java classes. You can do transformations on individual or groups of UML objects, classes, and interfaces in the following ways:

- Transformation on the same diagram as the original.
- Transformation on a new diagram created for the transformed element.
- Transformation only in the current project, and not visually on a diagram.

To transform UML, Java classes, or interfaces:

1. Select the UML objects, Java classes, or interfaces to transform.
2. Do one of the following:
   - To create only the definitions of the transformed elements, but not add them to a diagram, choose **Diagram > Transform > Model Only.**
   - To create the transformed elements on the current diagram choose **Diagram > Transform > Same Diagram**.
   - To create a new diagram for the transformed elements, and display the transformed elements on it, choose **Diagram > Transform > New Diagram**.
   - Select the transformation you want to perform on the selected elements, then click **OK**, as show in Figure 22–11 and Figure 22–12.

Figure 22–11 Transform Dialog Showing UML Options

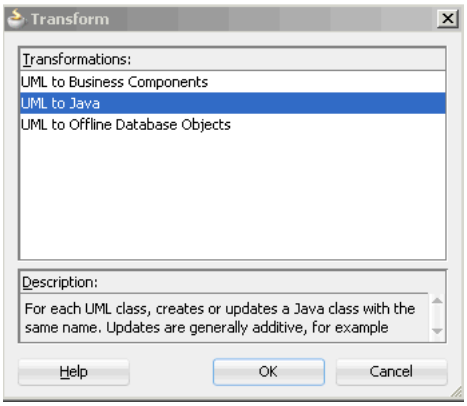
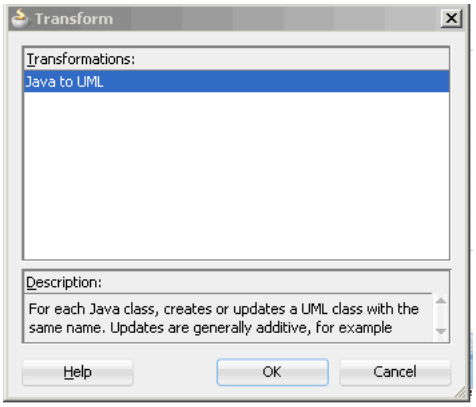


Figure 22–12 Transform Dialog Showing Java to UML Options



22.7.1 How to Transform UML and Offline Databases

You can use the UML modeling tools to create a UML Class model, and to then transform it to an offline database or vice-versa.

To transform a UML class diagram to an offline database:

1. Create or open the diagram to transform.
2. Select the class or classes to transform. Right-click and choose **Transform.**
3. Choose from one of the following:
   - **Model Only**. The offline database model based on the UML Class model is created in the Application Navigator in the current project.
   - **Same Diagram**. The offline database model based on the UML Class diagram is created. The offline database model can be viewed in:
     - The Application Navigator.
     - The UML Class diagram, as modeled tables.
   - **New Diagram**. The offline database model based on the UML Class diagram is created. The offline database model can be viewed in both:
     - The Application Navigator.

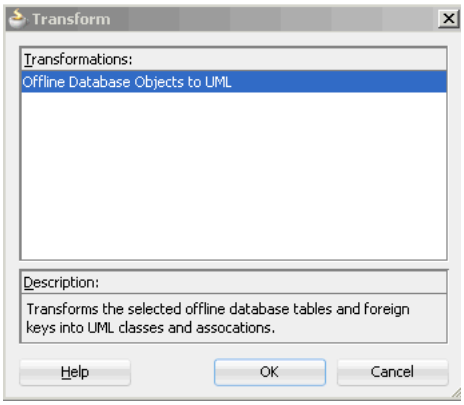
  - A new database diagram, as modeled tables and constraints.

4. Choose **UML to Offline Database.**
5. Click **Finish**.

To transform offline database objects to UML

1. Select the offline database object or objects you want to transform.
2. Right-click and choose **Transform**. Select from one of the following options:
   - **Model Only**. A UML class model based on the database schema is created in the Application Navigator in the current project.
   - **Same Diagram**. A UML class diagram based on the offline schema is created. The new classes can be viewed in both:
     - The Application Navigator.
     - The UML Class diagram, as classes for each transformed database table.
   - **New Diagram**. When the wizard finishes, a UML class diagram based on the offline database schema is created. The classes can be viewed in both:
     - The Application Navigator as a new UML classes.
     - A new class diagram, as classes for each transformed database table.
3. The Transform dialog appears as shown in Figure 22–13. Select **Offline Database Objects to UML**.
4. Click **OK.**

Figure 22–13 Transformation Dialog Showing Offline Database to UML Dialog Option



To create offline database objects from UML using the New Gallery Wizard:

1. In the Application Navigator, select the project containing the UML classes to transform.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Database Tier** and select **Offline Database Objects**. In the **Items** list, double-click **Offline Database Objects from UML Class Model**.

   If the category or item is not found, make sure the correct project is selected, and select **All Features** in the Filter By dropdown list.

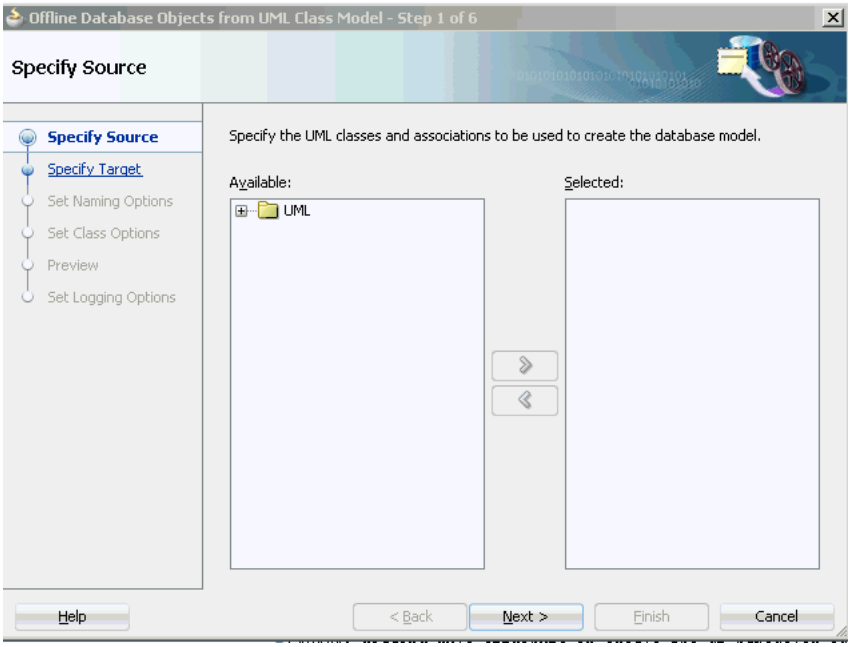
4. The Offline Database Objects from UML Class Model wizard opens, as show in Figure 22–14. Select the UML classes and associations to transform.
5. Click **Finish**. The offline database objects are created in the Application Navigator.

When you invoke the wizard from the New Gallery, the offline database objects are only available in the Application Navigator.

During transformation you have the following choices:

- How the UML names are converted to offline database names.
- How UML class hierarchies and many-to-many associations are handled.
- Where the schema the objects are going to.
- Which log messages you want to see. You can choose which transformation messages to be logged, and you can choose whether to send the messages to the Log Window or to a log file.

Figure 22–14 New Gallery Offline Database from UML



To choose the type of transformation messages to log:

1. Invoke the Offline Database Objects from UML Class Model wizard.
2. On the Set Logging Options page, choose where you want the log messages to appear, and the type of actions you want logged.

You have the option of reviewing the changes before running the transform. In addition, you can specify the types of transformation messages to be logged, and whether they appear in the Log Window or are sent to a log file.

Transformation generalization

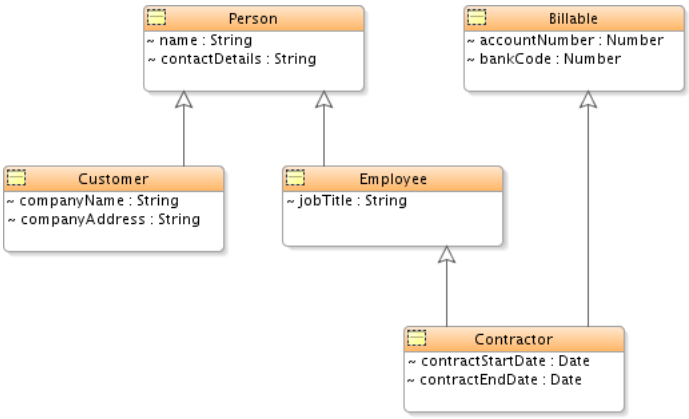
There are four types of generalization to specify on the Set Class Options page on the **Offline Database Objects from UML Class Model Wizard**.

The options are:

- Transform root classes
- Transform leaf classes
- Transform all classes, with generalization
- Transform all classes, creating foreign keys

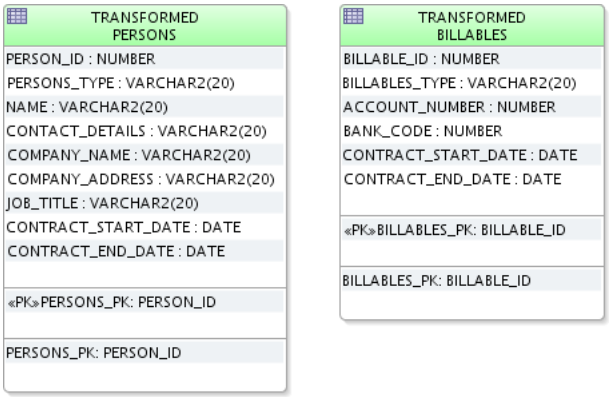
Consider the case of two root classes and three leaf classes, as shown in Figure 22–15.

Figure 22–15 Diagram Showing Two Root and Three Leaf Classes



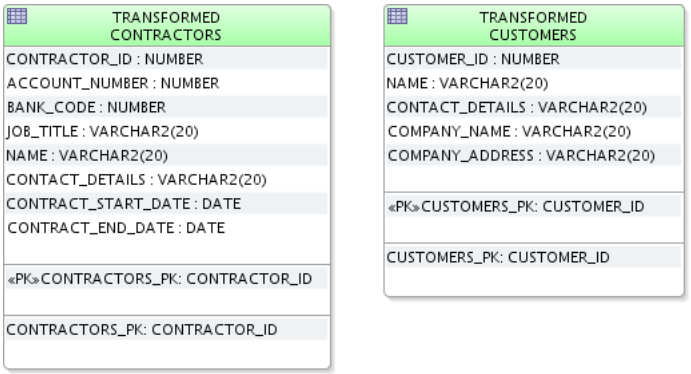
If you select the option **Transform Root Classes**, root classes are transformed into offline tables, and all the columns and foreign keys from their descendant classes in the hierarchy are also transformed as shown in Figure 22–16. You also have an option of creating a discriminator column. The discriminator column contains marker values for the persistence layer to decipher what subclass to instantiate for a particular row.

Figure 22–16 Java Classes Transformed Root Classes



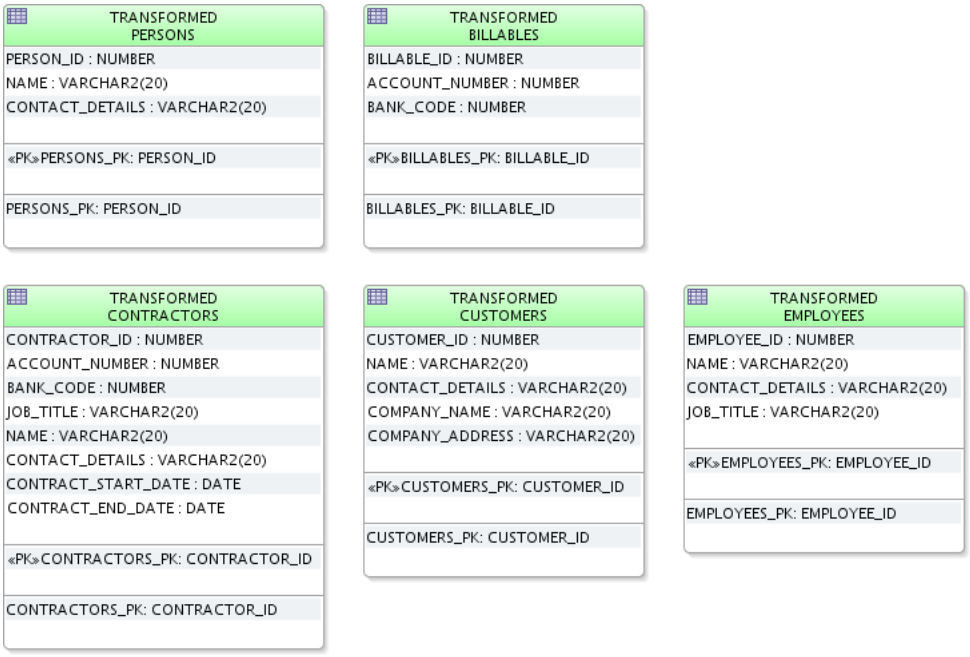
If you select the option **Transform only leaf classes, inheriting from generalized classes**, leaf classes are transformed into offline tables, and they inherit their columns and foreign keys from their ancestor classes, as shown in Figure 22–17.

Figure 22–17 Leaf Classes Transformed Inheriting From Generalized Classes



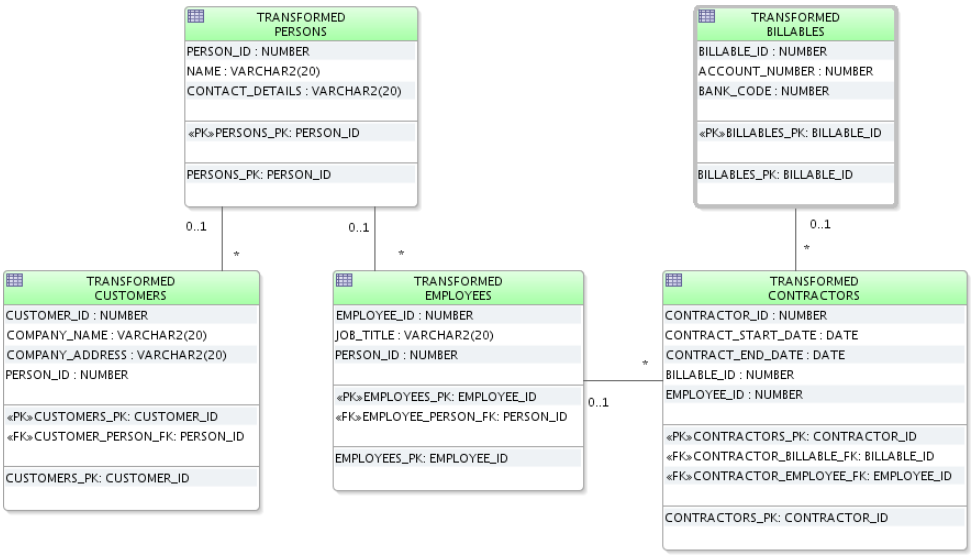
If you select the option **Transform all classes, inheriting from generalized classes**, an offline table is created for every class in the transform set. Each table inherits columns and foreign keys from its ancestor tables but is otherwise independent, as shown in example Figure 22–18.

Figure 22–18 All Classes Transformed, Inheriting from Generalized Classes



If you select the **Transform all classes, creating foreign keys** option to generalized classes, an offline table is created for every class in the transform set. No columns or foreign keys are inherited; a foreign key is created to the parent table or tables, as shown in Figure 22–19.

Figure 22–19 All Classes Transformed with Foreign Keys



22.7.2 Using DatabaseProfile

JDeveloper comes with a UML profile called DatabaseProfile which you can use to specify what happens to class models when they are transformed to offline database models. For more information about UML profiles, see Section 22.9, "Using UML Profiles."

For example, attributes can be assigned to column datatypes that are to be used when the attributes are transformed to columns.

DatabaseProfile allows you to use properties for the stereotypes it contains to control how elements are transformed. The stereotypes and their properties in this profile are described in Table 22–1.

Table 22–1 Stereotypes and Properties in DatabaseProfile

Stereotype	Applied to	Offline Database Type	Properties	Notes
Database Package	UML::Package	SCHEMA	Name after transfer	
Database Class	UML::Class	TABLE	Name after transfer	
Database Attribute	UML::Property	COLUMN	Name after transfer Datatypes Primary	A UML Property can be an "attribute" or an association end
Database Datatype	UML::Type	n/a	Datatype Primary	In the UMl metamodel, Type is the superclass for a large range of elements, including, Class, Association, PrimitiveType and so on. This stereotype can be applied to all of them. It can be read by the transformer whenever a property (attribute) is of a certain type.

Table 22–1 (Cont.) Stereotypes and Properties in DatabaseProfile

Stereotype	Applied to	Offline Database Type	Properties	Notes
Database Association	UML:: Association	CONSTRAINT	Name after transfer Foreign Key naming rule	
Database Generalization	UML:: Generalization	CONSTRAINT	Name after transfer Foreign Key naming rule	Certain transforms create foreign keys out of generalizations and this stereotype can apply here.

The attributes, or properties, are described in Table 22–2.

Table 22–2 Properties of Stereotypes in DatabaseProfile

Property	Description	Type
Name after transfer	The name of the transformed database object. If blank, default naming rules are applied.	String
Datatype	SQL text of the datatype. There are a number of datatypes, including default, ansi, Oracle, and other supported database types.	String
Foreign key naming rule	The rule to use when naming a foreign key from an association: use the UML name, use the databaseName property or derived a default name from the table names.	Both Tables Database Name Owning Name UML Name
Primary	Flag to indicate that transformed column is part of the primary key for the parent table.	Boolean

To use DatabaseProfile to transform a class model:

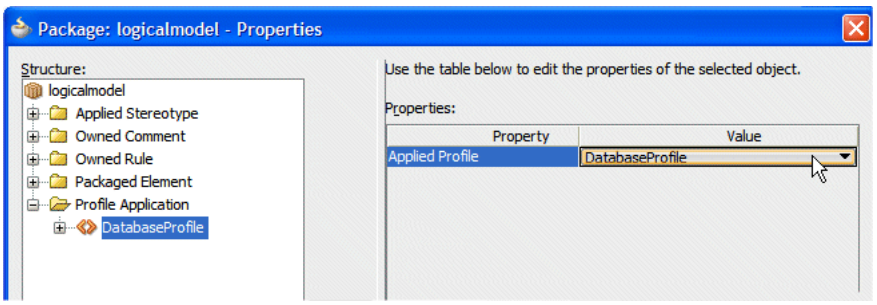
1. Open the Package Properties dialog of a UML package `package.uml_pck` by right-clicking on it in the Application Navigator and choosing **Properties**.

   For more help at any time, press F1 or click **Help** from within the Package Properties dialog.

2. In the Package properties dialog, select **Profile Application** and click **Add**.

3. In the Properties window, select `DatabaseProfile` from the list of available profiles, as shown in Figure 22–20.

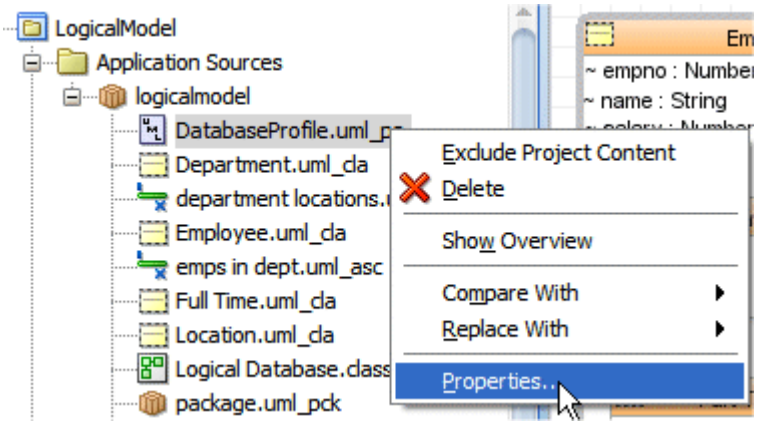
Figure 22–20 Choosing the Profile to Apply



Optionally, choose to specify the name to use after the package has been transformed into an offline database schema. Select the **Applied Sterotype** node and click **Add**. Under Properties, a new property `Name after transfer` is listed. Enter a name.

Click **OK**. A new file `DatabaseProfile.uml_pa` is now listed in the Application Navigator, as shown in Figure 22–21.

Figure 22–21 Profile Application File

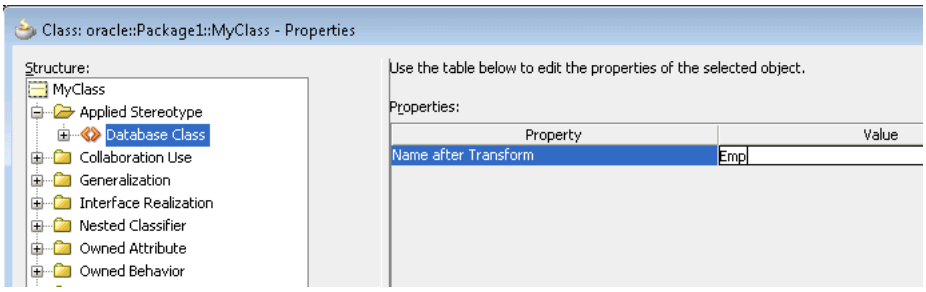


4. To examine the Database Profile file, right-click it in the Application Navigator and choose **Properties**. The dialog shows the profile that is will be used in the transform. Click **OK** to close the dialog.

5. Now you can apply stereotypes to the various elements in the project. In the example shown in Figure 22–21, you apply stereotypes to the Employee class by right-clicking `Employee.uml_cla` and choosing **Properties**. This opens the Class Properties dialog for that element.

   To specify the name to use after transformation, select **Applied Stereotype**, click **Add**, and choose `Database Class`. Under **Properties**, enter a value next to **Name after Transform**, as shown in Figure 22–22.

Figure 22–22 Applying a Name after Transform

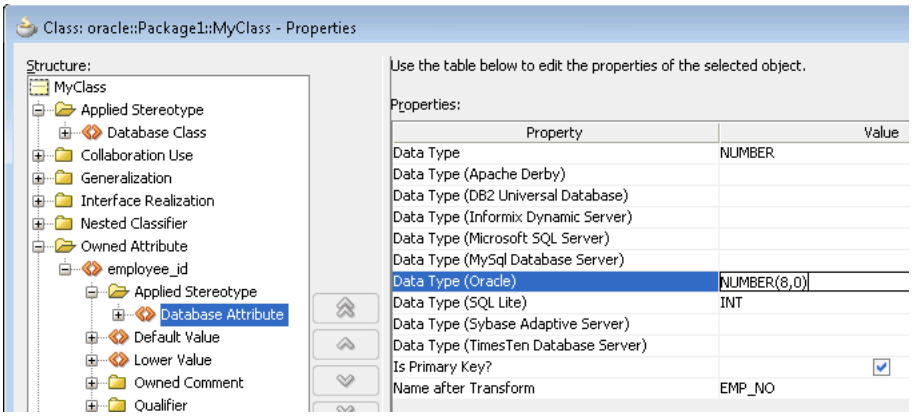


6. You can apply stereotypes to other elements. For example, you can specify datatypes and a primary key to attributes owned by this class. In the same Class Properties dialog, expand **Owned Attibute**. and select an existing attribute, or create one by clicking **Add** and entering a name for it.

   Expand the node for the owned attribute, select **Applied Stereotype** and click **Add**. Figure 22–23 shows that at this level you can specify a number of datatypes,

whether the attribute should be transformed to a primary key, and the name after transform.

Figure 22–23 Using Stereotypes with Owned Attributes



For information about the stereotypes and properties covered by DatabaseProfile, see Table 22–1, " Stereotypes and Properties in DatabaseProfile" and Table 22–2, " Properties of Stereotypes in DatabaseProfile".

7. Once you have set the stereotypes you want to have applied, you can proceed to transform the UML Class model following the steps in Section 22.7.1, "How to Transform UML and Offline Databases." This time, the stereotypes and properties in DatabaseProfile you have used are applied during transformation.

22.8 Importing and Exporting UML Using XMI

UML models created using other modeling software can be imported into JDeveloper using XML Metadata Interchange (XMI) if the models are UML 2.1.1 compliant.

The XMI specification describes how to use the metamodel to transform UML models as XML documents. JDeveloper complies with the specification. For more information see the "Catalog of OMG Modeling and Metadata Specification" at `http://www.omg.org/technology/documents/modeling_spec_catalog.html`.

22.8.1 How to Import and Export UML Models Using XMI

The following are restrictions that apply to import processes:

- Diagrams cannot be imported.
- XMI must be contained in a single file. Any profiles referenced by XMI must be registered with JDeveloper through **Tools > Preferences > UML > Profiles** before importing. The profiles must be in separate files. For more information see, Section 22.9, "Using UML Profiles".

Diagrams can be automatically created when you import classes, activity, sequence, and use case elements. The dialog will offer you this option during the process, as shown in Figure 22–25.

To import UML model as XMI:

1. With an empty project selected in the navigator, choose **File > Import.**
2. Select **UML from XMI**, then click **OK**.

3. Complete the Import UML from XMI dialog.

Figure 22–24 Import Choose UML from XMI Dialog

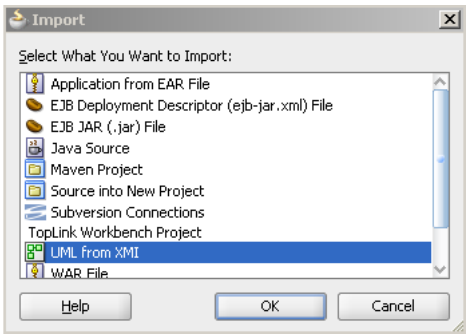
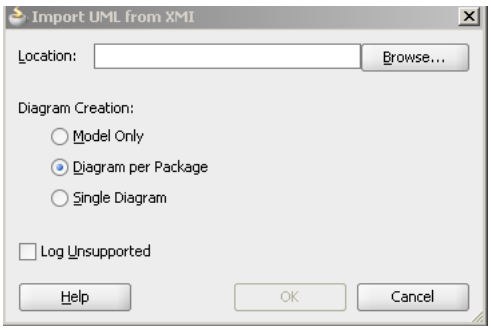


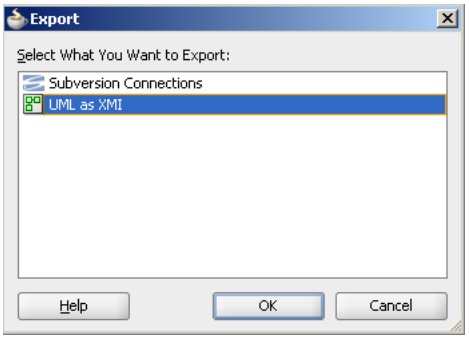
Figure 22–25 Import UML from XMI Detail Dialog



To export UML model as XMI:

1. Select your project.
2. Choose **File > Export**.

Figure 22–26 Export Choose UML as XMI Dialog



22.8.2 Typical Error Messages When Importing

There are some typical errors and warnings that you can encounter in your XMI Import Log during import. Many of these can be easily resolved with a few simple steps.

As with other XML, the structure of a valid file is specified by an XML schema which are referenced by xmlns namespaces. The XML consists of elements that represent objects or the values of their parent element object and attributes that are values. Sometimes the values are references to other objects that may be represented as an href as for HTML. Double clicking on the items in the log navigates to the problem element. Often issues arise because of incorrect namespaces and standard object references.

The following are typical error messages, and how to resolve them:

Missing Profile

- Error(16,80): The appliedProfile property has multiplicity [1..1]
- Error(17,70): Attempt to deference missing element http://example.oracle.com/MyProfile#_0
- Warning(2,356): http://example.oracle.com/MyProfile is not a recognized namespace
- Warning(22,142): Element urn:uuid:2b45f92d-31c8-4f67-8898-00a2f5bbfd22 ignored

In UML there is an extension mechanism that allows further XML schemas to be specified in a 'profile'. The first three problems above indicate that a relevant profile has not been registered. To register a profile see, Chapter 22.9, "Using UML Profiles".

Invalid XMI Version

- Error(2,360): 2.0 is incorrect version for http://schema.omg.org/spec/XMI/2.1

  This message occurs because there is a mismatch between the xmi:version attribute and the xmlns:xmi namespace. The xmi:version should be 2.1.

Invalid UML Namespace

- Warning(2,356): http://schema.omg.org/spec/UML/2.1.1/Unknown is not a recognized namespace.

  This message occurs because the xmlns:uml namespace should be http://schema.omg.org/spec/UML/2.1.1/uml.xml.

Invalid Standard L2 Profile Namespace

- Error(13,80): The appliedProfile property has multiplicity [1..1].
- Error(14,81): Attempt to deference missing element http://schema.omg.org/spec/UML/2.1.1/L2Unknown#_0
- Warning(2,344): http://schema.omg.org/spec/UML/2.1.1/L2Unknown is not a recognized namespace

This case is when a standard profile is already registered with the tool. Change the XMI so that the xmlns namespace is http://schema.omg.org/spec/UML/2.1.1/StandardProfileL2.xmi and the reference is http://schema.omg.org/spec/UML/2.1.1/StandardProfileL2.xmi#_0.

Invalid Standard L3 Profile Namespace

There is a second standard profile already registered. If the profile is incorrectly referenced the messages will be similar to the L2 Profile Namespace. The correct namespace is http://schema.omg.org/spec/UML/2.1.1/StandardProfileL3.xmi and the correct reference is http://schema.omg.org/spec/UML/2.1.1/StandardProfileL3.xmi#_0.

Invalid Standard Data Type Reference

- Error(7,75): Attempt to deference missing element http://schema.omg.org/spec/UML/2.1.1/Unknown#String

There is a standard set of data types to reference and resolve this error, such as attribute type. The XMI href should be updated to one of the following:

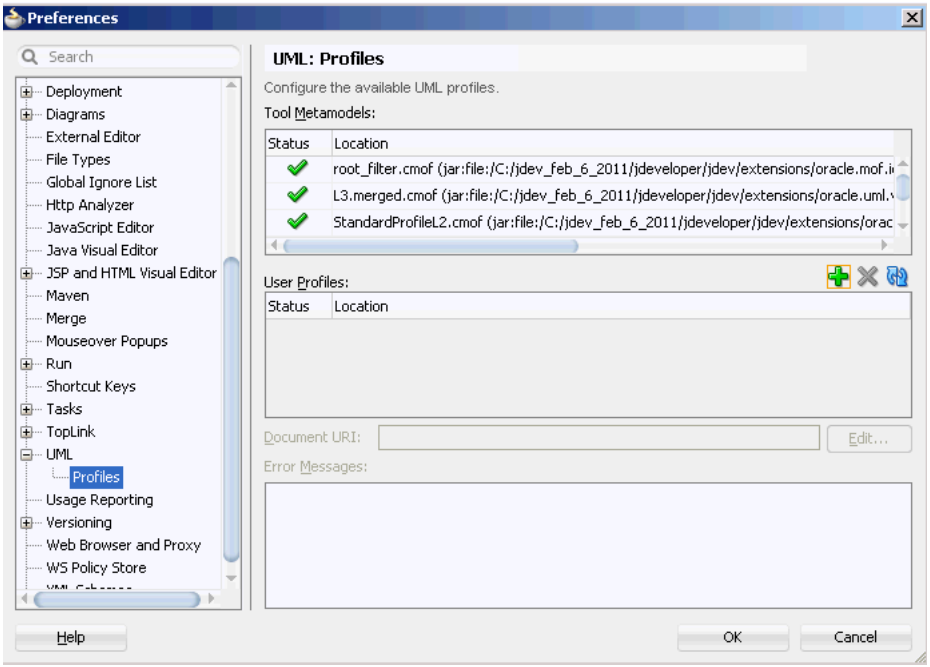
- http://schema.omg.org/spec/UML/2.1.1/uml.xml#Boolean
- http://schema.omg.org/spec/UML/2.1.1/uml.xml#Integer
- http://schema.omg.org/spec/UML/2.1.1/uml.xml#String
- http://schema.omg.org/spec/UML/2.1.1/uml.xml#UnlimitedNatural

22.9 Using UML Profiles

UML Profiles are subsets of the UML metamodel. These subsets specify well-formedness rules beyond those identified in the UML metamodel. Well-formedness rule is a term used in the normative UML metamodel specification to describe a set of constraints written in UML Object Constraint Language (OCL) that contribute to the definition of a metamodel element. JDeveloper comes with a handful of standard profiles for UML.

You can add your own profile by going to **Tools > Preferences > UML > Profiles**, and clicking the add symbol, as shown on Figure 22–27. Once you have added a custom profile, edit the document URL by selecting the profile and clicking **Edit**. For more information on UML Profiles, see the OMG Catalog at, http://www.omg.org/technology/documents/profile_catalog.html.

Figure 22–27 UML Profiles Dialog



Profiles allow you to apply stereotypes to UML models, and you use them by applying them to a UML package.

To use a UML profile:

1. If necessary, create a new UML package from the New Gallery. Choose **File > New**, and in the **General** category choose **UML** and under Items choose **Package**.

   Otherwise open the Package Properties dialog of an existing UML package by right-clicking on it in the Application Navigator and choosing **Properties**.

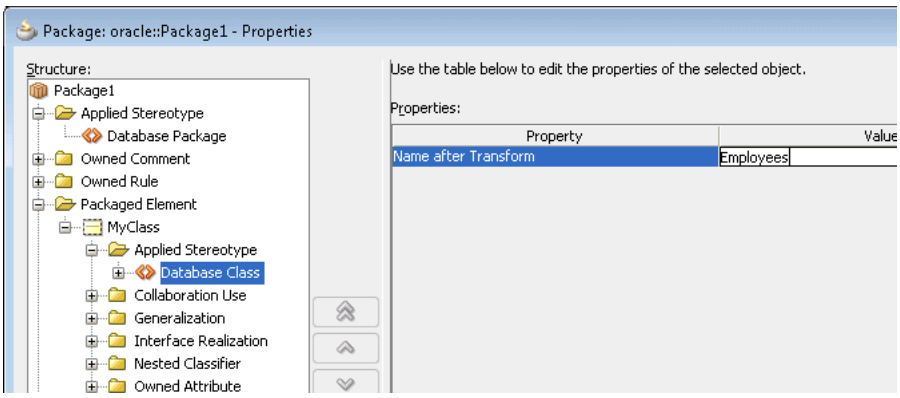
   For more help at any time, press F1 or click **Help** from within the Package Properties dialog.

2. In the Package Properties dialog, select the Profile Application node, click **Add** and choose the UML profile you want to use from the list of those available.

3. Now you can create a class, and add a stereotype. In the Package Properties dialog, select the **Packaged Element** node and click **Add** and choose `Class`.

4. Expand the Class node, and choose **Applied Stereotype** and click **Add**. The property that you can give a value to depends on the UML profile you are using. Figure 22–28 shows the Name after Transform property from the UML profile DatabaseProfile, which is delivered as part of JDeveloper.

Figure 22–28 Stereotype Properties



This section has described how to associate a profile with a UML package, and apply a stereotype at package level. Depending on the type of profile, you may be able to apply stereotypes to other elements. An example is given in Section 22.7.2, "Using DatabaseProfile."

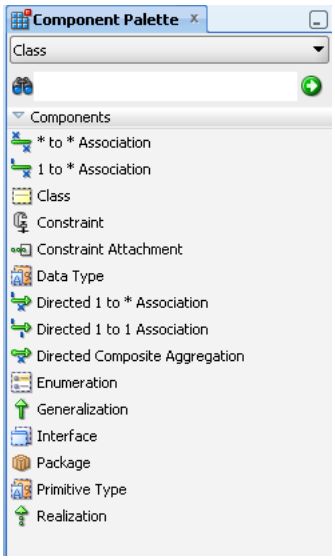
22.10 Working with UML Class Diagrams

Use the UML class diagram to model a collection of static elements such as classes and types, their contents and relationships, as well as visually create or inspect classes, interfaces, attributes, operations, associations, generalizations and realizations.

22.10.1 How to Work with Class Diagrams

You can use the Component Palette to add classes and related elements to your class diagram. Each of the elements is represented by a unique icon and description as shown in Figure 22–29.

Figure 22–29 Class Diagram Component Palette



To add classes and interfaces to a diagram:

Create classes and interfaces by dragging the class onto the diagram. The element is created in the location specified by the model path in your project settings and default properties. (Application > Default Project Properties). You can also model packages by clicking on the package.

If you right-click a modeled package and choose **Drill Down**, a diagram is displayed for that package.

To add or edit class properties:

Class properties are added to modeled classes and interfaces on a diagram by doing one of the following:

- Double-click the modeled class or interface to access the properties dialog.
- Right-click the class or interface and choose **Properties**.

To add generalizations, realizations, and associations:

Generalized structures are created on a diagram of classes by using the Generalization icon on the Class Component Palette.

Where an interface is realized by a class, model it using the Realization icon on the Class Component Palette for the diagram.

A variety of associations can be created between modeled classes and interfaces using the association icons. Associations are modified by double-clicking the modeled association and changing its properties.

To add nested classes and nested interfaces:

Nested classes and nested interfaces are created either by creating them in the modeled class or interface using in-place create (with symbolic presentation only) and by changing shape display preferences, or by right-clicking the class and choosing **View As > Expanded** then creating another class inside the expanded node.

To add attributes and operations:

Attributes and operations are added to modeled classes and interfaces by doing any of the following:

- Double-click the modeled class or interface, then add the attribute or operation using the element property dialog.
- Right-click the class or interface and choose **Properties,** then add the attribute or operation using the element property dialog.
- Drag an existing attribute or operation from one class or interface on a diagram to another class or interface on the same diagram.

  The order of an attribute or operation within a class or interface is changed by dragging it up or down on the screen. The Sort Alphabetically property for attributes or operations must be deselected: (**Tools >Preferences > Diagrams > Class > Edit Preferences for: Class or Interface | Attributes or Operations | Sort Alphabetically**).

To hide one or more attributes or operations:

1. Select the attributes or operations to hide.
2. Right-click the selected items and choose **Hide > Selected Shapes**.

To show all hidden attributes or operations on a class or interface:

Select the class or interface, then right-click and choose **Show All Hidden Members**.

22.10.1.1 How to Read a Class Diagram

Classes are represented as rectangles containing class name and details. Classes can be displayed as compact, symbolic or expanded nodes. To set the display properties go to **Tools > Preferences > Diagrams > (Diagram Type) > Display Options**.

You can also change the way elements are represented on a diagram. On the diagram, classes and interfaces are divided into compartments, with each compartment containing only one type of information. An ellipsis (...) is displayed in each compartment when not large enough to display its entire contents. To display all the attributes of a modeled class, right-click the class and choose **Optimize Shape Height**. Figure 22–30 shows an example of a typical class diagram layout.

All attributes and operations display symbols to represent their visibility. The visibility symbols are: + Public, - Private, # Protected, and ~ Package.

Figure 22–30 UML Class Diagram with Elements

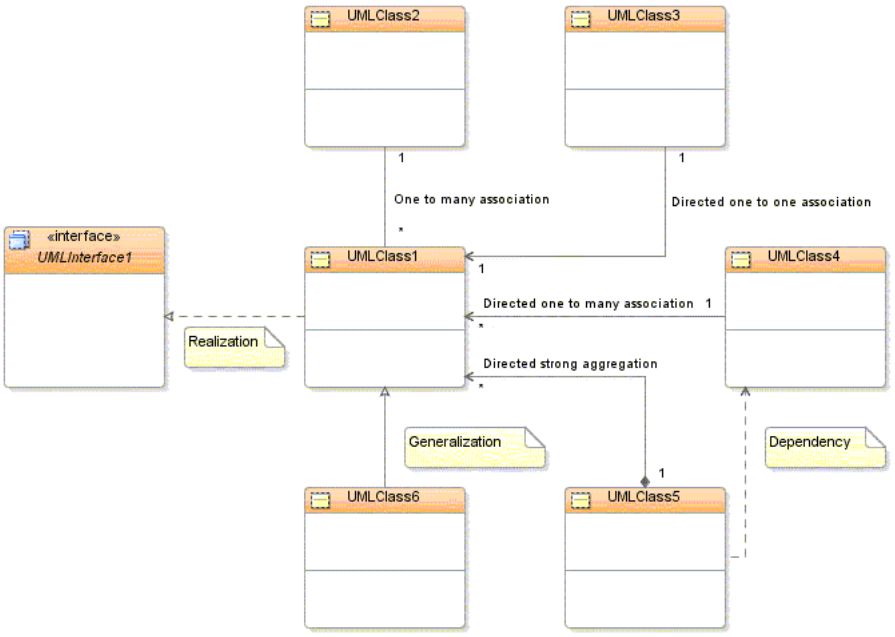


Table 22–3 Elements in Class Diagram

Elements	Description
Interface	Interfaces are represented with a keyword in the name compartment: «interface». interfaces can be displayed as compact, standard or expanded nodes. Nested classes and interfaces can be modeled inside standard and expanded interfaces.
Enumeration	Enumerations are data types with a finite, and normally small, set of named literals. Enumerations contain sets of named identifiers to represent the values of the enumeration. An enumeration has a name that describes its role in a model.
*1 to * Association	One to Many defines a relationship between classes. Represented on the diagram as a solid line.
Directed One to One Association	Directed One to One Association is represented on the diagram as a solid line with an open arrowhead in the direction of the association.
Directed One to Many Association	Directed One to Many Association is represented on the diagram as a solid line with an open arrowhead in the direction of more than one association.
Directed Composite Aggregation	Represented on the diagram as a solid line with an open arrowhead in the direction of the association and a filled diamond shape at the originating end of the association.
Generalization	Defines generalization relationships between classes. Represented on the diagram as a solid line with an empty arrowhead pointing towards the specialized class or interface.
Realization	Defines where an interface is realized by a class. Represented on the diagram as a dashed line with an empty arrowhead pointing towards the implemented interface.

Table 22–3 (Cont.) Elements in Class Diagram

Elements	Description
Dependency	Represents where one diagram element depends on another. Represented on the diagram as a dashed line with an open arrowhead in the direction of the dependency.
Class	Represents an object. Classes form the main building blocks of an object-oriented application. Represented on the diagram as a rectangle containing three compartments stacked vertically.*
Package	Use to divide a system into multiple packages, which can simplify and make the system easier to understand.
Data Type	Data types are modeled elements that define your data values.
Primitive Type	Primitive types or data types are data types such as boolean, byte, decimal, DateTime, Double Float, and Time.
Constraint	Constraints are the degree of freedom, or lack thereof that you have in modeling a system behavior, or solution.

22.10.1.2 How to Specify UML Operation Notation

Operation notation in this release is closely related to the UML2 notation. Previously the notation for an operation in a class diagram followed Java-line syntax.

```
print(int param1, int param2):void
```

In this release of JDeveloper, operation notation follows UML2 syntax:

```
print(param1:int, param2:int):void
```

22.10.2 Refactoring Class Diagrams

If you rename or move a class using the in-place edit functionality on a diagram, the source code for the class is refactored automatically. Renaming or moving a Java package on a diagram automatically refactors the contents of that package.

Deleting a field, method, or inner class on a diagram automatically applies the Delete Safely refactoring pattern. To apply a refactoring pattern to a Java class, interface, enum, or member on a diagram, select the class or member on the diagram and choose the refactoring pattern from the refactoring menu.

The following refactoring patterns are available for the Java classes, interfaces, and enums on a Java class diagram:

- Rename
- Move (applies to both single and multiple selections on the diagram)
- Duplicate
- Extract Interface
- Extract Superclass

The following refactoring patterns are available for the Java fields and methods on a Java class diagram:

- Rename
- Move
- Make Static
- Pull Members Up

- Push Members Down
- Change Method (Java methods only)

22.10.2.1 How to Invoke a Refactoring Operation

There are several automated refactoring operations available that enhance code quality by improving the internal structure of code without altering the external behavior.

To invoke a refactoring operation:

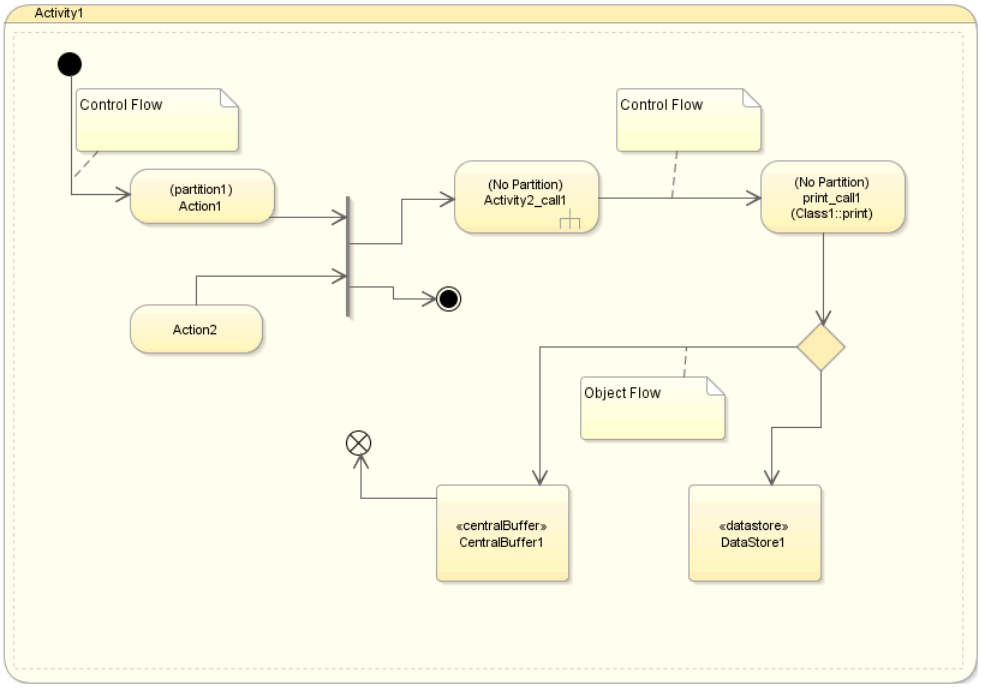
1. Select a program element in a source editor window, navigator pane, or structure pane.
2. Right-click on the program element.
3. Choose an operation from the context menu.
4. You can also choose **Refactor** from the toolbar and select a refactoring operation from the drop-down list.

22.11 Working with UML Activity Diagrams

Use activity diagrams to model your business processes. Your business process are coordinated tasks that achieve your business goals such as order processing, shipping, checkout and payment processing flows.

Activity diagrams capture the behavior of a system, showing the coordinated execution of actions, as shown in Figure 22–31.

Figure 22–31 Sample Activity Diagram Showing Elements



22.11.1 How to Work with Activity Diagrams

The Component Palette contains the elements that you can add to your activity diagram. An Activity is the only element that you can place directly on the diagram. You can place the other elements inside an Activity. Each of the elements is represented by unique icons as well as descriptive labels, as shown in Figure 22–32 and Table 22–4.

Figure 22–32 Component Palette for Activity Diagram

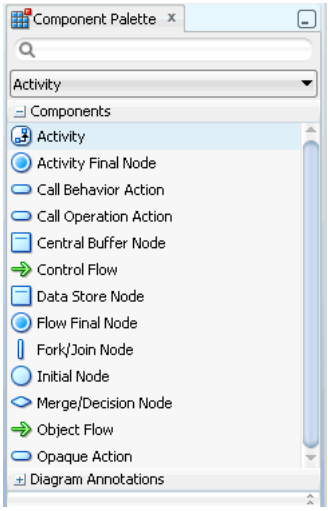


Table 22–4 Activity Diagram Elements

Element	Description
Action	An action is the fundamental unit of behavior specification, for example, Send Invoice or Receive Payment. It represents a single step within an activity. An action takes a set of inputs and converts them into a set of outputs. The execution of an action represents some transformation or processing in the modeled system. An action may receive inputs in the form of control flows and object flows (the latter via input pins) and passes the results of its processing or transformations to outgoing control flows or object flows (the latter via output pins) and onto downstream nodes. Execution of the action cannot begin until all its prerequisites are satisfied.
Activity	A behavior performed by a system, for example a business process. An activity is a behavior defined by its owned actions, object nodes and the flows between them.
Activity Final Node	Terminates the execution of the activity when it first receives a control token. There can be multiple final nodes in an activity. An Activity Final Node indicates that every action on this diagram has finished.
Fork/Join	Displayed as a vertical or horizontal bar. A Fork is a control node that has a single incoming flow and two or more outgoing flows. A Join is a control node that synchronizes a number of incoming flows into a single outgoing flow. Fork/Join pairs can be combined as a single diagram node.
Call Behavior Action	Maps the action inputs and outputs are simply mapped to the behavior parameters as appropriate.

Table 22–4 (Cont.) Activity Diagram Elements

Element	Description
Call Operation Action	Transmits an operation call request to the target object, where it may cause the invocation of associated behavior. The behavior results become the action outputs. The argument values of the action are available to the execution of the invoked behavior.
Central Buffer	A type of object node. It gives the node the capability of storing (buffering) tokens. It manages the tokens that arrive at incoming flows from one or more object nodes and selects which tokens and in what order these tokens will be presented to the downstream object nodes via the outgoing flows.
Control Flow	Shows the flow of control tokens.
Data Store	A type of object node that passes a buffer for non-transient data.
Flow Final Node	Terminates any incoming flow without terminating the execution of the entire activity.
Initial Node	The starting point for executing an activity. It has no incoming flows and one or more outgoing flows. There can be only one initial state on a diagram.
Object Flow	Connects object nodes. Object flows can be connected to actions using pins.
Merge Node	A merge node has two or more incoming flows and a single outgoing flow. A Decision has one incoming flow and two or more outgoing flows

To create an activity diagram:

1. Create a new diagram following the steps in To create a new diagram:.
2. Choose the elements to add to your diagram from the Component Palette as shown in Figure 22–32.

To show a partition on an activity diagram:

1. In the activity diagram, select an action.
2. In the Property Inspector, expand the Display Options node.
3. Select **Show Activity Partition**. The action on the diagram displays the text, (No Partition).
4. Click on the text. An editing box appears where you can enter a name for the partition.

22.11.1.1 Getting a Closer Look at the Activity Diagram Elements

To model activities, actions, central buffer nodes, and data store nodes, you need to start with a Activity diagram.

Partitions

You create partitions on a diagram by selecting an action, then selecting **Show Activity Partition** under Display Options in the Property Inspector.

Activities and Actions

Activities are created by first selecting the Activity icon on the Component Palette, then clicking on the diagram where you want to create the activity or action.

Initial and Final Nodes

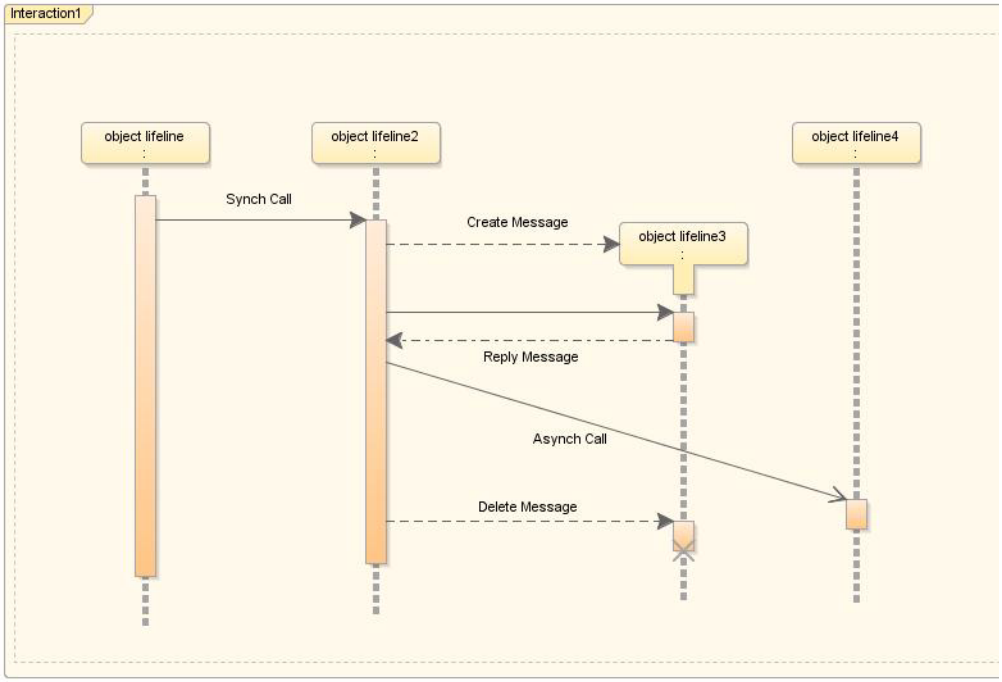
An initial node is a starting point for an activity execution. A final node is the termination of execution, either for the activity if an activity final node, or a particular flow if a flow final node.

To create, click the **Initial Node** icon, the **Activity Final Node** icon, or **Final Flow Node** icon on the Component Palette, then click on the diagram where you want to place the node.

22.12 Working with Sequence Diagrams

The sequence diagram describes the interactions among class instances. These interactions are modeled as exchanges of messages. At the core of a sequence diagram are class instances and the messages exchanged between them to show a behavior pattern, as shown in Figure 22–33.

Figure 22–33 Typical Sequence Diagram Example



22.12.1 How to Work with Sequence Diagrams

The elements you add from the Component Palette are laid out in a default position on your sequence diagram. Object lifelines are aligned vertically, unless they are related to another object lifeline, in which case they are aligned with the create message. Synchronous and asynchronous messages and are placed in time order down the page.

Figure 22–34 displays the elements in the Component Palette available to use in your sequence diagram. Each element is represented by a unique icon as well as a descriptive label.

Figure 22–34 Sequence Diagram Component Palette

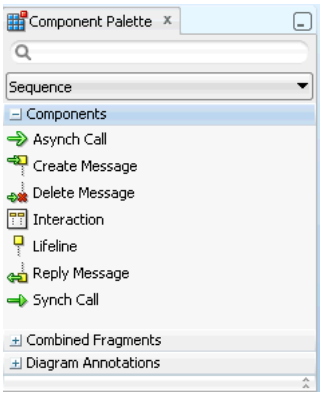


Table 22–5 Sequence Diagram Elements

Element	Description
Asynchronous Message	Represented on a diagram by a diagonal line with an open arrowhead. An asynchronous message is one for which the sender does not have to wait for a response before continuing with processing.
Creation Message	Represented on a diagram by the shifting down, relative to the originating object, of the rectangle and dashed line that represents the object to be created. A creation message is a message that leads to the creation of an object
Interaction	Captures the behavior of a single case by showing the interaction of the objects in the system to accomplish the task.
Message	A message is a model element that defines a specific kind of communication between participants in an interaction. A message conveys information from one participant, which is represented by a lifeline, to another participant in an interaction.
Object Lifeline	Represented on a diagram by a rectangular box with a vertical dashed line descending beneath it. An object lifeline represents the existence of an object over a period of time
Return	A return message is a message that returns from an object to which a message was previously sent. Return messages are valid only from synchronous messages, and are themselves synchronous.
Stop or Destroy Message	Represented on a diagram by showing the execution specification at the end of the message with a large cross through it. A stop message is a message that leads to the deletion of an object (or to the indication that an object is no longer needed).

To create a sequence diagram:

1. Create a new diagram following the steps in "To create a new diagram:" on page 22-2.
2. Choose the elements to add to your diagram from the Component Palette as shown in Figure 22–34, "Sequence Diagram Component Palette".

To start the sequence tracer:

On any part of the sequence diagram, open the context menu and choose **Trace Sequence**. The tracer steps through each of the execution specifications and messages, highlighting each one.

Note: **Trace Sequence** is available for a selected Interaction. It does not appear in the context menu for the diagram.

22.12.1.1 Getting A Closer Look at the Sequence Diagram Elements

The sequence diagram elements are added to your diagram by clicking or dragging and dropping from the component palette.

Interactions

Model your interactions within a system by adding the Interaction component to your diagram. Interactions are a container for the specific elements that comprise the behavior.

You can right-click an Interaction and choose **Sequence**, then **Automatic Layout** to autolayout the elements within the Interaction.

Object Lifelines

You add object lifelines to a sequence diagram by first adding an interaction then clicking on the Object Lifeline icon, and then clicking on the interaction. An edit box opens for you to enter an instance name for the object. This can be left blank for anonymous instances.

You can add a classifier by right-clicking on the object lifeline and choosing Attach Classifier, which opens a list of elements from which you choose the one you want associated with the object lifeline. Another way to attach a classifier is to drag the classifying object from the navigator onto the object lifeline. These methods are confirmed by the appearance (in the top left of the object lifeline) of an icon representing the classifying element.

Execution Specifications and Synchronous Messages

You add a synchronous message by clicking on the Message icon, then clicking on the vertical dashed line or execution specification that is the starting point for the message, and then on the vertical dashed line that is the destination of the message.

Merge execution specifications by overlapping them on the diagram. Then right-click and choose **Merge Overlapping Occurrences**.

You can move an execution specification (and the messages attached to it) to a position higher or lower than its original one. In some cases this will result in an invalid diagram. When this happens, the message line will turn red and the destination object will contain an arrow icon which indicates the direction the object should be moved, to make the diagram valid.

To resize an execution specification box, drag the small black box that appears on the lower edge when you select it. An execution specification will be resized if you drag a message line extending from it.

Open an editing box for the text by clicking on the message line and then clicking inside the gray box that appears.

The starting point and destination point of a synchronous message can be the same object lifeline, in which case you have created a self call.

Synchronous messages are depicted on the sequence diagram by solid lines with filled arrowheads.

Creation Messages

Add a creation message by clicking on the Creation Message icon, then on the originating object, then on the object to create. The rectangle and dashed line that represents the object is shifted down the page relative to the originating object.

If the object to be created is not already on the diagram, click within the interaction that contains the originating object to create an object lifeline representing the object. By default, a creation message is given the name "create". You can open an edit box for the message name by clicking on the message line and then clicking inside the gray box that appears.

Stop or Destroy Messages

Before you add a stop or destroy message, you must already have added an object lifeline for the object that the message deletes.

Add a stop or destroy message by clicking on the Stop or Destroy Message icon, then on the originating object, then on the object to delete. If you start and end the stop message on the same object, you will create a self-deleting object. The execution specification at the end of a stop message is shown with a large cross through it. You can open an edit box for the message name (for example, close) by clicking on the message line and then clicking inside the gray box that appears.

If you add a stop message earlier in the sequence than other execution specifications, those execution specifications will be not be implementable and will be shown with a warning flag.

Return (Messages)

Add a return message by clicking on the Return icon, then on an end execution specification, then on the corresponding start execution specification. You will not be able to end this return message line on any other object. The return message is depicted by a dashed line with a filled arrowhead. You can open an edit box for the text of the message by clicking on the message line and then clicking inside the gray box that appears.

Asynchronous Messages

Add an asynchronous message (and the execution specifications at each end) by clicking on the Async Message icon, then clicking on the vertical dashed line or execution specification that is the starting point for the message, then on the vertical dashed line that is the destination of the message. You can open an edit box for the text of the message by clicking on the message line and then clicking inside the gray box that appears.

The starting point and destination point of an asynchronous message can be the same object lifeline, in which case you have created a self call.

Asynchronous messages are depicted on the sequence diagram using diagonal lines and open arrowheads.

22.12.1.2 How to Work with Sequence Diagram Combined Fragment Locks

On your sequence diagram interactions you will see combined fragment lock icons. Locking and unlocking an interaction allows you to keep the combined fragment behavior within that interaction on that diagram, or extend its reach to other interactions and other diagrams.

22.12.1.3 Using Combined Fragments

A combined fragment defines an expression of an interaction defined by an interaction operator and corresponding interaction operands. A Combined Fragment reflects a piece or pieces of interaction (called interaction operands) controlled by an interaction operator, whose corresponding boolean conditions are known as interaction constraints. It displays as a transparent window, divided by horizontal dashed lines for each operand.

Figure 22–35 shows a loop fragment that iterates through purchase items, after the cashier requests payment. At this point, two payment options are considered and an alternative fragment is created, divided to show the two operands: cash and credit card. After the fragment completes its trace, the cashier gives a receipt to the customer, under the fulfilled condition of payment requirements met.

Figure 22–35 Typical Sequence Diagram with Combined Fragments

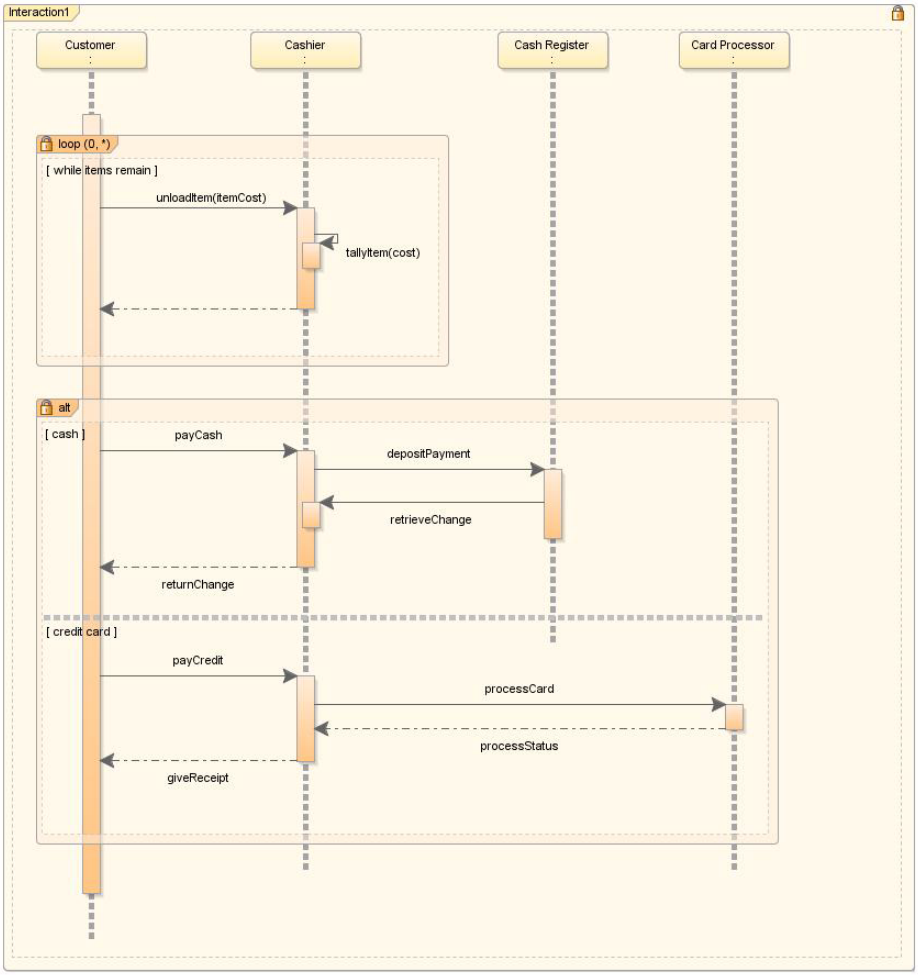


Figure 22–36 shows the combined fragments that display in the Component Palette when your diagram is open in the diagrammer.

Figure 22–36 Combined Fragments Drop-down List in Component Palette

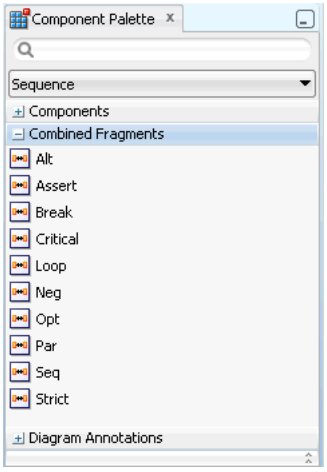


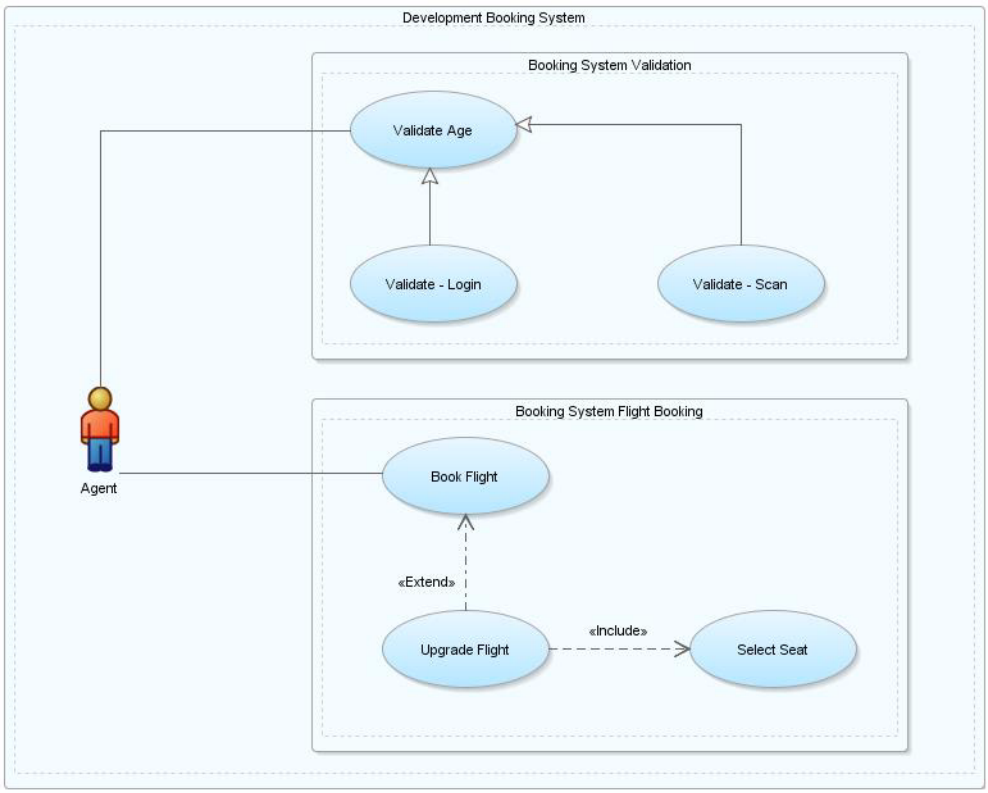
Table 22–6 Combined Fragments Interaction Operators

Interaction Operator	Description
alt	Use to divide up interaction fragments based on Boolean conditions.
assert	Use to specify the only valid fragment to occur.
Break	Use to designate that the combined fragment represents a breaking scenario in the sense that the operand is a scenario that is performed instead of the remainder of the enclosing interaction fragment.
Critical	Use to indicate a sequence that cannot be interrupted by other processing.
Loop	Use to indicate that the operand repeats a number of times, as specified by interaction constraints.
Neg	Use to assert that a fragment is invalid, and implies that all other interaction is valid.
Opt	Use to enclose an optional fragment of interaction.
Par	Indicate that operands operate in parallel.
Seq	Use to indicate that the combined fragment is weakly sequenced. This means that the ordering within operands is maintained, but the ordering between operands is undefined, so long as an occurrence specification of the first operand precedes that of the second operand, if the occurrence specifications are on the same lifeline.
Strict	Use to indicate that the behaviors of the operands must be processed in strict sequence.

22.13 Working with Use Case Diagrams

Use case diagrams overview the usage requirements for a system. For development purposes, use case diagrams describe the essentials of the actual requirements or workflow of a system or project, as shown in Figure 22–37.

Figure 22–37 Typical Use Case Diagram



22.13.1 How to Work with Use Case Diagrams

Use case diagrams show how the actors interact with the system by connecting the actors with the use cases with which they are involved. If an actor supplies information, initiates the use case, or receives information as a result of the use case, then there is an association between them.

Figure 22–38 displays the Component Palette with the elements available to add to your use case diagram. Each element is represented by a unique icon and descriptive label.

An Interaction is the only element you can add directly to the diagram. You put all of the other elements within an Interaction.

Figure 22–38 Use Case Elements in the Component Palette

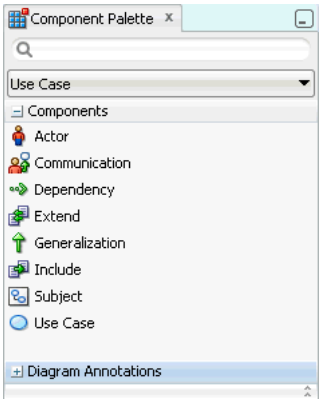


Table 22–7 Use Case Elements

Component	Description
Actor	Represents an abstract role within a system.
Communication	Identifies where an actor is associated with a particular use case.
Dependency	Shows a relationship between one element and another.
Extend	Shows a target use case extends the definition of a source use case.
Generalization	Identifies where one or more elements specialize another element. For example, an actor Team Member could be specialized to actors Manager and Developer.
Include	Shows a relationship in a use case that includes another use case.
Subject	Two types of subjects are available. One system usually contains sets of use cases and actors that comprise the whole system being modeled. The second type usually contains groups of use cases that comprise a coherent part of the system being developed.
Use Case	Indicates that one element requires another to perform some interaction.

To create a use case diagram:

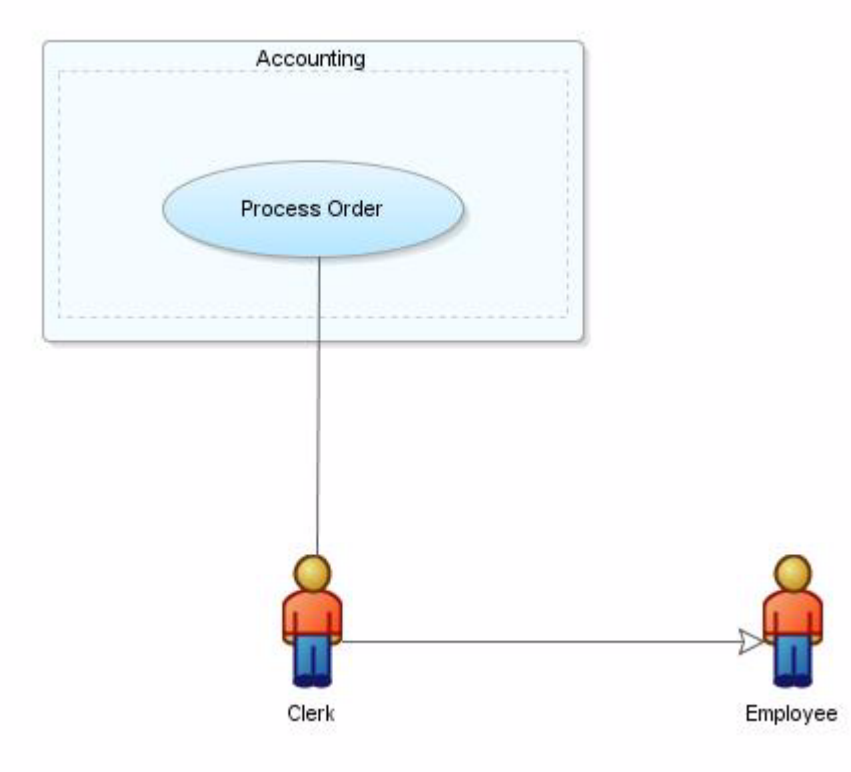
1. Create a new diagram following the steps in "To create a new diagram:" on page 22-2.
2. Choose the elements to add to your diagram from the Component Palette as shown in Figure 22–38.

22.13.1.1 Getting A Closer Look at the Use Case Diagram Elements

You can determine the appearance and other attributes for subject, actor and other objects of these types by modifying the properties in the Property Inspector, or by right-clicking the object and modifying the properties, or by creating and customizing an underlying template file.

You can use templates to add the supporting objects to the Component Palette. For more information, see Chapter 22.13.1.3, "How to Work with Use Case Component Palette Templates".

Figure 22–39 Use Case Subject, Actor and object Example



Subjects

You can show the system being modeled by enclosing all its actors and use cases within a subject. Show development pieces by enclosing groups of use cases within subject lines. Add a subject to a diagram by clicking on **Subject** in the Component Palette, then drag the pointer to cover the area that you want the subject to occupy. Figure 22–39 shows an accounting subject attached to their related actors. If you drop an element just inside a subject, the subject line expands to enclose the element. You also can manually resize subjects. If you reduce the size and there are elements that can no longer be seen, an ellipsis appears in the lower right corner.

Actors and Use Cases

Create actors on a diagram by clicking on the **Actor** icon on the Component Palette, and then clicking on the diagram where you want to create it.

To change the properties of an actor or use case, double-click on the modeled element and edit the element details in the editor.

Relationships Between Actors and Use Cases

You can represent interactions between actors and use cases on a diagram using the **Communication** icon on the Component Palette. You can create generalization structures between actors and between use cases by using the **Generalization** icon. To represent where one use case includes another, use the **Include** icon, and to represent where one use case extends another use the **Extension** icon.

You can annotate a diagram of use cases using notes, dependency relationships and URL links. Annotation components are available at the lower part of the component palette under **Diagram Annotations**.

Relationships Between Use Cases and Subjects

You can represent interactions between use cases and subjects using the **Communication** icon on the Component Palette.

Relationships Between Use Cases and Subjects

You can represent interactions between use cases and subjects using the **Communication** icon on the Component Palette.

22.13.1.2 How to Work with Use Case Templates

XHTML files can be used as templates for the documents that support some objects in use case diagrams. Templates allow you to determine the appearance and other attributes of the document initially created for each object type. You can either edit the source code of a template directly or use the visual editor.

To open a use case template:

Choose **File > Open**, then navigate to the `<jdeveloper_install>/jdev /system[...]/o.uml.v2.usecase.html/templates/usecase folder` and open (by double-clicking) one of the files located there.

The files have the following names and uses:

- `Actor.xhtml_act - template` for actor documents.
- `Casual.xhtml_usc - template` for casual use case document.
- `FullyDressed.xhtml_usc - template` for fully dressed use case documents.
- `Milestone.xhtml_sub - template` for subject - milestone use case documents.
- `SystemBoundary.xhtml_sub - template` for subject - use case documents.

To edit the use case template source code:

Select the Source tab. To insert HTML tags at specific locations, open the context menu and choose **Insert HTML**.

As a general rule, do not remove tags that include the uml: prefix (for example <uml:usecase_extends>). If you are removing an entire section of the template (for example, between a td tag and a /td tag), any uml: prefixes can be safely removed along with the other content of the section.

To change the appearance of text on the use case [modeler] template:

Select the Editor tab and the text that you want to change, then choose one or more of the formatting options from the toolbar.

To insert HTML interface objects, code objects, and development inputs:

Select the Editor tab, select the place in the template where you want to make the insertion, then choose the required location and object from the context menu.

You can insert HTML interface objects (such as buttons and checkboxes), HTML code objects (such as anchors and tables), and development inputs (such as comments and processing instructions).

You are initially given the choice of inserting an object before a selected template item or after it, or into the <head> section of the template. Subsequently, these options are unavailable if inappropriate. If you choose **HTML** or **Browse** from the context menu, a dialog box is opened from which you can choose one of various HTML interface

objects and code objects. Choosing an item here will open an insertion dialog to enter properties appropriate to the object.

22.13.1.3 How to Work with Use Case Component Palette Templates

New use case templates can be created by copying the one included in JDeveloper. You can modify the copies as needed.

For each new component palette there must be one use case template that supports it. The use case template that you specify is not copied or moved: it remains in its original location.

To create a new Component Palette page and add a new use case component:

1. Go to **Tools > Configure Palette**.
2. In the Configure Component Palette dialog, select use case for Page Type.
3. Click **Add**.
4. Enter a name for the new Component Palette page.
5. From the drop-down list, choose **Use Case**, then click **OK**. The new page appears in the Component Palette. The Pointer item is included automatically.
6. Open the context menu for the component area of the Component Palette and select **Add Component.** The Add Use Case Template dialog opens.
7. Enter a name for the new component.
8. Enter the path to the use case template on which the new component is to be based. The path can be a URL.
9. To add the new component to the Component Page, click **OK**.

22.14 How Diagrams are Stored on Disk

Diagrams are stored on disk as diagram files. Diagram files reference the elements that are displayed on the diagram and contain display information for those elements (size, color, font, display of various properties etc.). Diagram files are stored in the folder for the package in which the diagram resides, and is stored in the model path specified in the project settings. Notes, diagram links and dependencies are also stored in the diagram file.

To set the model path, choose **Application** > **Default Project Properties** > **Project Source Paths > Modelers**.

Diagram elements such as Java classes are referenced in the diagram file, but their definition and implementation details are only stored in the implementation files for those elements. Although the diagrammatic details for these elements (position, color, size, etc.) are stored in the diagram file, no separate model definitions of these elements are stored.

22.15 How UML Elements are Stored on Disk

UML elements are stored in individual files. Their location is dependent on the package property of the element. These element files hold the properties defined against the various elements, but the diagram file still defines which elements are displayed on the diagram and the visual properties of those elements. Element files for modeled UML elements are stored in the appropriate package folder under the folder

specified in the project model path. To set the model path, choose **Application** > **Default Project Properties** > **Project Source Paths > Modelers**.

23

Developing Java EE and Java Applications Using Modeling

This chapter describes how to work with Java EE and Java application diagrams using the latest modeling tools and technologies included in Oracle JDeveloper.

This chapter includes the following sections:

23.1 About Developing Java EE and Java Applications Using Modeling

Oracle JDeveloper provides you with a wide range of diagramming tools to model your Java EE and Java applications systems.

23.2 Business Component Diagram

Use the business component diagram to visualize and organize the business entities and objects in your enterprise application.

For more information see, “Creating an Entity Diagram for Your Business Layer” in the *Oracle Fusion Middleware Fusion Developer’s Guide for Oracle Application Development Framework*.

23.3 Modeling EJB/JPA Components on a Diagram

Enterprise JavaBeans (EJBs) modeling helps you visualize your EJB entity relationships and architecture, and to quickly create a set of beans to populate with properties and methods, and to create a graphical representation of those beans and the relationships and references between them. Whenever a bean is modeled, the underlying implementation files are also created.

To model EJBs start by creating an EJB diagram. For more information, see "To create a new diagram:" on page 22-2. You can later add other elements like UML classes, Java classes, business components, offline database tables, UML use cases and web services to the same diagram. For more information, see "How to Work with Diagram Elements" on page 22-5.

About Entity, Session and Message-Driven Beans

Enterprise JavaBeans are created on a diagram by using the **Entity Bean** icon, **Session Bean** icon or **Message-Driven Bean** icon on the EJB Component Palette for the diagram, and then clicking on the diagram where you want to create the element. The implementation files for the modeled elements are created in the location specified by your project settings.

> **Tip:** If you want to model the implementing Java classes for a modeled bean on a diagram, right-click the modeled bean and choose **Show Implementation Files**.

Properties and methods are added by either double-clicking the bean and adding the property or method using the EJB Module Editor or by creating the new property or method 'in-place' on the modeled bean itself. For more information, see Section 23.3.5, "Modeling Properties and Methods".

Relationships Between Beans

References can be created from any bean to any other bean with a remote interface using the **EJB Reference** icon and local references can be created from any bean to any other bean with a local interface using the **EJB Local Reference** icon on the EJB Component Palette for the diagram. For more information, see Section 23.3.6, "How to Model Cross Component References".

A variety of relationships can be created quickly between modeled entity beans using the **1 to * Relationship** icon, **Directed 1 to 1 Relationship** icon, **Directed 1 to * Relationship** and **Directed Strong Aggregation** icons on the EJB Component Palette for the diagram. For more information, see Section 23.3.3, "How to Model a JPA Relationship".

Note: If you change, add to, or delete from, the implementation files for anything that is displayed on a diagram, those changes will be reflected on modeled representations of those elements. Conversely, any changes to the modeled Enterprise JavaBeans are also made to the underlying implementation files.

You can annotate a diagram of Enterprise JavaBeans using notes, dependency relationships and URL links.

23.3.1 Creating a Diagram of EJB/JPA Components

The relationships between entity, session, and message-driven beans, and their properties and methods, can all be modeled graphically on an EJB diagram. EJB diagrams are also used for modeling UML classes, Java classes, UML use cases, offline database objects, business components and web services.

To create a diagram of Java classes:

1. Create a new EJB diagram in a project or application in the New Gallery.
2. Create the elements for the diagram using the EJB Component Palette. Table 23–1 shows the EJB Component Palette.

Table 23–1 EJB Component Palette Icons

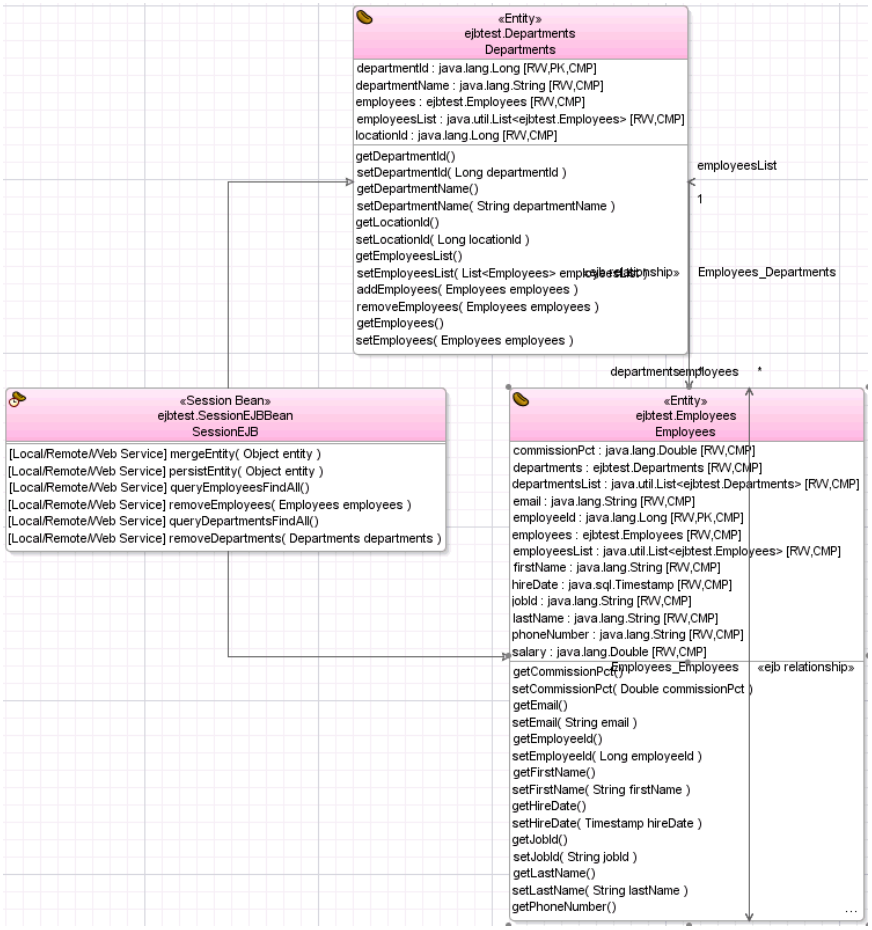
Drop-down List	Icon	Name
EJB Nodes		Entity
		Message-driven Bean
		Session Bean
Entity Relationships		Bidirectional * to * Relationship
		Bidirectional * to 1 Relationship
		Bidirectional 1 to 1 Relationship
		Unidirectional * to * Relationship
		Unidirectional * to 1 Relationship
		Unidirectional 1 to * Relationship
		Unidirectional 1 to 1 Relationship
EJB Edges		Entity Inheritence Edge
		Session Facade Edge
Diagram Annotations		Attachment
		Group
		Link
		Note

You can annotate your diagram by attaching notes to diagram elements. For more information, see Chapter 22.4, "How to Work with Diagram Annotations."

23.3.2 How to Read an EJB/JPA Components Diagram

You can model session, entity and message-driven beans in JDeveloper. Modeled session and entity beans are made up of several compartments. For example, Message-driven beans have only a name compartment containing the `«message-driven bean»` stereotype and the name of the bean. For EJB 3.0 beans the model looks different because there are no compartments for interfaces.

Figure 23–1 EJB/JPA Components Diagram



Notice the relationship and connectors between the beans. References can be created from any bean to any other bean with a remote or local interface. References can only be modeled between beans that are inside the current deployment descriptor.

23.3.3 How to Model a JPA Relationship

You can model a relationship between any two entities on a class diagram by dragging the relationship component from the palette. You can also show the inheritance edge between the root and child entity. The entity at each end of a relationship must have container-managed persistence and a local interface.

To model a relationship between two entities on a diagram:

1. Click the icon for the relationship you want to create, from those listed on the EJB Component Palette.

   Notes: The navigability and multiplicity of a relationship end can be changed after it has been created.

   If these icons are not displayed, select **EJB Components** from the dropdown on the Component Palette.

2. Click the entity at the 'owning', or 'from', end of the relationship.

Note: If you want to represent this relationship using a container-managed relationship (CMR) field, this is the bean in which the field will be created.

3. Click the entity bean at the 'to' end of the relationship.
4. Click the relationship line on the diagram, then click the text fields adjacent to the association to enter the relationship name.

Note: To change the multiplicity of a relationship end on the diagram, right-click on the relationship end and choose either **Multiplicity > 1** or **Multiplicity > ***.

23.3.4 How to Model an EJB/JPA Component On a Diagram

Entity, session, and message-driven beans can be modeled using the diagram features in JDeveloper. Properties and methods can also be modeled.

- Entity beans can be either Container-Managed Persistence (CMP) or Bean-Managed Persistence (BMP). Before creating entity beans with bean-managed persistence, you may want to first consider whether you will need to create relationships between those entity beans. Relationships can only be created between entity beans with container-managed persistence.
- Session beans can be have their session type changed on a class diagram by right-clicking on the session bean and choosing **Session Type**, then **Stateful** or **Session Type**, then **Stateless**.
- Message-driven beans are most often used to interact (using EJB References) with session and entity beans.

23.3.5 Modeling Properties and Methods

Properties and methods can be added to modeled EJBs by either double-clicking the bean and adding the property or method using the EJB Module Editor or by creating the new property or method directly on the modeled bean.

23.3.5.1 Creating Properties on Modeled Beans

When creating a property directly on a modeled bean, enter the name and datatype of the property. For example:

```
name : java.lang.String
```

A public (+) visibility symbol is automatically added to the start of the property.

Note: If a property type from the `java.lang` package is entered without a package prefix, for example, `String` or `Long`, a property type prefix of `java.lang.` is automatically added. If no type is given for a property, a default type of '`String`' (`java.lang.String`) is used

23.3.5.2 Creating Methods on Modeled Beans

Both local/remote and local/local home methods can be created on modeled beans on a class diagram. Modeling a method on a bean also creates the corresponding code in the implementing Java class.

When creating a method in-place on a modeled bean, enter the name, and optionally the parameter types and names, and return type of the method. The method return type must be preceded by a colon (:). For example:

```
getName(String CustNumber) : java.lang.String
```

A public (+) visibility symbol is automatically added to the start of the method.

Notes: If a return type from the java.lang package is entered without a package prefix, for example, `String` or `Long`, a return type prefix of `java.lang.` is automatically added to the Java in the method's class.

If no parameter types are provided, the method will be defined with no parameters. If no return type is specified, a default return type of void is used. To change a property of the method, double-click the class on the diagram, or on the Navigator pane, then change the details of the method using the EJB Editor.

23.3.6 How to Model Cross Component References

References can be created between modeled beans on a class diagram.

- EJB References can be created from any bean to any other bean with a remote interface.
- EJB Local References can be created from any bean to any other bean with a local interface.

Note: References can only be made to beans that are inside the current deployment descriptor.

To model a reference between modeled beans:

1. Click the icon for the reference you want to create, from those listed on the EJB Component Palette:
   - EJB Reference
   - EJB Local Reference

   Note: If these icons are not displayed, select **EJB** from the list on the Component Palette.

   ---

2. Click the bean at the 'owning', or 'from', end of the reference.
3. Click the bean at the 'to' end of the reference.

23.3.7 How to Display the Implementing Source Code for a Modeled Bean

Each modeled bean has underlying Java source files that contain the implementation code for that element. These implementation files can be displayed on the diagram as modeled Java classes.

To display a modeled implementing Java class for a modeled bean:

- Select the bean, the Java implementation of which you want to model on the diagram, then choose **Model**, then **Show**, then **Implementation Files**.
- Or, right-click the bean on the diagram, the Java implementation of which you want to model on the diagram, then choose **Show Implementation**, then **Files**.

23.3.8 How to Display the Source Code for a Modeled Bean

The Java source code for a modeled bean can be displayed in the source editor with simple commands on the diagram.

To display the Java source code for a model element:

- Right-click the element on the diagram. Choose **Go to Source**, then choose the source file you want to view.
- Or, select the element on the diagram. Choose **Model** > **Go to Source**.

23.3.9 How to Change the Accessibility of a Property or Method

You can change the accessibility of a property or method on a modeled bean on a diagram with simple commands on a diagram.

To change the accessibility of a property or method:

1. Right-click the property or method you want to change.
2. Choose the required accessibility option from the Accessible from option on the context menu.

   The accessibility options are:

   - Local Interface
   - Remote Interface
   - Local and Remote Interfaces

23.3.10 How to Reverse-Engineer a JPA Entity on a Diagram

Modeled entity beans can be reverse-engineered on a diagram of EJBs from table definitions in your application database connection.

To reverse-engineer a table definition to an entity bean:

1. Open, or create a diagram.
2. Expand the node in the Connections Navigator for your database connection.
3. Expand the user node, then the **Tables** nodes.
4. Click the table, the definition to use to create an entity bean, and drag it to the current diagram.

   To reverse-engineer several tables to entity beans, hold down the **Ctrl** key, select the tables in the navigator and drag these tables to the diagram, then release the **Ctrl** key.
5. Select the EJB version and click **OK**.

23.4 Java Class Diagram

Model your Java EE Java classes and class systems using the Java class diagram features. For more information see Chapter 22.10.1, "How to Work with Class Diagrams."

23.5 Database Diagram

Modeling your database structures gives you a visual view of your database schema and the relationships between the tables, stored in your online or offline database. You can also transform database tables to Java classes and interfaces and vice-versa using the transformation features. For more information on database transformation see Chapter 22.7, "Transforming Java Classes and Interfaces".

23.5.1 How to Work with the Database Modeling Features

With JDeveloper, you can model offline database objects as well as database objects from a live database connection. You can also create database objects such as tables and foreign key relationships right on your diagram and integrate them with an online or offline database.

Use database diagrams to view your business entity structure and relationships, as well as create directly on your diagram components such as tables and foreign key relationships, views and join objects, materialized views, synonyms and sequences.

All of the database objects from online or offline databases, as well as the new objects you create, are displayed in the Application Navigator.

23.5.1.1 Benefits of Database Modeling

Add or create offline and live online database objects using the Application Navigator and the Component Palette.

Use a database diagram to do the following:

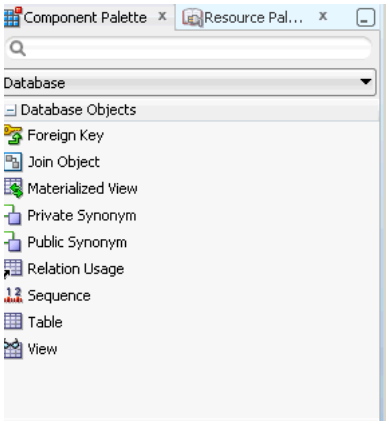
- Create and visualize tables and their columns.
- Create and visualize foreign key relationships between tables.
- Create and visualize views, materialized views, and materialized view logs.
- Create and visualize sequences and synonyms.

In addition, you can create database objects on a diagram based on templates.

23.5.1.2 How to Get Started with Database Modeling

Create your database using the New Gallery. See "To create a new diagram:" on page 22-2. Once you've created your database diagram, you can choose from the available components in the Component Palette, as shown in Figure 23–2.

Figure 23–2 Database Component Palette



Create an offline database object on the diagram by clicking on the icon on the Database Objects Component Palette, and then clicking on the diagram where you want to create the object. You can also drag objects from a database connection in the Database Navigator, or from an offline schema in the Application Navigator, onto a diagram.

You can model online or offline tables on any type of diagram, (except an activity diagram), and you can add elements like UML classes, Java classes, business components, Enterprise JavaBeans, UML use cases, and web services on the same diagram.

You can create offline database objects on a diagram that are based on templates. The first time you add an object to a new database diagram, the Specify Location dialog appears, where you specify whether the objects are offline database objects in a project, or database objects in a database connection. If you want to create offline database objects based on templates, you can create a set of templates at the same time.

You can add the elements that define the object by either double-clicking the modeled object to display the appropriate edit dialog, or by creating the new element 'in-place' on the modeled object.

You can annotate a diagram of database objects using notes, dependency relationships, and URL links.

If you change, add to or delete from the definition of anything that's displayed on a diagram, those changes will be reflected on the modeled representations of those database objects. Conversely, any changes to the modeled database objects are also made to the underlying definitions.

Foreign keys

Foreign keys can be created quickly between modeled tables clicking Foreign Key on the Database Component Palette, then clicking the table you want to originate the foreign key, and then clicking the destination table for the foreign key. The Create Foreign Key dialog allows you to select an existing column in the target table, or create a new column.

Join Objects

You can create join objects between two table usages in a view by clicking (Join Objects) on the Database Component Palette, then clicking on the two table usages to be joined. The Edit Join dialog allows you to specify the join.

Materialized Views

Materialized Views are created on a diagram by clicking on (Materialized View) on the Database Component Palette, and then clicking on the diagram where you want to create the materialized view. You can also drag materialized views from a database connection defined in JDeveloper, or from an offline database in the Application Navigator, and drop them on the diagram.

Materialized Views from Template

Materialized Views based on templates are created on a diagram by clicking on (Materialized View from Template) on the Database Component Palette, and then clicking on the diagram where you want to create the materialized view. The Choose Template Object dialog is displayed, which allows you to choose the template you want to base the materialized view on.

Private and Public Synonyms

Synonyms are created on a diagram by clicking on (Synonym) on the Database Component Palette, and then clicking on the diagram where you want to create the synonym. You can also drag synonyms from a database connection defined in JDeveloper, or from an offline database in the Application Navigator, and drop them on the diagram.

Public synonyms are created in the PUBLIC schema.

Relation Usage

Define a base relation for a view by clicking on (Relation Usage) on the Database Component Palette, and then clicking on the view.

Sequences

Sequences are created on a diagram by clicking on (Sequence) on the Database Component Palette for the diagram, and then clicking on the diagram where you want to create the sequence. You can also drag sequences from a database connection defined in JDeveloper, or from an offline database in the Application Navigator, and drop them on the diagram.

Sequences from Templates

Sequences based on templates are created on a diagram by clicking on (Sequence from Template) on the Database Component Palette for the diagram, and then clicking on the diagram where you want to create the sequence. The Choose Template Object dialog is displayed, which allows you to choose the template you want to base the materialized view on.

Synonyms from Templates

Synonyms based on templates are created on a diagram by clicking on (Synonym from template) on the Database Component Palette for the diagram, and then clicking on the diagram where you want to create the synonym. The Choose Template Object dialog is displayed, which allows you to choose the template you want to base the materialized view on.

Tables

Tables are created on a diagram by clicking on (Table) on the Database Component Palette, and then clicking on the diagram where you want to create the table. You can also drag tables from a database connection defined in JDeveloper, or from an offline database in the Application Navigator, and drop them on the diagram.

You can choose to view table column icons on a modeled table which indicates which columns are primary keys, or foreign keys, or unique keys.

The first column in the modeled table indicates whether the column is in a primary, unique, or foreign key:

- Column is in a primary key
- Column is in a foreign key
- Column is in a unique key
- Column has a check constraint

The second column indicates whether the table column is mandatory.

> **Note:** If a table column is in a primary key it will only display the primary key icon even though it may also be in a unique key or foreign key.

Tables from Templates

Tables are created on a diagram by clicking on (Table from template) on the Database Component Palette, and then clicking on the diagram where you want to create the table. The Choose Template Object dialog is displayed, which allows you to choose the template you want to base the materialized view on.

Views

Views are created on a diagram by clicking on (View) on the Database Component Palette, and then clicking on the diagram where you want to create the view. You can also drag views from a database connection defined in JDeveloper, or from an offline database in the Application Navigator, and drop them on the diagram.

Define the view by adding tables and views, or table columns or elements of other views to the newly defined view. You can add the views by creating them from the Component Palette, by dragging other views and tables on the diagram onto the view, by dragging offline database objects from the Application Navigator, or by dragging database objects from a connection in the Database Navigator.

Views from Templates

Views are created on a diagram by clicking on (View from template) on the Database Component Palette, and then clicking on the diagram where you want to create the view. The Choose Template Object dialog is displayed, which allows you to choose the template you want to base the materialized view on.

23.5.1.3 How to Change the Database or Schema

1. On the database diagram, right-click and choose **Create Database Objects In > Database or Schema**.
2. Complete the Specify Location dialog (selecting Application Project for offline database objects, or Database Connection for database objects) or Select Offline Schema dialog.

> **Note:** All subsequent database objects will be created in the database or schema you have chosen. Existing objects are unchanged.

Part VI

Working with Databases

This part describes the general concepts of working with Oracle JDeveloper to connect to and design databases and contains the following chapters:

- Chapter 24, "Getting Started with Working with Databases"

  This chapter introduces the various concepts and features.

- Chapter 25, "Using the Database Tools"

  This chapter introduces the tools and features that JDeveloper provides to help you to design and work with databases.

- Chapter 26, "Connecting to and Working with Databases"

  This chapter introduces connecting to Oracle and non-Oracle databases.

- Chapter 27, "Designing Databases Within Oracle JDeveloper"

  This chapter introduces designing databases in JDeveloper by modeling database objects, and how to generate these to a database.

- Chapter 28, "Using Java in the Database"

  This chapter describes the features that allow you to write and execute Java programs that access Oracle Databases.

- Chapter 29, "Running and Debugging PL/SQL and Java Stored Procedures"

  This chapter how to use PL/SQL and Java Stored Procedures in a database.

24

Getting Started with Working with Databases

This chapter describes how to get started using JDeveloper to work with databases. If you are new to using database you will find the section on connecting to Oracle Database XE, an entry-level, small-footprint database.

This chapter includes the following sections:

24.1 About Working with Databases

JDeveloper enables you to work with Oracle and non-Oracle databases directly, and to design, create, and edit databases by working with offline database definitions.

Refer to the following documentation to quickly get started with using Oracle databases in JDeveloper:

- Using Oracle Database 10g Express Edition. For more information, see Section 24.2, "Getting Started With Oracle Database 10g Express Edition."
- Creating connections to Oracle and non-Oracle databases. For more information, see Section 26.4, "Connecting to Databases."
- Working in the Database Navigator. For more information, see Section 25.1, "Using the Database Navigator."
- Database Development with JDeveloper. For more information, see "Database Development with JDeveloper 11*g*" at `http://st-curriculum.oracle.com/obe/jdev/obe11jdev/11/index.html`.

24.1.1 Connecting to and Working with Databases

Usually you start working with a database by creating a connection to it. JDeveloper helps you quickly to create connections to Oracle databases, and you can also connect to and work with a number of non-Oracle databases. Once you have a database connection you can search for database objects in the Database Navigator, or use the search tools to find specific objects, or compare databases and their contents. You can also edit data and import and export data, and you can create reports about the database and objects in it.

24.1.2 Designing Databases

You can work directly with databases through a database connection using the integrated tools in JDeveloper which include SQL Worksheet and the database object editors, and you can edit database objects using the database object editors. Alternatively, you can create an offline database and working either in the Application Navigator or the database diagrammer you can work with offline database definitions to model the database and then generate the results to a database through a database connection.

Database connections can be listed in the Application Navigator or Database Navigator, where they are available to applications you are working on, or in the Resource Palette, where they are available for reuse in other applications.

Once you have a database connection, you can:

- Browse and search databases for specific objects.
- Produce reports about databases and their contents.
- Import and export data.
- Copy, compare and export databases.

You can work with offline databases, which you can model on the database diagrammer or work with in the Application Navigator.

You can create, edit, and drop objects in a database or in an offline database.

You can write and execute Java programs using JDBC that access Oracle and non-Oracle databases.

If you are new to using databases with JDeveloper, one of the easiest ways to get started is to try out Oracle Database 10g Express Edition (Oracle Database XE).

24.2 Getting Started With Oracle Database 10*g* Express Edition

If you are new to using databases with JDeveloper, one of the easiest ways to get started is to try out Oracle Database 10*g* Express Edition (Oracle Database XE). Oracle Database XE is an entry-level, small-footprint database based on the Oracle Database 10g Release 2 code base. It is free to develop, deploy, and distribute; fast to download; and simple to administer. You can download it from Getting Started: Oracle Database 10g Express Edition (XE), which is available at `http://www.oracle.com/technetwork/database/express-edition/overview/index.html`

After you have downloaded and installed Oracle Database XE, follow the instructions in the Getting Started Guide to unlock the sample user, HR. You may want to grant additional privileges, for example to create tables and materialized views. Now you can create a database connection from JDeveloper to the sample user.

In the Create Database Connection dialog, use the following values. Leave blank any fields that are not mentioned.

Table 24–1 Connection details for Oracle Database 10g Express Edition

Field	Value
Create Connection In	Choose **Application resources**. The connection will be displayed in the Application Navigator, under Application Resources.
Connection Name	Enter a meaningful name for this connection.
Connection Type	`Oracle (JDBC)` (default).

Table 24–1 (Cont.) Connection details for Oracle Database 10g Express Edition

Field	Value
Username	`HR` to use the sample user. If you have created a new database user, enter the name of that user.
Password	Enter the password you entered when you unlocked the sample user or created a new user.
Save Password	Selected (default).
Driver	`thin` (default)
Host Name	When Oracle Database XE is installed on the local system use the default of `localhost` or `127.0.0.1`. Otherwise enter the IP address or resolvable hostname of the machine where it is installed.
JDBC Port	`1521` (default)
SID	`XE` (default)

Click **Test Connection** at the bottom of the dialog. `Success!` indicates that you have a connection to the database. If you get any other message, check that you have entered the values above correctly, and check that the Oracle Database XE has started.

24.3 How to Manage Database Preferences and Properties

There are a number of preferences that allows you to control how to use the database functionality in JDeveloper. These are available in the Preferences dialog, available from the **Tools** menu:

- Database page, where you can choose not to have date and time default values validated, set the default path for export DDL files, and enter the location of a database startup script.
- Database: Advanced page, where you set options such as the SQL array fetch size and display options for null values.
- Database: Autotrace/Explain Plan page, where you specify the information to be displayed on the Autotrace and Explain Plan pages in the SQL Worksheet.
- Database: Drag and Drop page, where you specify the type of SQL statement created in the SQL Worksheet when you drag an object from the Database Navigator into the SQL Worksheet.
- Database: JDBC Driver Options page, where you register and manage JDBC drivers for the BI JDBC driver, and the WebLogic JDBC drivers for DB2, Informix, SQL Server and Sybase.
- Database: NLS page, where you specify globalization support parameters, such as the language, territory, sort preference, and date format.
- Database: Objectives options page, where you specify whether to freeze object viewer windows, and display options for the output.
- Database: PL/SQL Compiler page, where you specify options for compilation of PL/SQL subprograms.
- Database: Reports page, where you can choose that database reports in JDeveloper are closed when the database is disconnected.
- Database: SQL*Plus, where you set the path to the SQL*Plus command line tool.

- Database: SQL Editor Code Templates page, which allows you to view, add, and remove templates for editing SQL and PL/SQL code. Code templates assist you in writing code more quickly and efficiently by inserting text for commonly used statements.
- Database: SQL Formatter page, which allows you to control how statements in the SQL Worksheet are formatted.
- Database: User Defined Extensions page. (Not used by JDeveloper.)
- Database: Worksheet page, where you specify options for the SQL Worksheet.
- Diagrams: Database (under the Diagrams node). Use to set preferences that control how diagrams are displayed.

To manage database preferences in the Preferences dialog:

1. Choose **Tools > Preferences**.
2. From the Preferences page, select the page you want. For more information at any time, press F1 or click **Help** from within the dialog.

To manage properties in the Project Properties dialog:

1. Choose **Application > Project Properties** (to change or specify a property for just the current project), or **Default Project Properties** (to set default properties).
2. In the dialog, choose the page you want. For more information at any time, press F1 or click **Help** from within the dialog.

As well as managing these preferences and properties, you can also filter schemas or objects in a database connection to just see the ones you want.

25

Using the Database Tools

This chapter provides an introduction to the various tools that JDeveloper uses to help you work with and manage databases.

This chapter includes the following sections:

25.1 Using the Database Navigator

The Database Navigator provides you with a complete editing environment for online databases. You can create, update and delete database objects using the navigator.

The Database Navigator is integrated with:

- The SQL Worksheet.
- The Database Object Viewer.

In addition, you can drag database objects from a database connection onto a database diagram to either:

- Model the database objects on the diagram.
- Copy the database objects to a project, and model the offline database objects on the diagram.

For more information about database modeling, see Section 23.5, "Database Diagram."

When you first open the Database Navigator, it appears in the docked position, along with any other open navigators. Its default docked position is in the upper left-hand

corner, flush with the main work area of JDeveloper. When more than one navigator is open, each appears with a tab displaying its name.

Right-click on a node within the Database Navigator to bring up a context-sensitive menu of commands. The menu commands available depend on the node selected. You can open nodes in their default editors, as well as other editors common to that node type, using the context menu.

Table 25–1 Database Navigator Toolbar Icons

Icon	Name	Function
	**New Connection**	Click to open the Create Database Connection wizard, where you enter the details to create a connection to a database.
	**Refresh**	Click to synchronize the display in the navigator with the contents of the connection.
	**Apply Filter**	Click to filter which objects will be displayed for a given connection. To enable the icon, select a node within the connection in the navigator and wait for the connection to be established.

You can perform various tasks from the context menus in the Database Navigator. Right-click the **IDE Connections** node (for globally defined connections) or an application name node (for connections that are locally-scoped, and just available within the application) and select the appropriate menu item to:

- Open the New Gallery.
- Create a new database connection.
- Import an XML file with connection definitions.
- Export current connections.

You can perform the following operations from a database connection node:

- Connect to and disconnect from the database.
- Open the New Gallery.
- Delete the database connection.
- Generate SQL from database objects.
- Copy database objects as offline database objects to a project.
- Run SQL*Plus.
- Filter the objects displayed in the connection.
- Edit the database connection properties.
- Open the SQL Worksheet.
- Generate DB doc
- Remote Debug
- Gather Schema Statistics
- XML DB Protocol server configuration
- Perform remote debugging if you are using the Java Platform Debugger Architecture (JPDA) using a debugger to listen so that a debuggee can attach to the debugger.

There are additional options available from database object type nodes (for example, Tables, Indexes, or Procedures) or from database object nodes (such as a specific table, or a specific view). The options available depend on the node selected.

25.2 Using the Structure Window

The database view of the Structure Window displays details of a database object selected in the Database Navigator, or an offline database object selected in the Application Navigator.

When you select a database object such as a table in a database connection in the Database Navigator or an offline database object such as a table in an offline database in the Application Navigator, a node for that object is shown in the Structure Window. You can expand the node to see details of the sub objects that make up the database object. In the case of a table, these include sub objects such as columns, constraints, and indexes.

Table 25–2 Icons in the SQL Worksheet Toolbar

Icon	Name	Function
	**Refresh**	Click to refresh the content of the Structure Window. You may want to do this when a database object has been changed outside JDeveloper, for example using SQL*Plus, and you want to be sure that the Structure Window reflects the current state of the object.

You can perform the following operations from the database view of the Structure Window:

- View properties or edit properties (offline database objects only) by choosing **Properties** from the context menu of an appropriate node. The Edit dialog for the object type opens. It is read only for database objects.
- Use a database object or offline database object such as a table as a template to create a new object by choosing **Use as Template** from the context menu. The Create dialog for the object type opens.
- Find usages of an offline database object such as a table by choosing **Find Usages** from the context menu.

25.3 Using the Database Reports Navigator

Use the Database Reports Navigator to view reports about the database and its objects.

You can also create your own user defined reports.

To open a pre-defined report, expand Data Dictionary Reports and navigate to the report you want. Double-click the report name to run it. A number of dialogs may be displayed before the report is opened in the Reports Results window:

- Select Connection dialog (all reports), where you can choose an existing database connection or create a new database connection. Once you have chosen the connection, the same connection is used for subsequent reports you run.
- Enter Bind Values dialog (All Objects reports), where you can enter values for each bind variable. Bind variables enable you to restrict the output.
- Diagnostic Pack Required dialog (ASH and AWR reports). You must have a licensed copy of Oracle Diagnostic Pack running on the database to run these reports, and the dialog allows you to confirm that you have one.

You can create your own reports and store them in folders and sub-folders under the User Defined Reports node.

Some reports may take some time to run, and the time is affected by the number and complexity of objects involved, and by the speed of the network connection to the database.

From the Data Dictionary Reports node you can:

- Export a report into an XML file that can be imported later by right-clicking the report name and choosing Export.
- Create a shared report from an exported report.

User Defined reports are any reports that are created by JDeveloper users.

Information about user defined reports, including any folders for these reports, is stored in `UserReports.xml` in the directory for user-specific information.

You can perform the following operations from the User Defined Reports node:

- Create a user defined report by choosing Add Report from the User Defined Reports context menu.
- Organize user defined reports in folders, and create a hierarchy of folders and subfolders. Choose Add Folder from the User Defined Reports context menu.
- Import a report that had previously been exported. Select report folder in which to store the imported report, right-click, and select Import.

The Shared Reports node is displayed once you have defined the first shared report in the Preferences dialog.

For more information about creating and sharing database reports, see Section 26.8, "Working with Database Reports".

25.4 Using the Find Database Object Window

The Find Database Object Window allows you to search for and work on database objects within a live database.

The Find Database Object Window is fully integrated with the online database functionality, including the SQL Worksheet and the Database Object Viewer.

While you are using the Find Database Object Window these features are available:

- Open any currently closed navigator, or bring a currently open navigator to the foreground, using **View > navigator-name**.
- Move, resize, float, minimize, maximize, restore or close the Find Database Object Window using the context menu available by right-clicking its tab or by pressing Alt+Minus.

Table 25–3 Find Database Object Toolbar

Name	Function
Connection	Choose the database connection to search in from the dropdown list. You must already have a connection to the database.
Name	Enter the search term. You can use the wildcard `%` to return a number of matching objects.
Type	Choose the type of database object to restrict the search to. The default is `ALL OBJECTS`.

Table 25–3 (Cont.) Find Database Object Toolbar

Name	Function
Usage	Only for certain types of object) Choose the usage of the object, for example `ALL`.
Lookup	Click to display the results of the search. The results of the search are displayed in the panel. Double-click on an object to open it in the appropriate editor.

You can perform the following tasks from the Find Database Object window:

- Close or open the panel by clicking its bar.
- Change the area used by the panel by grabbing its bar and moving it up or down.
- Remove the panel from view by opening its dropdown menu (panel bar, far right) and choosing **Minimize**. Restore it by clicking the three dots at the very bottom of the Application Navigator and then clicking **Recently Opened Files**.
- Open an object, or the parent object that contains the specified object, in its default editor, or bring the default editor into focus, by selecting the object in the list.

25.5 Using the SQL Worksheet

Use to enter and execute SQL, PL/SQL, and SQL*Plus statements. You can specify any actions that can be processed by the database connection associated with the worksheet, such as creating a table, inserting data, creating and editing a trigger, selecting data from a table, and saving that data to a file.

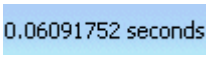
You enter SQL statements in the SQL Statement area, and use the buttons on the toolbar to perform actions.

Table 25–4 describes the icons and fields in the toolbar above the SQL Worksheet statement area.

Table 25–4 Icons in the SQL Worksheet Toolbar

Icon	Name	Function
	**Run Statement** (Ctrl+Enter)	Click to execute the statement at the mouse pointer in the SQL statement area. The SQL statements can include bind variables and substitution variables of type `VARCHAR2`. If necessary, `VARCHAR2` is automatically converted to `NUMBER`. If you use variable values, a window is displayed for you to enter them.
	**Run Script** (F5)	Click to execute all statements in the SQL statement area. The SQL statements can include bind variables and substitution variables of type `VARCHAR2`. If necessary, `VARCHAR2` is automatically converted to `NUMBER`. If you use variable values, a window is displayed for you to enter them.
	**Autotrace** (F6)	Click to generate trace information for the statement. Too see trace information, click the **Autotrace** tab.
	**Explain Plan** (F10)	Click to generate the execution plan for the statement, which internally executes the `EXPLAIN PLAN` statement. Trace information is shown in the Explain Plan Results window.
	**Commit** (F11)	Click to write any changes to the database. This ends the transaction and clears any output in the Results and Script Output tabs.

Table 25–4 (Cont.) Icons in the SQL Worksheet Toolbar

Icon	Name	Function
	**Rollback** (F12)	Click to discard any changes without writing them to the database. This ends the transaction and clears any output in the Results and Script Output tabs.
	**Unshared SQL Worksheet** (Ctrl+Shift+N)	Click to open a new unshared SQL Worksheet for a different connection.
	**To Upper/Lower/ Inicat** (Ctrl+Quote)	Click to switch the selected text between upper case, lower case, and initial capitals.
	**Clear** (Ctrl+D)	Click to erase the statement or statements in the **Enter SQL Statement** area.
	**Cancel**	(Only displayed while a script is running) Click to stop execution of the script.
		(Only displayed once a statement or script has run) Displays the time it took to execute a statement or run a script. This can be used with Explain Plan to provide useful tuning information.
		Use to choose a different database connection.

The results area has a number of tabs:

- Results tab, Displays the results of clicking **Run Statement**.
- Script Output tab, which displays the results of clicking **Run Script**.
- Autotrace tab, which displays output as a result of clicking **Autotrace**.
- Explain tab, which displays output as a result of clicking **Explain Plan**.

The SQL Worksheet provides code insight for SQL code. When you type a word, a dropdown menu of valid code appears. For example:

- If you type `select`, `SELECT` is displayed.
- If you type `select *`, a list containing **BULK, FROM, and INTO** is displayed.
- If you are connected to the HR schema and type `select * from em`, a list containing the table `employees` and the view `emp_details_view` is displayed.

To configure Code Insight for the SQL Worksheet:

1. Select **Tools > Preferences > Code Editor > Code Insight**.
2. In the Code Insight page, adjust font size or font type, and completion insight and parameter insight timing.
3. Click **OK**. Your changes are active the next time you use the editor.

To open the SQL Worksheet:

1. Choose **View > Database > Database Navigator**.
2. Expand IDE Connections or Application Connections.
3. Right-click the connection in the Navigator, and choose **Open SQL Worksheet**.

Alternatively, click the **SQL Worksheet button** on the JDeveloper toolbar.

For more information at any time, press F1 or choose **Help** from within the SQL Worksheet.

Alternatively, from the main toolbar, click and choose the database connection from the Choose Connection dialog.

You can create a `SELECT` statement by dragging and dropping table and view names, and by graphically specifying columns and other elements of the query using Query Builder. You can run the statement within Query Builder to see the results, and when you close Query Builder, the resulting `SELECT` statement is inserted into the SQL Worksheet.

To use Query Builder:

1. Open the SQL Worksheet.
2. Right-click and choose **Query Builder**. For more information at any time, press F1 or click **Help** from within the Query Builder.
3. Select the schema you want, and drag the table you want to base the query on onto the main pane of the dialog. There will be a delay of a few seconds while Query Builder connects to the database and loads information about the table.
4. Choose the columns you want to use in the query from the dialog that is displayed.
5. To add a `WHERE` clause, click the **Create Where Clause** tab, and enter values for the clause.
6. View the SQL comprising the query in the **View SQL** tab.
7. You can see the results of the query by selecting the **View Results** tab, and clicking **Run Results**.
8. When you press **Apply** in the Query Builder dialog, the dialog closes and the query is inserted into the SQL Worksheet.

To execute a SQL statement:

1. Enter a SQL statement in the worksheet's upper pane.
2. Do any one of the following:
   - Press Ctrl+Enter.
   - Click the **Execute the statement button** on the toolbar.
   - Right-click, and select **Execute SQL Statement** from the context menu.
3. View the data returned by the statement in the lower pane.

For more information, see "Using the SQL Worksheet" in the *Oracle Database SQLJ Developer's Guide*.

25.5.1 Using Execution Plan

An execution plan is the sequence of operations that will be performed to execute the statement, and you can use the SQL Worksheet to inspect the execution plans chosen by the Oracle optimizer for `SQL SELECT`, `UPDATE`, `INSERT`, and `DELETE` statements. You can also view explain plan for the SQL code for the query part of a view definition.

An execution plan shows a row source tree, which is the hierarchy of operations that comprise the statement. For each operation it shows the following information:

- An ordering of the tables referenced by the statement
- An access method for each table mentioned in the statement
- A join method for tables affected by join operations in the statement
- Data operations such as filter, sort, or aggregation

In addition to the row source tree, the plan table displays the following information about selected operations:

- Optimization, such as the cost and cardinality of each operation
- Partitioning, such as the set of accessed partitions
- Parallel execution, such as the distribution method of join inputs

For more information, see "Using EXPLAIN PLAN" in the *Oracle Database Performance Tuning Guide*.

An additional source of information that can be used to tune SQL queries is the elapsed time that is displayed in the toolbar of the SQL Worksheet when statements are executed or scripts are run.

To view a SQL statement's execution plan:

1. If necessary, open the SQL Worksheet.
2. Enter a SQL statement in the worksheet's upper pane.
3. Do one of the following:
   - Click the **Explain Plan button** on the toolbar.
   - Right-click to open the context menu, and select **Execute Explain Plan**.

   The Explain Plan tab shows the explain plan information for the SQL statement.

25.5.2 How to Recall Statements from the SQL Worksheet History

The statements executed in a session with the SQL Worksheet are preserved in a history list. You can retrieve previous statements from the history, and re-execute them or view their execution plans.

To recall a statement from the SQL Worksheet history:

1. Do either of the following:
   - Click the **View History dialog** button.
   - Right-click in the SQL Worksheet to open the context menu, and select **History**.

   A dialog showing the list of the statements previously entered is displayed.
2. Select the desired statement from the dialog.
3. Click **OK**.

The statement is displayed in the upper pane of the worksheet.

25.6 Using the SQL History Window

The SQL History Window allows you to reuse statements previously executed in a session with the SQL Worksheet.

SQL statements and scripts that you have executed are listed in the window, and you can select one or more statements to have them either replace the statements currently on the SQL Worksheet or be added to the statements currently on the SQL Worksheet.

While you are using the SQL History Window these features are available:

- Open any currently closed navigator, or bring a currently open navigator to the foreground, by choosing it from the View menu.
- Move, size, float, minimize, maximize, restore or close the Find Database Object Window using the context menu available by right-clicking its tab or by pressing Alt+Minus.

Table 25–5 SQL History Toolbar Icons

Icon	Name	Function
	**Append**	Click to append the selected statement or statements to any statements currently on the SQL Worksheet. You can also append the selected statement or statements by dragging them from the SQL History window and dropping them at the desired location on the SQL Worksheet.
	**Replace**	Click to replace any statements currently on the SQL Worksheet with the selected statement or statements.
	**Clear History**	Click to remove all statements from the SQL history.
	**Filter**	Use to filter the SQL statements visible in the SQL History window. Type a string in the text box and click **Filter**. Only SQL statements containing that string are listed. To remove the filter, delete the string in the field and click **Filter** again.

25.7 Using the Snippets Window

Snippets are code fragments, such as SQL functions, Optimizer hints, and miscellaneous PL/SQL programming techniques. Some snippets are just syntax, and others are examples. The Snippets Window is integrated with the SQL Worksheet and when you are creating or editing a PL/SQL function or procedure.

In the Snippets Window, the snippets are organized in categories in the drop-down list, such as Aggregate Functions or Character Functions. You can create new snippets and add them to an existing category, or to a new category. To see a brief description of a snippet, hover the mouse pointer over the function name.

To insert a snippet into your code in a SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the snippets window and drop it into the desired place in your code; then edit the syntax so that the SQL function is valid in the current context.

For example, you could type `SELECT` and then drag `CONCAT(char1, char2)` from the **Character Functions** group. Then, edit the CONCAT function syntax and type the rest, as in this example:

```
SELECT CONCAT(title, ' is a book in the library.') FROM books;
```

Table 25–6 Snippets Window Toolbar Icons

Icon	Name	Function
	**Add Snippets**	Click to open the Save Snippet dialog where you can create a new snippet and save it in an existing group or in a new group.
	**Edit User Snippets**	Click to open the Edit Snippets dialog which lists user snippets. You can create, edit and delete snippets in this dialog.

From the Snippets window, you can:

- Display snippets by choosing the category from the list.
- Add a snippet to the cursor position in a file such as a SQL file by double-clicking it.

25.8 Using the Database Object Viewer

The Database Object Viewer allows you to manage the structure and contents of objects in a database. The tabs available depend on the type of object being viewed.

You can edit the value in any of the cells by double-clicking the cell to select it, then clicking **...** to open the Edit Value dialog.

Information about the object is contained in a number of tabs.

- Columns, which shows the columns comprising the object.
- Data, which shows the data in this object. You can edit the value in any of the cells by double-clicking the cell to select it, then clicking **...** to open the Edit Value dialog.
- Constraints, which shows the details of any constraints.
- Grants, which shows privilege details.
- Statistics, which shows statistical information.
- Triggers, which shows information about triggers
- Dependencies, which shows information about references.
- Details, which Indexes, which displays details of any indexes.
- SQL, which displays the SQL that represents this object.

25.8.1 Database Object Viewer Tabs Toolbars

The specific buttons on the toolbar vary from tab to tab.

Table 25–7 Database Object Viewer Tabs Toolbar Icons

Icon	Name	Function
	**Freeze and Unfreeze View**	Use to toggle freezing the table viewer on the current view.
	**Edit**	Click to open the Edit Table dialog.
	**Refresh**	Click to refresh the data.

Table 25–7 (Cont.) Database Object Viewer Tabs Toolbar Icons

Icon	Name	Function
	**Insert Row**	Click to insert a new blank row below the row where the focus is.
	**Delete Selected Row(s)**	Click to delete the selected rows of data.
	**Commit**	Click to commit the changes to the database. The changes are logged in the Data Editor log window, and commit will fail if there is an error, such as a unique constraint violation.
	**Rollback**	Click to rollback database changes already made. The Data Editor log window reports on whether the rollback has succeeded.
Sort...		Click to open the Sort dialog where you specify the columns to sort by and the sort order.
Filter		Enter a value to reduce the number of records displayed, for example `DEPARTMENT_ID>20`.
Actions...		Click to perform one of a range of common table actions.

25.9 Using SQL*Plus

SQL*Plus is an interactive and batch query tool that is installed with every Oracle Server or Client installation. It has a command-line user interface. You can launch SQL*Plus from within JDeveloper. For more information, see the *SQL*Plus User's Guide and Reference*.

In most cases, using SQL Worksheet is preferable to using SQL*Plus as it is fully integrated with JDeveloper, and you can use SQL Worksheet to enter and execute SQL, PL/SQL, and some SQL*Plus statements.

SQL*Plus can use parameter substitution. The default escape character is '&', thus any comments that have '&' in them may cause an error. Additionally, the character used in the SQL*Plus session that runs the script can be changed from the default using SET DEFINE, so JDeveloper cannot look for the parameter substitution character in comments and warn you. If you encounter this error in a script, you can use SET DEFINE OFF to ignore the parameter substitution character or remove the character from the comment. For more information, see "'Using Scripts in SQL*Plus'" in the *SQL*Plus User's Guide and Reference*.

In order to launch SQL*Plus from JDeveloper, you must have SQL*Plus installed on your machine. For information about installing a SQL*Plus client, see the information about Oracle Database Instant Client at `http://www.oracle.com/technetwork/database/features/instant-client/index-100365.html`.

You can launch SQL*Plus from:

- The Tools menu
- A database connection in the Database Navigator
- A SQL file in the Application Navigator

If you have not already specified the SQL*Plus executable in JDeveloper, you will able to do so when you launch SQL*Plus. Alternatively, you can specify the SQL*Plus executable in the Preferences dialog. You only need to perform this task once.

To specify the SQL*Plus executable:

1. Choose **Tools > Preferences**, and select Database Connections.
2. Specify the path to the SQL*Plus executable.
3. Click **OK** to close the dialog. Now the SQL*Plus item is active in the Tools menu.
4. Select a database connection in the Database Navigator, then choose **Tools > Database > SQL*Plus**. If the path specified in step 2 is correct, a SQL *Plus command window will open.

> **Note:** On Unix, use xterm to create a terminal window to run the SQL*Plus command in.

To launch SQL*Plus from a connection:

1. Choose **View > Database > Database Navigator**.
2. Right-click the connection, and choose **SQL*Plus**.

To launch SQL*Plus from a SQL file:

1. In the Application Navigator, navigate to a SQL file.
2. Right-click the SQL file, and choose **Run in SQL*Plus**.
3. In the submenu, select the connection you wish to use. If you have not already specified the location of the SQL *Plus executable, you will be prompted for that first.

25.10 DBMS Output Window

The PL/SQL DBMS_OUTPUT package enables you to send messages from stored procedures, packages, and triggers. The `PUT` and `PUT_LINE` procedures in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the `GET_LINE` procedure. The DBMS Output window is used to display the output of that buffer.

Add New DBMS Output Tab: Prompts you to specify a database connection, after which a tab is opened within the DBMS Output pane for that connection, and the SET SERVEROUTPUT setting is turned on so that any output is displayed in that tab. (To stop displaying output for that connection, close the tab.)

Table 25–8 DBMS Output Window Toolbar Icons

Icon	Name	Function
	**Enable DMBS Output**	Click to toggle the SET SERVEROUTPUT setting between ON and OFF. Setting server output ON checks for any output that is placed in the DBMS_OUTPUT buffer, and any output is displayed in this tab.
	**Clear**	Click to erase the content of this tab.
	**Save File**	Click to open the Save dialog where you can enter a filename to save the results in this tab.
	**Print**	Click to open the Print dialog, where you can choose the printer to print the content of this tab.

Table 25–8 (Cont.) DBMS Output Window Toolbar Icons

Icon	Name	Function
	Buffer Size	For databases before Oracle Database 10.2, click to limit the amount of data that can be stored in the DBMS_OUTPUT buffer. The buffer size can be between 1 and 1000000 (1 million).
	Poll	Move the slider to set the interval (in seconds) at which JDeveloper checks the DBMS_OUTPUT buffer to see if there is data to print. The poll rate can be between 1 and 15.
	Choose DB Connection	Change to a different database connection by choosing it from the list.

25.11 OWA Output Window

OWA (Oracle Web Agent) or MOD_PLSQL is an Apache (Web Server) extension module that enables you to create dynamic Web pages from PL/SQL packages and stored procedures. The OWA Output window enables you to see the HTML output of MOD_PLSQL actions that have been executed in the SQL Worksheet.

Table 25–9 OWA Output Window Toolbar Icons

Icon	Name	Function
	**Enable OWA Output**	Click to toggle the SET SERVEROUTPUT setting between ON and OFF. Setting server output ON checks for any output that is placed in the DBMS_OUTPUT buffer, and any output is displayed in this tab.
	**Clear**	Click to erase the content of this tab.
	**Save File**	Click to open the Save dialog where you can enter a filename to save the results in this tab.
	**Print**	Click to open the Print dialog, where you can choose the printer to print the content of this tab.
	Choose DB Connection	Change to a different database connection by choosing it from the list.

26

Connecting to and Working with Databases

This chapter describes how to create and work with database connections.

This chapter includes the following sections:

26.1 About Connecting to and with Working with Databases

You can connect to and work with Oracle databases and a number of non-Oracle databases.

Database connections can be available in the Application Navigator or Database Navigator, where they are available to applications you are working on, or in the Resource Palette, where they are available for reuse in other applications.

Once you have a database connection, you can:

- Browse for database objects
- Search for specific database objects
- Import and export data
- Copy a database objects from one database schema to another
- Compare one database schema to another
- Export some or all objects of one or more database types to a DLL file
- Use pre-defined reports and create new reports to provide information about a database and its objects

If you are new to using databases with JDeveloper, one of the easiest ways to get started is to try out Oracle Database Express Edition (Oracle Database XE). For more

information, see Section 24.2, "Getting Started With Oracle Database 10g Express Edition."

26.2 Configuring Database Connections

You can define and manage connections to external data sources using the Create Database Connection dialog.

- The Resource Palette, where they can be added to catalogs to facilitate collaborative working or to make them available to more than one application.
- The Database Navigator, where you can create, edit, and modify objects in the database.
- Application Resources in the Application Navigator, where they are available in the current application.

When you delete a connection, JDeveloper does not warn you that a project may be dependent upon it. For this reason, it is best to use caution when deleting connections.

26.2.1 Connection Scope

In JDeveloper 11*g* you have two ways of creating and managing database connections. You can define database connections for an application (called an Application Resource connection) or for the IDE as a whole (called an IDE connection). You use the same dialog to define these, but their scope within JDeveloper is different.

When you first create a database connection, you choose the connection scope, which you cannot subsequently change. For more information, see Section 3.7.2.2, "Defining the Scope of a Connection."

26.2.2 What Happens When You Create a Database Connection

When you create a database connection, JDeveloper creates a node for the connection in the Database Navigator, and an additional node in either the Resource Palette or in the Application Resources panel of the Application Navigator depending on the scope of the connection.

In the Application Navigator and Database Navigator, you can expand the database connection node to view and work with database objects. In the Resource Palette, you can only work with a database connection after you have added it to the application.

Database Connections Created as Application Resources

Database connections created as application resources are only available to the application in which they are created.

In the Database Navigator, the node for the connection is under the node with the same name as the application.

In the Application Navigator, the node for the connection is under Connections in the Applications Resources panel. Connection information is stored in `connections.xml`, which is under the Descriptors node, under ADF META-INF. You can open the file in the XML editor by double-clicking it, and you can discover the file path by hovering the mouse over the filename.

The file system location for the connection descriptor definition information is `application_folder/.adf/META-INF/connections.xml` where `application_folder` is the path for the selected application.

Database Connections Created as IDE Connections

These database connections are globally defined connections.

You can copy an IDE connection to the application navigator to use it in an application by:

- From the Resource Palette, dragging the connection and dropping it on the Connections node in the Application Navigator under Application Resources.
- From the Resource Palette, right-clicking the connection and choosing Add to Application.
- In the Database Navigator, dragging the connection from under the IDE Connections node to the Application Connections node under the node for the application.

The file system location for the connection descriptor definition information is `sys-dir/jdeveloper/system11.1.x.x.nn.nn.nn/o.jdeveloper.rescat2.model/connections/connections.xml`.

26.2.3 About Connection Properties Deployment

A **connections.xml** file is included with JDeveloper deployments, and in the application it is in the folder `.adf\META-INF`. This file contains the connection information necessary for deployment and the runtime connection execution.

26.2.4 How to Create Database Connections

After you have defined a connection, you can return to the dialog and edit its attributes.

> **Note:** You cannot change the connection type after the database connection has been created.

To create a database connection:

1. If necessary, choose **View > Database > Database Navigator**. Right-click **IDE Connections** or application, and choose **New Connection** to open the Create Database Connection dialog.

   Alternatively, from the main menu, choose **File > New** to open the New Gallery. In the Categories list, expand **General** and select **Connections**. In the Items list, double-click **Database Connection** to open the Create Database Connection dialog.

   For more information at any time, press F1 or click **Help** from within the Create Database Connection dialog.

2. Enter the appropriate connection information, then click **Test Connection**. You may have to briefly wait while JDeveloper connects to the database.

   If the test succeeds, a success message appears in the status text area. If the test does not succeed, an error appears. In this case, change any previously entered information as needed to correct the error, or check the error content to determine other possible sources of the error.

26.2.5 Connecting to Oracle Database Using OCI8

The recommended way of connecting to Oracle Database is using the thin driver, however you can connect using OCI8 (thick connection).

To connect using OCI8:

- Define the jar location using the system property `oracle.jdbc.library`. For example:

```
jdev -J-Doracle.jdbc.library=/jdev_install/jdeveloper/ojdbc6.jar
```

26.2.6 How to Edit Database Connections

To edit a database connection:

1. Choose **View > Database > Database Navigator**.
2. Expand IDE Connections or application, and select a database connection.
3. Right-click the connection and choose Properties to open the Edit Database Connection. For more information at any time, press F1 or click **Help** from within the Create Database Connection dialog.

Note: You can filter which schemas appear in the connection.

26.2.7 How to Export and import Database Connections

You can also import and export database connections created as IDE connections in the Resource Palette. For more information, see Section 3.7.6, "How to Import and Export Catalogs and Connections."

26.2.7.1 Exporting Database Connections

When you export connections, selected connection descriptors are copied to an XML file. The file can be imported by other users to easily create connections.

To export a database connection:

1. Choose **View > Database > Database Navigator**.
2. Right-click either **IDE Connections** or **application** and choose **Export Connections**.
3. In the Export Connection Descriptors dialog, enter the filename or click Browse to specify a location and name for the connection file. For more information at any time, press F1 or click **Help** from within the Export Connection Descriptors dialog.
4. After you have specified a filename, select the appropriate connections from the list.

   The connection information for the selected connections is saved in the file and can be imported for use by others.
5. Click **OK**.

An alternative way of exporting connections, including database connections that are IDE Connections, is to use the Resource Palette.

26.2.7.2 Importing Database Connections

You can import connection descriptors that have previously been exported.

To import a database connection:

1. Choose **View > Database > Database Navigator**.
2. Right-click either IDE Connections or application and choose **Import Connections**.
3. In the Import Connection Descriptors dialog, enter the file name of your exported connection file or click Browse to locate it. For more information at any time, press F1 or click **Help**.
4. After you have specified a file name, select one or more connections from the list that appears.
5. Click **OK**.

An alternative way of importing connections that can include database connections is to use the Resource Palette.

26.2.8 How to Open and Close Database Connections

You can manually connect to a database connection already defined in JDeveloper, or disconnect a database connection.

To open a database connection:

1. Choose **View > Database > Database Navigator**.
2. Expand **IDE Connections** or **application**, and select a database connection.
3. Expand the node.

   Alternatively, right-click the closed connection and choose **Connect**.

To close a database connection:

1. Choose **View > Database > Database Navigator**.
2. Expand **IDE Connections** or **application**, and select a database connection.
3. Right-click the connection and choose **Disconnect**.

26.2.9 How to Delete Database Connections

Deleting connections removes them from the Database Navigator and the installation of JDeveloper.

When you delete a connection, JDeveloper does not warn you that a project may be dependent upon it, and removes the connection from all of JDeveloper, not just a workspace or project. It is best to use caution when deleting connections.

To delete a database connection:

1. Choose **View > Database > Database Navigator**.
2. Expand **IDE Connections** or **application**, and select a database connection to delete.
3. Right-click the connection and choose **Delete**.
4. In the confirmation dialog, click **Yes**.

26.2.10 How to Register a New Third-Party JDBC Driver

If you plan to use a third-party JDBC driver for the BI JDBC driver, or the WebLogic JDBC drivers for DB2, Informix, SQL Server and Sybase, you must register it with JDeveloper so that it will be available when you define the connection.

To register a new third-party JDBC driver:

1. Choose **Tools > Preferences**.
2. In the Preferences dialog, select **JDBC Driver Options**.
3. The list of third-party JDBC drivers currently registered with JDeveloper is displayed. To add a new entry to the list, click **New**.

   A new entry appears in the list and in the Driver Class field, with a default driver class name.
4. In the **Driver Class** field, alter the new entry to reflect its fully qualified class name.

   Make sure that the correct entry is still selected in the **Registered JDBC Drivers** list.
5. Select a library to associate the driver with. You can browse to an existing library, or enter the fully qualified path to the library. The classpath for the library is displayed in **Classpath**.

   Be sure to include this library in any project that uses the third-party driver.
6. Click **OK**.

   The driver will now appear in the list of available third-party JDBC drivers both in this dialog (after you return to it) and in the Create Database Connection dialog.

Alternately, if you are already in the Create Database Connection dialog, you can register a third-party JDBC driver without leaving the dialog. Choose **Generic JDBC** as the Connection Type, and click **New** to open the Register JDBC Driver dialog where you provide the class name and library for the driver.

26.2.11 How to Create User Libraries for Non-Oracle Databases

To connect to a non-Oracle database, you first have to create a library containing the JDBC drivers.

After you have created a user library, you can create a database connection.

To create a user library:

1. Choose **Tools > Manage Libraries**.
2. In the Manage Libraries dialog, select the **Libraries** tab, then select the **User** node, and click **New**.
3. In the Create Library dialog, enter a library name, select the **Class Path** node, and click **Add Entry.** In the Select Path Entry dialog, browse to the location of the drivers for the database you are connecting to. Select the driver files, and click **Select**.
4. In a similar way, in the Create Library dialog, enter a library name, select the **Source Path** node, and click **Add Entry**. In the Select Path Entry dialog, browse to the location of the drivers for the database you are connecting to. Select the driver files, and click **Select**.

5. In the Create Library dialog, click **OK**, and in the Manage Libraries dialog, click **OK**.

The library containing the JDBC drivers will be available for you to select when you create a connection to the non-Oracle database.

26.2.12 Reference: Connection Requirements for Oracle's Type 2 JDBC Drivers (OCI)

When you create connections using Oracle's JDBC/OCI drivers, be aware of the following platform-specific requirements:

- You must have the required native libraries (`.dll` files on Windows, and `.so/.sl` files on UNIX).

  With the Oracle Type 2 driver (JDBC/OCI), the version of the JDBC driver must match the version of the Oracle home. For example, the Oracle JDBC Driver version 11 requires that Oracle home contain version 11 of `ocijdbc11.dll`, as well as the Oracle Network software and Required Support Files.

  You can download drivers from the JDBC Driver Downloads page at http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html.

  If you are connecting to a local database which is a different version from the JDBC driver you are using, then you must install the Oracle client software into a separate Oracle home, and connect via the Oracle Net Listener.

- You must place the `ORACLE_HOME` directory in which the client-side file for the required native libraries resides into a directory listed in your PATH environment variable.
  - On Windows: In your PATH environment variable list the `%ORACLE_HOME%\bin` directory in which the client-side DLL file resides. If you have multiple Oracle homes installed on your machine, use the Oracle home Switch utility to choose the correct Oracle home.
  - On UNIX: List the `{ORACLE_HOME}/lib` directory in which the client-side `.so/.sl` file resides in your PATH environment variable.
- If your Oracle home for the OCI driver is not the same as the Oracle home in which JDeveloper is installed, you must set the `ORACLE_HOME` environment variable.
- If your Oracle home for the OCI driver is not the same as the Oracle home in which JDeveloper is installed and you have no other OCI drivers listed in `java.library.path`, you can edit `{$ORACLE_HOME}/jdeveloper/jdev/bin/jdev.conf` with a line similar to the following, replacing the path shown with the full path to your Oracle home:

  On Windows: `AddNativeCodePath C:/ORACLE/ORAnn/BIN`

  On UNIX: `AddNativeCodePath /u01/app/oracle/product/n.n.n/lib`

  `AddNativeCodePath` adds to `java.library.path` the directory name in which the Java VM searches for shared libraries.

  Note: Because `AddNativeCodePath` only appends the directory to the path, if you have an OCI driver path already in the `PATH` environment variable, set `ORACLE_HOME` instead of editing `PATH` with `AddNativeCodePath`.

26.3 Browsing and Searching Databases

You can control how much of the data source you view and how you view it, and search for database objects.

26.3.1 Browsing Databases

You can browse online databases and offline database objects.

26.3.1.1 Browsing Online Databases

You can browse online databases by opening JDBC connections accessible in the Database Navigator.

JDBC connections permit access to PL/SQL objects and blocks and the Java classes that implement those objects. Any database can be browsed; however only Oracle Database permits access to the full range of database objects.

Database connections are shown in the Database Navigator, under the IDE Connections node or the node for the application. Expand the connection to show the database's schemas. By default, the connection only allows the schema of the user identified in the connection to be browsed. Other schemas can be browsed as well, if the user has the required privileges. Expanding a schema shows nodes for the object types that the schema contains. Expanding the node for an object type show the individual objects it contains. When you have expanded a node as far as it can be expanded, you can double-click an object (or right-click and choose Open) to display its content. Depending on the type of the object, its structure may also be displayed in the structure pane.

26.3.1.2 Browsing Offline Database Objects

You can browse offline database objects using the Application Navigator.

26.3.1.3 How to View Online and Offline Database Objects

You can view database objects:

- To view database objects through a real time connection (online database), use the Database Navigator.
- To view offline database objects, use the Application Navigator.

  Changes to database objects in projects (i.e. visible via Application Navigator) can be reconciled against a live database, but until reconciliation, no changes to the offline objects affect online databases.

To open a navigator:

1. Choose **View** from the main toolbar.
2. To open:
   - The Application Navigator, choose **Application Navigator**.
   - The Database Navigator, choose **Database > Database Navigator**.

26.3.2 How to Browse online Database Objects

You can browse schemas and the objects they contain via a JDBC connection to an online database.

To browse live database connections:

1. Choose View > **Database** > **Database Navigator**.
2. Expand a connection to view the schemas available.
3. Expand a schema to view all the object types visible.
4. If necessary, apply a filter at the connection, schema, or database object type level.

> **Note:** By default, a filter is set on tables to exclude those in the recycle bin for an Oracle database.

26.3.3 How to Browse Offline Databases and Schemas

Browse offline databases and schemas in the Application Navigator to find objects such as offline tables or views.

To view offline schemas and the objects they contain:

1. In the Application Navigator, expand the project containing your offline schemas.
2. Expand Offline Database Sources and then expand the database and schema you want to browse.

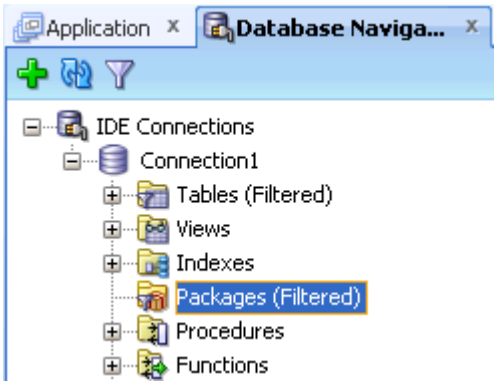
26.3.4 How to Use Database Filters

You can filter schemas, database object types, and database objects within a type, so that a subset that you define is displayed under the connection node. This is useful in environments where there may be thousands of schemas accessible from a connection.

> **Note:** When you create a connection to Oracle Database, objects for the schema named in the connection are shown. To see the contents of other schemas, expand the Other Users node and then expand the node for the schema you want.
>
> If you connect via Generic JDBC or JDBC-ODBC connections, all schemas are shown.

You can define a filter for schemas in a connection, or for any set of object types (Tables, Views, etc.) within a schema, or for any set of objects within an object type node (for example, display only the tables that begin with DB).

Figure 26–1 Filtered Objects in Database Navigator



To use filters:

1. Choose **View > Database > Database Navigator**.
2. Expand IDE Connections or application, and select a database connection.
3. Expand the connection if it has not yet been loaded. Filtering is not available until the connection has been loaded (once per connection per JDeveloper session). Select a connection, schema within a connection, or node within a schema:
4. If a filter is currently applied to the selection, a filter icon appears on the node of the selected object, and (filtered) appears next to the node name, as shown in Figure 26–1, "Filtered Objects in Database Navigator".

   With the object still selected, click the **Apply Filter button** in the Database Navigator toolbar and a dialog appears, showing the current selection, if any. From this dialog, you can change the filter currently applied.

26.3.5 How to Enable and Disable Database Filters

JDeveloper provides filters so that you can view defined sets of schemas, tables, views, or other objects.

To create filters for online database objects:

1. Choose **View > Database > Database Navigator**.
2. Expand IDE Connections or application.
3. Select the connection, schema within a connection, or node within a schema, then perform either of these actions:
   - Right-click your selection and choose Apply Filter.
   - In the Database Navigator toolbar, click the **Apply Filter button** in the Database Navigator toolbar.
4. A filter dialog appears, appropriate to the object you selected. For connections and schema, a selection box appears. For other objects, type in the text (case-sensitive) which JDeveloper matches to object names in the selected node. You can use the wildcard character `%`.
5. Click **OK**. Notice that the list of objects is now filtered to display only those names that match the criteria you selected.

26.3.6 How to Open a Database Table in the Database Object Viewer

You can open a table in a live database connection in the Database Object Viewer.

There are a number of tabs along the bottom of the Database Object Viewer that allow you to examine and change the structure of the table and the data contained in the table.

To view and edit the structure of the table in the object viewer:

1. Open the table in the Database Object Viewer by selecting it in the Database Navigator and double-clicking it. Alternatively, you can right-click the table and choose Open.
2. Select the tab that contains the information you are interested in, for example, Columns. For more information at any time, press F1 or click Help from within the Database Object Viewer.

An alternative way of viewing and editing the structure of a table is in the Edit Table dialog.

You can edit the data in a table.

26.3.7 How to Edit Table Data

You can change the data in a database table, for example to test the functionality of an application you are developing. You can change the value in a single cell, and add and delete rows. When you have finished you can choose to commit your changes to the database, or to rollback the changes and leave the database table unchanged.

To edit data in a table:

1. Display the table in the Database Object Viewer by double-clicking it in the Database Navigator.
2. Click the **Data** tab to display the contents of the table.
3. Position the cursor in the cell you want to change and type the new value.
   - To add a new record, click the **Insert Row button**.
   - To delete one or more records, select them and click the **Delete Selected Row(s) button**.
4. When you have finished, either:
   - Click the **Commit Changes button** to commit your changes to the database.
   - Click the **Rollback Changes button** to rollback your changes.

26.3.8 How to Find Objects in the Database

You can search for database objects in Oracle Database which has a connection to JDeveloper using the Find Database Object Window.

You must already have a connection to the database.

To find database objects:

1. From the main menu, choose **View > Database > Find DB Object** to open the Find Database Object Window.

   For more information at any time, press F1 or click **Help** from within the navigator.
2. Select the connection name from the Connection list.
3. Enter search terms in the Name field. You can use the wildcard `%` to return a number of matching objects. For example, enter `EM%` to return all objects with names starting with `EM`.
4. If necessary, click `More` to enter more search criteria.
5. Click Lookup. The results are returned in the Search window. To view or edit one of the objects (or the parent object that contains the specified object), double-click or right-click its name in the results display.

26.4 Connecting to Databases

This section describes how to connect to Oracle and non-Oracle databases.

26.4.1 What Happens When You Create a Connection to a Database

When you create a database connection using the Create Database Connection dialog, the new connection is created and a node representing the connection is displayed in the:

- Database Navigator.
- Application Navigator.
- Resource Palette.

26.4.2 How to Create Connections to Oracle Databases

You can connect to and work with Oracle databases. For information about the specific versions that are supported, see "JDeveloper Certification Information" at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html.

26.4.2.1 How to Create a Connection to Oracle Database

JDeveloper allows you to connect to:

- Oracle Database 11g Release 2
- Oracle Database 11g Release 1
- Oracle Database 11g Release 2 XE.
- Oracle Database 10g XE.
- Oracle Database 10g Release 2
- Oracle Database 10g Release 1
- Oracle Database 9i Release 2

You can also connect to (although the connection does not use Oracle (JDBC):

- MySQL 4.1 or 5.0
- Oracle TimesTen In-Memory Database
- Oracle Database Lite 10g Release 1 and Release 3

To create a database connection to Oracle Database:

1. Use a connection type of Oracle (JDBC).
2. Enter appropriate username, role, and password values for the database connection.
3. By default the Save Password field is checked so that you will not be prompted to enter it again.
4. Select the thin driver.
5. If the database is on the local machine, use the default of localhost. Otherwise enter an IP address or a host name that can be resolved by TCP/IP, for example, `myserver`.
6. Enter either the SID or service name for the database.
7. Test the connection by clicking Test Connection. You may have to briefly wait while JDeveloper connects to the database.

If the test succeeds, a success message appears in the status text area. If the test does not succeed, an error appears. In this case, change any previously entered information as needed to correct the error, or check the error content to determine other possible sources of the error.

26.4.2.2 How to Create a Connection to MySQL

JDeveloper allows you to connect to MySQL 4.1 or, 5.0, or to emulate MySQL 4.1 or, 5.0 for offline database operations. For more information about MySQL, see http://www.oracle.com/us/products/mysql/index.htm.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with MySQL. You can:

- Create tables:
  - Add column(s) specifying data types, NOT NULL constraints, default values and column comments
  - Add primary key and foreign key constraints
- Alter tables:
  - Add column(s)
  - Drop column(s)
  - Add index
  - Drop index
  - Add constraint (primary key, unique key, and foreign key)
  - Drop constraint (primary key, unique key, and foreign key)
- Rename table
- Drop table

To create a database connection to MySQL:

1. From http://mysql.com/downloads, download and install `MySQL Connector/J 3.1`.
2. Set up the user library to contain the following `mysql-connector-java-3.1.8-bin.jar`.
3. Create a database connection to MySQL.
4. Use the following values:
   - Connection Type: `MySQL`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `com.mysql.jdbc.Driver`
   - Library: the library you created for the driver.
   - JDBC URL: `jdbc:mysql://machine-name/database-name`

26.4.2.3 How to Create a Connection to Oracle TimesTen In-Memory Database

Oracle TimesTen In-Memory Database is a memory-optimized relational database that provides applications with extremely fast response time and very high throughput as required by many application in a wide range of industries. Deployed in the

application tier, TimesTen databases reside entirely in physical memory with persistence to disk storage for recoverability.

JDeveloper allows you to connect to Oracle TimesTen In-Memory Database 6.0, 7.0, or 11g, or to emulate TimesTen databases for offline database operations. For more information about Oracle TimesTen In-Memory Database 11g, see http://www.oracle.com/technetwork/database/timesten/overview/index.html.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with TimesTen databases. You can:

- Create tables.
  - Add columns
  - Add primary keys and foreign keys
- Alter tables.
  - Add columns
  - Drop columns
  - Add primary keys and foreign keys
  - Drop primary keys and foreign keys
- For Oracle TimesTen In-Memory Database v6.0, a current limitation is that in order to see constraints such as primary keys, you must ensure that your connection username is the same as the name of the schema you are connecting to.

To create a database connection to Oracle TimesTen In-Memory Database:

1. Create a database connection to the TimesTen database.
2. Use the following values:
   - Connection Type: `Generic JDBC`
   - Username and Password: leave blank
   - Driver Class: `com.timesten.jdbc.TimesTenDriver`
   - Library:
     - Release 6.0.1: `timesten-install\tt60\lib\classes14.jar`
     - Release 7.0.5: `timesten-install\tt70_32\lib\ttjdbc5.jar`
     - Release 11.2.1: `timesten-install\tt1121_32\lib\ttjdbc5.jar`
   - JDBC URL:
     - Release 6.0.1: `jdbc:timesten:client:RunDataCS60`
     - Release 7.0.5: `jdbc:timesten:client:RunDataCS_tt70_32`
     - Release 11.2.1: `jdbc:timesten:client:cachealone1_CS`

26.4.2.4 How to Create a Connection to Oracle Database Lite

Oracle Database Lite allows an image of an Oracle database to exist on a remote device. Users can update the data on Oracle Database Lite and commit it to the main database at given intervals. For more information about Oracle Database Lite 10*g*, see http://www.oracle.com/technetwork/database/database-lite/overview/index.html.

JDeveloper supports connections to and database emulation of Oracle Database Lite 10g Release 1 and Release 3. For information, see "JDeveloper Certification Information" at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html.

This driver requires installation of Oracle Database Lite 4.0 JAR or higher. Projects using the driver must include `ORACLE_HOME/lite/classes/olite.jar` in a library.

To create a database connection to Oracle Database Lite:

1. Download a Java JDBC driver for Oracle Database Lite. Download and install Oracle Database Lite 4.0 JAR or higher, and create a library to include `oracle-database-lite/lite/classes/olite.jar`.
2. If you are using a type 2 driver, you must edit the `ide.conf` file to provide one new value:
   1. Close JDeveloper.
   2. In a text editor, open `ide.conf` in the `jdev-install/ide/bin` directory.
   3. Add the new entry

      ```
      AddJavaLibPath oracle-database-lite\olite40.jar
      ```
3. Create a database connection to the Oracle Database Lite.

   Use the following values:

   - Connection Type: `Oracle Lite`
   - Username and Password: enter the appropriate values for the connection.
   - Driver: `Type 2`
   - Datasource name: enter an appropriate value for the connection.

26.4.3 How to Create Connections to Non-Oracle Databases

You can connect to and work with non-Oracle databases. For information about the specific versions that are supported, see "JDeveloper Certification Information" at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html.

In general, you can:

- Import database objects to JDeveloper.
- Create offline database objects.
- Edit offline database objects.

Creating a database connection:

1. Create a library containing the JDBC drivers.
2. Create a database connection.
3. In the Create Database Connection dialog, enter the appropriate values for the database. For more information, refer to the help topic for the database you are connecting to.
4. Finally, you must configure your projects to use the correct data types.

In the descriptions below for specific types of connection the JDBC URL is shown, however if you prefer you can enter details of the server, port, and database in the fields of the Create Database Connection dialog.

26.4.3.1 How to Create a Connection to Apache Derby

Apache Derby is an open source relational database implemented entirely in Java. JDeveloper allows you to connect to Apache Derby 10.5, or to emulate Apache Derby 10.5 for offline database operations. For more information about Apache Derby, see `http://db.apache.org`.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with Apache Derby. You can:

- Create tables:
  - Add column(s)
  - Add primary key and foreign key constraints
- Alter tables:
  - Add column(s)
  - Drop column(s)
  - Add constraint
  - Drop constraint

> **Note:** Column default values are not supported

You can connect to Apache Derby using Derby's embedded JDBC driver or you can create a connection as a client.

To connect to Apache Derby using the embedded driver:

1. Create a database connection to the Apache Derby database.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `org.apache.derby.jdbc.EmbeddedDriver`
   - Library: `lib/derbyclient.jar`
   - JDBC URL: `jdbc:derby://machine-name:port/databases/database-name`

To connect to Apache Derby as a client:

1. Create a database connection to the Apache Derby database.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `org.apache.derby.jdbc.ClientDriver`
   - Library: `lib/derbyclient.jar`

- JDBC URL:
  `jdbc:derby://machine-name:port/databases/database-name`

26.4.3.2 How to Create a Connection to IBM DB2 Universal Database

JDeveloper allows you to connect to IBM DB2 Universal Database 9.5 or 8.1, or to emulate IBM DB2 Universal Database 9.5 or 8.1 for offline database operations. For more information about IBM DB2 Universal Database, see `http://www.ibm.com`.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with IBM DB2, and working with IBM DB2 databases is subject to the following limitations:

- Create tables, and add columns specifying Datatypes, NOT NULL constraints and default values, add primary and foreign keys, and create indexes.
- Alter tables, and add and drop columns, add and drop indexes, add and drop constraints (primary keys, unique keys, check and foreign keys).
- Rename tables.
- Drop tables.

Notes: IBM DB2 Universal Database 9.5 syntax of DROP column and ALTER COLUMN is supported for IBM DB2 Universal Database 9.5.

You can only connect to DB2 Universal Database 8.1 with Fix Patch 3 or higher, or to DB2 Universal Database 9.5. When you have a DB2 connection, column and constraint information is not displayed in the Database Navigator. Instead columns and constraints are displayed in the Structure window when the table is selected in the Database Navigator.

You can connect to IBM DB2 using the WebLogic JDBC driver or using IBM's native driver.

To connect to IBM DB2 using the WebLogic JDBC driver:

1. Create a database connection to the IBM DB2 database.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `weblogic.jdbc.db2.DB2Driver`
   - JDBC URL:
     `jdbc:weblogic:db2://machine-name:port;DatabaseName=database-name`

To connect to IBM DB2 using the native driver:

1. Download the Type 4 JDBC driver for IBM DB2.
2. Set up the user library to contain the following files.
   - DB2 UDB 8.1
     - db2jcc.jar
     - db2jcc_javax.jar

  - db2jcc_license_cu.jar
- DB2 UDB 9.5
  - db2jcc.jar
  - db2jcc4.jar

3. Create a database connection to IBM DB2.

   Use the following values:

   - Connection Type: `DB2 UDB`
   - Username and Password: enter appropriate values for the database connection.
   - Driver Class: `com.ibm.db2.jcc.DB2Driver`
   - Library: the library you created for the driver.
   - JDBC URL: `jdbc:db2://machine-name:50000/database-name`

26.4.3.3 How to Create a Connection to IBM Informix Dynamic Server

JDeveloper allows you to connect to IBM Informix DS 10 or 11.5, or to emulate IBM Informix DS 10 or 11.5 for offline database operations. For more information about IBM Informix DS, see www.IBM.com.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with IBM Informix DS. You can:

- Create tables, and add columns.
- Add primary key and foreign key constraints.
- Alter tables, add columns, and drop columns.

You can connect to IBM Informix DS using the WebLogic JDBC driver or using IBM's native driver.

To connect to IBM Informix DS using the WebLogic JDBC driver:

1. Create a database connection to the IBM Informix DS database.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `weblogic.jdbc.informix.InformixDriver`
   - JDBC URL:
     `jdbc:weblogic:informix://machine-name:port;informixServer=server-name;databaseName=database-name`

To connect to IBM Informix DS using native drivers:

1. From www.IBM.com, download and install the appropriate Informix JDBC Driver:
   - For IBM Informix DS 10, choose v2.21.JC5 or later.
   - For IBM Informix DS 11.5, choose v3.00.JC3 or later.
2. Set up the user library to contain `install-directory\lib\ifxjdbc.jar`.
3. Create a database connection to IBM Informix DS.

Use the following values:

- Connection Type: `Generic JDBC`
- Username and Password: enter the appropriate values for the connection.
- Driver Class: `com.informix.jdbc.IfxDriver`
- Library: the library you created for the driver.
- JDBC URL:
  `jdbc:informix-sqli://machine-name:port/database-name:INFORMIXSERVER=machine-name`

26.4.3.4 How to Create a Connection to Microsoft SQL Server

JDeveloper allows you to connect to Microsoft SQL Server 2005, or 2008, or to emulate Microsoft SQL Server 2005, or 2008 for offline database operations. For more information about Microsoft SQL Server, see `http://www.microsoft.com`.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with Microsoft SQL Server. You can:

- Create tables:
  - Add column(s) specifying data types, NOT NULL constraints, default values and column comments
  - Add primary key and foreign key constraints
  - Create indexes
- Alter tables:
  - Add column(s)
  - Drop column(s)
  - Add indexes
  - Drop indexes
  - Add constraint (primary key, unique key, and foreign key)
  - Drop constraint (primary key, unique key, and foreign key)
- Drop tables

You can connect to Microsoft SQL Server using the WebLogic JDBC driver or using Microsoft's native driver.

To connect to Microsoft SQL Server using the WebLogic JDBC driver:

1. Create a database connection to the Microsoft SQL Server database.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `weblogic.jdbc.sqlserver.SQLServerDriver`
   - JDBC URL:
     `jdbc:weblogic:sqlserver://machine-name\MSSQLSERVER:port;databaseName=database-name`

To connect to Microsoft SQL Server:

1. From www.microsoft.com, download and install the appropriate Microsoft SQL Server driver:
   - For Microsoft SQL Server 2005, choose Microsoft SQL Server 2005 Driver.
   - For Microsoft SQL Server 2008, choose Microsoft SQL Server 2008 Driver.
2. Set up the user library to contain install-directory\sqljdbc.jar.
3. Create a database connection to Microsoft SQL Server. Use the following values:
   - Connection Type: `SQLServer`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class: `com.microsoft.sqlserver.jdbc.SQLServerDriver`
   - Library: the library you created for the driver.
   - JDBC URLs: `jdbc:sqlserver://machine-name:port;DatabaseName=database-name`, where the section `DatabaseName=database-name` is optional

What you May Need to Know

If you are using Windows Authentication credentials to connect to Microsoft SQL Server, you need to add do the following:

- Add the connection property `integratedSecurity=TRUE` and the username and password values to the JDBC URL, for example

  ```
  jdbc:sqlserver://machine-name:port;DatabaseName=database-name;username=USERNAME
  ;password=PASSWORD;integratedSecurity=TRUE
  ```

- Add the location of `sqljdbc_auth.dll` to your PATH variable:
  - For 32bit JVM, this is `installation-directory\sqljdbc_version\language\auth\x86`
  - For 64bit JVM, this is `installation-directory\sqljdbc_version\language\auth\x64`

For more information, see Building the Connection URL, which is available as part of Connecting to SQL Server with the JDBC Driver at the Microsoft MSDN website.

26.4.3.5 How to Create a Connection to SQLite

SQLite is a relational database management system represented by a platform-independent file that resides on a host computer, for example, smartphone platforms. JDeveloper allows you to connect to a SQLite 3.6 database file, or to emulate SQLite 3.6 for offline database operations. For more information about SQLite, see http://www.sqlite.org.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with SQLite. You can:

- Create tables, and add columns.
- Alter tables, and add columns.
- Copy To Project, where you copy tables and their columns and primary keys from a connection to a SQLite database to an offline database which emulates SQLite.
- Constraints, indexes and the column properties can be modeled in an offline database, but DDL is only generated for tables and columns; there is no support

for generating constraints (including primary keys) on tables, or generating any other object type (for example, indexes, views, triggers). This means that for tables in an online SQLite database, the Create/Edit Table dialog only shows the columns panel.

To create a database connection to SQLite:

- Download a Java JDBC driver for SQLite and create a library for it.
- Create a database connection to SQLite.
- Use the following values:
  - Connection Type: `Generic JDBC`
  - Username and Password: leave blank
  - Driver Class: `org.sqlite.JDBC`
  - Library: the library you created for the driver.
  - JDBC URL: `jdbc:sqlite://path/database-name`, where path is the path of the database file and database-name is the name of the SQLite database at the specified location. If the database does not exist at specified location, it will be created when the connection is made.

26.4.3.6 How to Create a Connection to Sybase ASE

JDeveloper allows you to connect to Sybase Adaptive Server Enterprise 12.5 or 15, or to emulate Sybase ASE 12.5 or 15 for offline database operations. For more information about Sybase Adaptive Server Enterprise, see www.sybase.com.

The Create Table or Edit Table dialog is generic, and some features may not be available when working with Sybase ASE. You can:

- Create tables:
  - Add column(s)
  - Add primary key and foreign key constraints

  Add column(s)
- Alter tables:
  - Add column(s)
  - Drop column(s)
  - Add constraint
  - Drop constraint

> **Note:** Column default values are not supported

You can connect to Sybase ASE using the WebLogic JDBC driver or using Sybase's native driver.

To connect to Sybase ASE using the WebLogic JDBC driver:

1. Create a database connection to the Sybase ASE database.

   Use the following values:

   - Connection Type: `Generic JDBC`

- Username and Password: enter the appropriate values for the connection.
- Driver Class: `weblogic.jdbc.sybase.SybaseDriver`
- JDBC URL: `jdbc:weblogic:sybase://machine-name:port;DatabaseName=databas-name`

To connect to Sybase ASE using the native driver:

1. From www.sybase.com, download and install the appropriate Sybase JDBC driver:
   - For Sybase ASE 12.5, choose jConnect Version:5.5 or later.
   - For Sybase ASE 15, choose jConnect Version: 6.0.5 or later.
2. Set up the user library to contain the following:

   ```
   install-directory\jConnect-5_5\classes\jconn2.jar install_
   directory\\jConnect-5_5\classes\jTDS2.jar
   ```

3. Create a database connection to Sybase ASE.

   Use the following values:

   - Connection Type: `Generic JDBC`
   - Username and Password: enter the appropriate values for the connection.
   - Driver Class:
     - For Sybase ASE 12.5, use `com.sybase.jdbc2.jdbc.SybDriver`
     - For Sybase ASE 15 use `com.sybase.jdbc3.jdbc.SybDriver`
   - Library: the library you created for the driver.
   - JDBC URL: `jdbc:sybase:Tds:machine:port/database-name`

26.5 Importing and Exporting Data

You can import data into tables in a database through a database connection.

> **Note:** You cannot import date into offline tables as offline tables are just representations of database tables.

You can import data from:

- csv, a file containing comma-separated values including a header row for column identifiers.
- xls, a file in Microsoft Excel format (only for import into existing and new tables).

You can import the data into:

- A existing table in the database.
- A new table that you create as part of the import process.
- Using a SQL*Loader control file.
- An external table.

26.5.1 Importing Data Using SQL*Loader

When you choose the SQL*Loader option in the Data Import Wizard, JDeveloper creates the following files in the same location as the import file containing the data: table.ctl, which contains information about the file containing the data and the table into which it can be imported. table.bat and table.sh, to run the import.

26.5.2 Importing Data Into an External Table

You can import data into an external table, which is a flat file in which you can query data as though it were an Oracle table.

When you choose the External Table option, JDeveloper creates the SQL and displays it in the SQL Worksheet where you can examine it and make any necessary changes before running the script.

26.5.3 How to Import Data into Existing Tables

You can import data into a table in a database through a database connection.

The following import file formats are supported:

- csv, a file containing comma-separated values including a header row for column identifiers.
- xls, a file in Microsoft Excel format.

To import data to an existing database table:

1. From the main menu, choose **View > Database > Database Navigator** to open the Database Navigator.
2. If necessary, create a connection to the database.
3. Expand the node for the database connection, the schema, Tables, and select the table node you want to import data to.
4. Right-click and choose **Import Data** and in the Open dialog enter or browse to the location of the file.

   Click **OK** to launch the Data Import Wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.
5. On the Column Definition page of the Data Import wizard, enter the name of the new table.

26.5.4 How to Import Data to New Tables

You can import data into a database table that you create as part of the import process.

To import data to a new database table:

1. From the main menu, choose **View > Database > Database Navigator** to open the Database Navigator.
2. If necessary, create a connection to the database.
3. Expand the node for the database connection, the schema, Tables, and select the table node you want to import data to.
4. Right-click and choose **Import Data** and in the Open dialog enter or browse to the location of the file. Click **OK** to launch the Data Import Wizard.

For more information at any time, press F1 or click **Help** from within the wizard.

5. On the Column Definition page of the Data Import wizard, enter the name of the new table.

26.5.5 How to Import Data Using SQL*Loader

You can create a SQL*Loader control file which can be used to import data.

To import data to a SQL*Loader control file:

1. From the main menu, choose **View > Database > Database Navigator** to open the Database Navigator.
2. If necessary, create a connection to the database.
3. Expand the node for the database connection, the schema, Tables, and select the table node you want to import data to.
4. Right-click and choose **Import Data** and in the Open dialog enter or browse to the location of the file.

   Click **OK** to launch the Data Import Wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.
5. On the Data Preview page of the Data Import wizard, choose **SQL*Loader Table**.
6. On the Options page of the Data Import wizard, choose the options for the generated file.
7. When you complete the Data Import wizard, the SQL*Loader control file called `table.ctl` is created in the same location as the data file, along with a table.bat and table.sh files which allow you to run it. If you selected **Send to worksheet** on the Finish page of the Data Import wizard, the SQL defining the table is displayed in the SQL Worksheet.

26.5.6 How to Import Data Using External Tables

You can import data into tables in a database through a database connection.

To import data to an external table:

1. If necessary, create a connection to the database.
2. Expand the node for the database connection, the schema, Tables, and select the table node you want to import data to.
3. Expand the node for the database connection, the schema, Tables, and select the table node you want to import data to.
4. Right-click and choose **Import Data** and in the Open dialog enter or browse to the location of the file.
5. Click **OK** to launch the Data Import Wizard.

   For more information at any time, press F1 or click Help **from** within the wizard.
6. On the Data Preview page of the Data Import wizard, choose External Table.
7. On the Options page of the Data Import wizard, choose the options for the generated file. When you complete the Data Import wizard, the SQL is displayed in the SQL Worksheet, where you can examine it and make any changes. When you are satisfied, right-click in the Worksheet, and choose **Run** in SQL*Plus.

26.5.7 Exporting Data from Databases

You can export data from tables in a database through a database connection.

The data can be saved to a file or to the clipboard. The following formats are supported:

- `csv`, to create a file containing comma-separated values including a header row for column identifiers.

  Note: You can choose a different delimiter.

- `fixed`, to create a file where records are the same byte length.
- `html`, to create an HTML file containing a table with the data. insert, to create a file containing SQL INSERT statements.
- `loader`, to create a SQL*Loader control file.
- `text`, to create a text file.
- `ttbulkcp`, to create a data files to be used with the TimesTen ttbulkcp command line utility. For more information, see Oracle TimesTen In-Memory Database 11g at `http://www.oracle.com/technology/products/timesten/index.html` .
- `xls`, to create a Microsoft Excel `.xls` file. The file will contain two worksheets, Export Worksheet, which contains the data, and SQL which contains the SQL statement used to export the data.
- `xml`, to create a file containing XML tags and data.

In the Export Data dialog, you can limit the data to be exported by selecting only some columns, and by entering a `WHERE` clause.

Note: If you encounter problems exporting large tables to Microsoft Excel files, try adding the following line to the `jdeveloper.conf` file to increase heap size, and then restarting JDeveloper:

```
AddVMOption -Xmx1024M
```

If the number of table rows exceeds 65,536, JDeveloper writes the rows to multiple worksheets within the `.xls` file.

You can also export data from a database using the Export Database wizard.

26.5.8 How to Export Data to Files

You can export data from tables in a database through a database connection.

To export data from a database table:

1. From the main menu, choose **View > Database > Database Navigator** to open the Database Navigator.
2. If necessary, create a connection to the database.
3. Expand the node for the database connection, the schema, Tables, and select the table node you want to export data from.

4. Right-click and choose Export Data to open the Export Data dialog.

   For more information at any time, press F1 or click Help from within the wizard.

26.6 Copying, Comparing, and Exporting Databases

You can copy database objects from a source schema to a destination schema. You can export database objects and data to a DDL file.

26.6.1 How to Copy Databases

You can copy database objects from a source schema to a destination schema, subject to any restrictions depending on the type of operation, which determines the behavior if objects of the same name exist in the destination schema.

You must have the source and the destination database connections already defined.

To copy a database:

1. From the main menu, choose **Tools > Database > Database Copy** to open the New Copy wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.

26.6.2 How to Compare Database Schemas

You can find differences between objects of the same type and name (for example, tables named CUSTOMERS) in two different schemas, and optionally update the objects in one schema (destination) to reflect differences in the other schema (source).

Using the Diff wizard requires the licensing of the Oracle Change Management option for Oracle Database. To purchase a license, contact your Oracle sales representative or authorized Oracle Reseller, or go to the Oracle Store to buy online at http://shop.oracle.com

You must have the source and the destination database connections already defined.

To compare database schemas:

1. From the main menu, choose **Tools > Database > Database Diff** to open the Diff wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.

26.6.3 How to Export Databases

You can export some or all objects of one or more types of database objects to a file containing SQL data definition language (DDL) statements to create these objects. Export Database wizard allows you to: Specify details of the DDL file that is generated. Select the database object objects to be exported. Choose to export data, and apply filters to specify the data to be included in the generated file.

You must have already defined a connection to the database you want to export.

To export a database:

1. From the main menu, choose **Tools > Database > Database Export** to open the Export Database wizard.

   For more information at any time, press F1 or click Help from within the wizard.

26.7 Working with Oracle and Non-Oracle Databases

This section describes how to work with Oracle Database, as well as with non-Oracle databases. There are limitations on what you can do with JDeveloper with different databases. For more information, see the relevant information in Section 26.4.2, "How to Create Connections to Oracle Databases" and Section 26.4.3, "How to Create Connections to Non-Oracle Databases."

26.8 Working with Database Reports

JDeveloper provides many reports about a database and its objects. You can also create your own user-defined database reports.

You can also run reports on offline database objects.

26.8.1 Using Database Reports

JDeveloper provides many reports about a database and its objects. You can also create your own user-defined database reports.

For some reports, you are prompted for bind variables before the report is generated. These bind variables enable you to further restrict the output. The default value for all bind variables is null, which implies no further restrictions.

The Database Reports Navigator allows you to run reports which query the database for the latest information. The time required to display specific reports will vary, and may be affected by the number and complexity of objects involved, and by the speed of the network connection to the database.

There are a number of predefined reports about the database and its objects.

You can also create your own user-defined reports.

You can examine the underlying SQL for a report, for example, to help you create your own report.

Database reports are organized in folders, and reports and folders can be exported.

You can share reports by exporting them.

The person who wants to share the report then adds it to their instance of JDeveloper using the Preferences dialog. Reports that have been exported can be imported into folders under the User Defined Reports node.

26.8.1.1 How to Run Database Reports

The Database Reports Navigator allows you to run reports which query the database for the latest information. The time required to display specific reports will vary, and may be affected by the number and complexity of objects involved, and by the speed of the network connection to the database.

Running a database report:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Locate the report you want to run, right-click and choose Open, which will overwrite any previous results in the Reports Viewer window, or Open New to open a new instance of the Reports Viewer.
3. If the Bind Variables dialog is displayed, enter the bind variables you want to use. For more information at any time, click F1 or **Help** in the Bind Variables dialog.

The report results are displayed in the Reports Viewer.

26.8.1.2 How to View the SQL for a Report

You can view the underlying SQL for a database report in the SQL Worksheet.

To view the SQL for a database report:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Run the report.
3. In the Reports Viewer, click the **Run Report in SQL Worksheet button**. The SQL Worksheet opens displaying the SQL code for the report.

26.8.1.3 How to Create User-Defined Database Reports

You can define your own reports for database features and objects.

To create user-defined reports:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Right-click the User Defined Reports node, or a folder that you have created under this node, and choose **Add Report**.
3. In the Create Report dialog, enter a name and the SQL for the report. For more information at any time, click F1 or **Help** in the Create Report dialog.

26.8.1.4 How to Edit User-Defined Database Reports

You can edit user-defined reports.

To edit a user-defined report:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Open the User Defined Reports node, and right-click on the report you want to edit, and choose Edit.
3. In the Create Report dialog, enter a name and the SQL for the report. For more information at any time, click F1 or **Help** in the Create Report dialog.

26.8.1.5 How to Create Reports Folders

You can organize user-defined reports in folders.

To create a folder:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Right-click the User Defined Reports node, and choose **Add Folder**.
3. In the Create Folder dialog, enter a name for the folder. For more information at any time, click F1 or **Help** in the dialog

26.8.1.6 How to Export User-Defined Reports

You can export database reports or folders of database reports.

If you are sharing a report, you export it, and users who want to share the report, then make it available in their instance of JDeveloper.

To export a database report or folder:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Right-click the report or folder you want to share, and choose **Export**.
3. Enter a location for the report in the Save dialog. The default name for the report is `explain.xml`.

26.8.1.7 How to Import User-Defined Reports

After you have exported database reports and folders, you can import them to a user-defined folder.

You need to first create the folder to hold the report.

This can also be a simple way to share database reports.

To import a database report or folder:

1. If it is not open, open the Database Reports Navigator. In the main menu, choose **View > Database > Database Reports**.
2. Under the User Defined Reports node, right-click the folder you want to add the report to, and choose **Import**.
3. In the Open dialog, enter or browse to the location for the exported report in the Save dialog. The default name for the report is `explain.xml`.

How to Share Database Reports

You can share database reports. The report is exported, then you add it to your invocation of JDeveloper.

Before a report can be shared:

- The report must be run.
- The report must then be exported.

Sharing a database report:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select **Database > User-Defined Extensions**. For more information at any time, press F1 or click **Help** from within the Preferences dialog.
3. Click Add Row, and under Type select REPORT, and under Location enter or browse to the location of the exported report.
4. Restart JDeveloper.
5. Choose **View > Database > Database Reports** to open the Database Reports Navigator. The shared report is listed under the Shared Reports node in the navigator.

26.8.2 Reference: Pre-Defined Database Reports

This section describes the pre-defined reports available under the Data Dictionary Reports node in the Database Reports navigator.

The reports are grouped into categories, with one or more different reports available in that category.

- About Your Database Reports

  These reports list release information about the database associated with the connection. The reports include Version Banner (database settings) and National Language Support Parameters (NLS_xxx parameter values for globalization support).

- All Objects Reports

  These reports list information about all objects accessible to the user associated with the specified database connection, not just objects owned by the user.

  - All Objects: For each object, lists the owner, name, type (table, view, index, and so on), status (valid or invalid), the date it was created, and the date when the last data definition language (DDL) operation was performed on it. The Last DDL date can help you to find if any changes to the object definitions have been made on or after a specific time.
  - Collection Types: Lists information about each collection type. The information includes the type owner, element type name and owner, and type-dependent specific information.
  - Dependencies: For each object with references to it, lists information about references to (uses of) that object.
  - Invalid Objects: Lists all objects that have a status of invalid.
  - Object Count by Type: For each type of object associated with a specific owner, lists the number of objects. This report might help you to identify users that have created an especially large number of objects, particularly objects of a specific type.
  - Public Database Links: Lists all public database links.
  - Public Synonyms: Lists all public synonyms.

- Application Express Reports

  These reports list information about Oracle Application Express 3.0.1 (or later) applications, pages, schemas, UI defaults, and workspaces. If you select a connection for a schema that owns any Oracle Application Express 3.0.1 (or later) applications, the Application Express reports list information about applications, pages, schemas, UI defaults, and workspaces. For more information, see *Oracle Application Express Administration Guide*.

- ASH and AWR Reports

  These reports list information provided by the Active Session History (ASH) and Automated Workload Repository (AWR) features.

- Database Administration Reports

  These reports list usage information about system resources. This information can help you to manage storage, user accounts, and sessions efficiently. (The user for the database connection must have the DBA role to see most Database Administration reports.)

  - All Tables: Contains the reports that are also grouped under Table reports, including Quality Assurance reports.
  - Cursors: Provide information about cursors, including cursors by session (including open cursors and cursor details.

- Database Parameters: Provide information about all database parameters or only those parameters that are not set to their default values.
- Locks: Provide information about locks, including the user associated with each.
- Sessions: Provide information about sessions, selected and ordered by various criteria.
- Storage: Provide usage and allocation information for tablespaces and data files.
- Top SQL: Provide information about SQL statements, selected and ordered by various criteria. This information might help you to identify SQL statements that are being executed more often than expected or that are taking more time than expected.
- Users: Provide information about database users, selected and ordered by various criteria. For example, you can find out which users were created most recently, which user accounts have expired, and which users use object types and how many objects each owns.

- Data Dictionary Reports

  These reports list information about the data dictionary views that are accessible in the database. Examples of data dictionary views are ALL_OBJECTS and USER_TABLES.

  - Dictionary View Columns: For each Oracle data dictionary view, lists information about the columns in the view.
  - Dictionary Views: Lists each Oracle data dictionary view and (in most cases) a comment describing its contents or purpose.

- Jobs Reports

  These reports list information about jobs running on the database.

  - All Jobs: Lists information about all jobs running on the database. The information includes the start time of its last run, current run, and next scheduled run.
  - DBA Jobs: Lists information about each job for which a DBA user is associated with the database connection. The information includes the start time of its last run, current run, and next scheduled run.
  - Your Jobs: Lists information about each job for which the user associated with the database connection is the log user, privilege user, or schema user. The information includes the start time of its last run, current run, and next scheduled run.

- PLSQL Reports

  These reports list information about your PL/SQL objects and allow you to search the source of those objects.

  - Program Unit Arguments: For each argument (parameter) in a program unit, lists the program unit name, the argument position (1, 2, 3, and so on), the argument name, and whether the argument is input-only (In), output-only (Out), or both input and output (In/Out).
  - Search Source Code: For each PL/SQL object, lists the source code for each line, and allows the source to be searched for occurrences of the specified variable.

    - Unit Line Counts: For each PL/SQL object, lists the number of source code lines. This information can help you to identify complex objects (for example, to identify code that may need to be simplified or divided into several objects).

- Security Reports

  These reports list information about users that have been granted privileges, and in some cases about the users that granted the privileges. This information can help you (or the database administrator if you are not a DBA) to understand possible security issues and vulnerabilities, and to decide on the appropriate action to take (for example, revoking certain privileges from users that do not need those privileges).

  - Auditing: Lists information about audit policies.
  - Encryption: Lists information about encrypted columns.
  - Grants and Privileges: Includes the following reports:
    - Column Privileges: For each privilege granted on a specific column in a specific table, lists the user that granted the privilege, the user to which the privilege was granted, the table, the privilege, and whether the user to which the privilege was granted can grant that privilege to other users.
    - Object Grants: For each privilege granted on a specific table, lists the user that granted the privilege, the user to which the privilege was granted, the table, the privilege, and whether the user to which the privilege was granted can grant that privilege to other users.
    - Role Privileges: For each granted role, lists the user to which the role was granted, the role, whether the role was granted with the ADMIN option, and whether the role is designated as a default role for the user.
    - System Privileges: For each privilege granted to the user associated with the database connection, lists the privilege and whether it was granted with the ADMIN option.
  - Policies: Lists information about policies.
  - Public Grants: Lists information about privileges granted to the PUBLIC role.

- Streams Reports

  These reports list information about stream rules.

  - All Stream Rules: Lists information about all stream rules. The information includes stream type and name, rule set owner and name, rule owner and name, rule set type, streams rule type, and subsetting operation.
  - Your Stream Rules: Lists information about each stream rule for which the user associated with the database connection is the rule owner or rule set owner. The information includes stream type and name, rule set owner and name, rule owner and name, rule set type, streams rule type, and subsetting operation.

- Table Reports

  These reports list information about tables owned by the user associated with the specified connection. This information is not specifically designed to identify problem areas; however, depending on your resources and requirements, some of the information might indicate things that you should monitor or address.

  For table reports, the owner is the user associated with the database connection.

- Columns: For each table, lists each column, its data type, and whether it can contain a null value. Also includes:
- Data type Occurrences: For each table owner, lists each data type and how many times it is used.
- Comments for tables and columns: For each table and for each column in each table, lists the descriptive comments (if any) associated with it. Also includes a report of tables without comments. If database developers use the COMMENT statement when creating or modifying tables, this report can provide useful information about the purposes of tables and columns
- Constraints: Includes the following reports related to constraints:
- All Constraints: For each table, lists each associated constraint, including its type (unique constraint, check constraint, primary key, foreign key) and status (enabled or disabled).
- Check Constraints: For each check constraint, lists information that includes the owner, the table name, the constraint name, the constraint status (enabled or disabled), and the constraint specification.
- Enabled Constraints and Disabled Constraints: For each constraint with a status of enabled or disabled, lists the table name, constraint name, constraint type (unique constraint, check constraint, primary key, foreign key), and status. A disabled constraint is not enforced when rows are added or modified; to have a disabled constraint enforced, you must edit the table and set the status of the constraint to Enabled (see the appropriate tabs for the Create/Edit Table (with advanced options) dialog box).
- Foreign Key Constraints: For each foreign key constraint, lists information that includes the owner, the table name, the constraint name, the column that the constraint is against, the table that the constraint references, and the constraint in the table that is referenced.
- Primary Key Constraints: For primary key constraint, lists information that includes the owner, the table name, the constraint name, the constraint status (enabled or disabled), and the column name.
- Unique Constraints: For each unique constraint, lists information that includes the owner, the table name, the constraint name, the constraint status (enabled or disabled), and the column name.
- Indexes: Includes information about all indexes, indexes by status, indexes by type, and unused indexes.
- Organization: Specialized reports list information about partitioned tables, clustered tables, and index-organized tables.
- Quality Assurance: (See Quality Assurance reports.)
- Statistics: For each table, lists statistical information, including when it was last analyzed, the total number of rows, the average row length, and the table type. In addition, specialized reports order the results by most rows and largest average row length.
- Storage: Lists information about the table count by tablespace and the tables in each tablespace.
- Triggers: Lists information about all triggers, disabled triggers, and enabled triggers.

- User Synonyms: Displays information about either all user synonyms or those user synonyms containing the string that you specify in the Enter Bind Variables dialog box (deselect Null in that box to enter a string).
- User Tables: Displays information about either all tables or those tables containing the string that you specify in the Enter Bind Variables dialog box (deselect Null in that box to enter a string).
- Quality Assurance reports: These are table reports that identify conditions that are not technically errors, but that usually indicate flaws in the database design. These flaws can result in various problems, such as logic errors and the need for additional application coding to work around the errors, as well as poor performance with queries at run time.
- Tables without Primary Keys: Lists tables that do not have a primary key defined. A primary key is a column (or set of columns) that uniquely identifies each row in the table. Although tables are not required to have a primary key, it is strongly recommended that you create or designate a primary key for each table. Primary key columns are indexed, which enhances performance with queries, and they are required to be unique and not null, providing some automatic validation of input data. Primary keys can also be used with foreign keys to provide referential integrity.
- Tables without Indexes: Lists tables that do not have any indexes. If a column in a table has an index defined on it, queries that use the column are usually much faster and more efficient than if there is no index on the column, especially if there are many rows in the table and many different data values in the column.
- Tables with Unindexed Foreign Keys: Lists any foreign keys that do not have an associated index. A foreign key is a column (or set of columns) that references a primary key: that is, each value in the foreign key must match a value in its associated primary key. Foreign key columns are often joined in queries, and an index usually improves performance significantly for queries that use a column. If an unindexed foreign key is used in queries, you may be able to improve run-time performance by creating an index on that foreign key.

- **XML Reports**

  These reports list information about XML objects.

  - XML Schemas: For each user that owns any XML objects, lists information about each object, including the schema URL of the XSD file containing the schema definition.

26.9 Troubleshooting Database Connections

This section contains information to help you if you have problems connecting to a database.

26.9.1 Deploying to a Database that Uses an Incompatible JDK Version

If you get the following ORA-29552: verification warning: `java.lang.UnsupportedClassVersionError` when deploying Java to the database you need to change the version of the JDK used for that project to a version compatible with that used by the database.

This version of JDeveloper uses Java JDK Version 1.6.

Table 26–1 JDK Version Used by Different Database Versions

RDBMS Version	JDK Version
9.2	1.3.1
10.2	1.4.2_04
11.1	1.5.0_10
11.2	1.5.0_10

For information about changing the Java SE on a project by project basis, see the section on setting the target Java SE in How to Set Properties for Individual Projects.

You can download previous releases of Java SE from http://www.oracle.com/technetwork/java/javase/downloads/previous-jsp-138793.html.

27

Designing Databases Within Oracle JDeveloper

This chapter describes how to:

- Work with database objects in a database connection.
- Work with offline databases in JDeveloper to create and edit offline database objects that can then be generated to a script database connection.

This chapter includes the following sections:

27.1 About Designing Databases Within Oracle JDeveloper

You can use JDeveloper to:

- Create, edit, or delete database objects
- Create, edit, or delete offline database objects
- Work with offline versions of database definitions, and then generate those definitions to a file that is processed immediately or at a time you choose to create a table or other database objects via the database connection
- Model offline databases, and model database objects in a live database connection on a diagram. For more information about modeling databases, see Section 23.5, "Database Diagram."

27.2 Creating, Editing, and Dropping Database Objects

You can create database objects and offline database definitions, you can edit those objects, and you can delete them or drop them a database connection.

27.2.1 Working with Offline Database Definitions

This section describes how to work with database objects, such as tables, views, constraints, outside the context of a database schema. Offline database is a technology in JDeveloper that allows you to create and edit database object definitions within a project, saved as `.xml` files, using the same editors that are used to create and edit database objects on live database connections.

You can create new offline database objects, or import them from a connection to a live database. After you have finished working with them, you can generate DDL that can be used to create and update database definitions in online database schemas.

The JDeveloper Offline database supports the following object types:

- Function
- Index
- Materialized View
- Materialized View Log
- Package
- Procedure
- Sequence
- Synonym
- Table
- Tablespace
- Trigger
- Type
- View

For more information about Oracle Database support of any of these object types, see the *Oracle Database SQL Language Reference*.

Working with Offline Database Definitions

When you work with offline database definitions in JDeveloper, you work with objects that are stored as XML files, but which provide a model of objects in a live database connections. You can generate offline database definitions to live database connections to create, alter, or drop database objects.

JDeveloper provides the tools you need to create and edit database objects such as tables and constraints outside the context of a database. For example

- You can create new tables and views and generate the information to a database.
- You can create new tables and views and generate the information to a file, which you can edit and later run on a database connection.
- You can import tables and views from a database schema, make the changes you want and then generate the changes back to the same database schema, to a new database schema, or to a file that you can run against a database at a later date. JDeveloper allows you to manually reconcile changes before committing them to a database.
- You can use the modeling tools in JDeveloper to visualize your offline database objects on a diagram. For more information about modeling databases, see Section 23.5, "Database Diagram."

How to Set Paths for Offline Database Files

You can configure a project's settings to specify the root locations for offline database objects available to that project. The database path is configured by default, so you only need to change it if you want to:

- Include offline database objects that are stored in another project

- Store new offline database object files somewhere else.

Offline database objects can be shared between projects by adding their file system location to the database path for a project. The order in which file system locations are entered in the database path signifies the order in which the directories are searched for offline database objects. The first location in the database path is the location in which new offline database object files are stored.

If you are modeling database objects, the model path (located on the Modelers preferences page in the Project Properties dialog) is used to specify the file location for the diagram.

> **Note:** When you are adding another database path to a project, you should save your work before proceeding. When you change the database path, the project reloads the offline database object definitions so any unsaved work for example, changes to tables, views, schemas or new objects that you have not yet saved, may be lost.

You can set a default root directory for database objects that will be used for all new projects.

To set the default root directory for database objects for new projects:

1. Choose **Application > Default Project Properties**.
2. Select **Project > Source Paths > Offline Database**, and enter the root directory.

You can change the database path for an existing project.

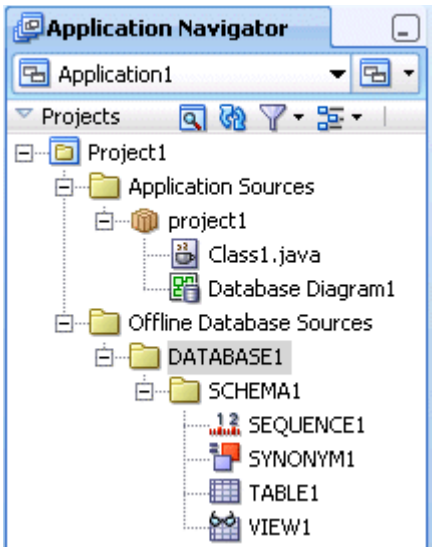
Setting the database path for an existing project:

1. Right-click the project and choose **Project Properties**.
2. Select **Project > Source Paths > Offline Database**, and enter the file system location for your project's offline database objects. Separate multiple file system locations using semicolons (`;`).
3. You can selectively include and exclude subfolders using the Included and Excluded tabs. For more information, press F1 or click **Help** from within the dialog.

27.2.1.1 Offline Databases

JDeveloper works with offline database definitions in the context of offline databases that act as containers in a similar way to packages. In the Application Navigator, the offline database is shown below the Offline Database Sources node, shown in Figure 27–1.

Figure 27–1 Offline Database in the Application Navigator



In this case, a Java class and a database diagram have been created in the package project1, which is under the Application Sources node, and some offline database definitions have been created in an offline database called `DATABASE1`, which is under the Offline Database Sources node.

When you create an offline database, you choose the database emulation the offline database should have.

27.2.1.2 Configuring Offline Database Emulation

You can specify the type of database an offline database emulates. This determines the data types supported in the project.

The Oracle Database options available are:

- Oracle 11g Database Release 1
- MySQL Database Server 4.1.x
- MySQL Database Server 5.x
- Oracle 11g Database Release 2 (default)
- Oracle Database 10g Release 1
- Oracle Database 10g Release 2
- Oracle Database 10g Express Edition Release 2
- Oracle Database 11g Express Edition Release 2
- Oracle Database 10g Lite Release 1
- Oracle Database 10g Lite Release 3
- Oracle 8i Server Release 3
- Oracle 9i Database Release 2
- TimesTen Database Server 11g
- TimesTen Database Server 6.0
- TimesTen Database Server 7.0

The non-Oracle database options available are:

- Apache Derby 10.5
- DB2 Universal Database 8.1
- DB2 Universal Database 9.5
- Generic JDBC Database
- Informix Dynamic Server 10.0
- Informix Dynamic Server 11.5
- Microsoft SQL Server 2005
- Microsoft SQL Server 2008
- SQLite Database 3.6
- Sybase Adaptive Server Enterprise 12.5
- Sybase Adaptive Server Enterprise 15

27.2.1.3 How to Create Offline Databases

An offline database is a node in the Application Navigator that contains offline schemas and offline database object definitions.

To create an offline database:

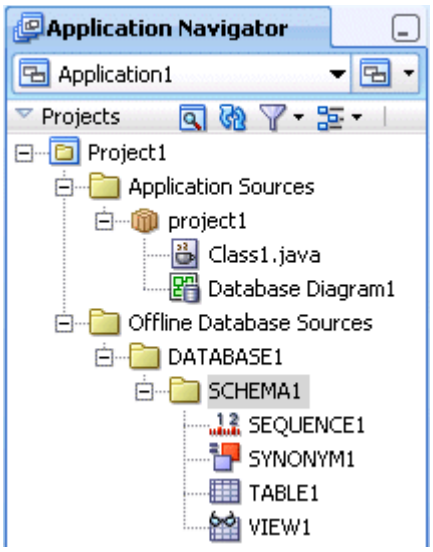
1. In the navigator, locate the project you want to work in.
2. Right-click a project or anything in it, and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**, and select **Offline Database**.
4. In the Create Offline Database dialog, enter a name for the offline database and choose the database type to emulate.

   For more information at any time, press F1 or click **Help** from within Create Offline Database dialog.

27.2.1.4 Offline Schemas

JDeveloper works with offline database definitions in the context of offline databases. Within the offline databases, schemas are the equivalent of schemas in live database connections. In the Application Navigator, the offline schema is shown below the Offline Database Sources node, illustrated in Figure 27–2.

Figure 27–2 Offline Schema in Application Navigator



In this case, a Java class and a database diagram have been created in the package `project1`, which is under the Application Sources node, and some offline database definitions have been created in a schema called `MYSCHEMA`, which is in an offline database called `DATABASE1` under the Offline Database Sources node.

27.2.1.5 How to Create Offline Schemas

To create an offline schema:

1. In the navigator, locate the project you want to work in.
2. Right-click a project or anything in it, and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**, and select **Schema**.
4. In the Offline Database dialog, choose the offline database to create the schema in.
5. In the Create Schema dialog, enter a name for the offline schema. For more information at any time, press F1 or click **Help** from within Create Schema dialog.

Context Menu Shortcut:

In the Application Navigator, right-click the offline database, and choose **New Schema**.

27.2.1.6 How to Create Offline Database Objects

You can create offline database objects from the New Gallery, or from the context menu of an offline database or offline schema in the Application Navigator. The process is similar regardless of the offline database object type.

About Tables

The types of tables that are available are:

- Normal. This is a regular database table which can be partitioned. A partitioned table is a table that is organized into smaller and more manageable pieces called partitions. SQL queries and DML statements do not need to be modified in order to access partitioned tables; however, after partitions are defined, DDL statements

can access and manipulate individual partitions rather than entire tables or indexes. Also, partitioning is entirely transparent to applications.

- External. An external table is a read-only table whose metadata is stored in the database but whose data is stored outside the database. Among other capabilities, external tables enable you to query data without first loading it into the database.
- Index Organized. An index-organized table is a table in which the rows, both primary key column values and non-key column values, are maintained in an index built on the primary key. Index-organized tables can be used to store index structures as tables in Oracle Database. Index-organized tables are best suited for primary key-based access and manipulation.
- Temporary. The temporary table definition persists in the same way as the definition of a regular table, but the table segment and any data in the temporary table persist only for the duration of either the transaction or the session, and the table is not stored permanently in the database. Temporary tables cannot be partitioned or index organized.

Note: You can only create and use relational table definitions in offline schemas, you cannot create and use object relational table definitions.

About Partitions

You can partition a table, an index, or a materialized view. A partitioned table or materialized view is a table or materialized view that is organized into smaller and more manageable pieces called partitions. SQL queries and DML statements do not need to be modified in order to access partitioned tables; however, after partitions are defined, DDL statements can access and manipulate individual partitions rather than entire tables or indexes. Also, partitioning is entirely transparent to applications.

Temporary tables cannot be partitioned.

A partitioned index consists of partitions containing an entry for each value that appears in the indexed column(s) of the table.

There are three types of partitions:

- RANGE, which partitions the table on ranges of values from the column list. For an index-organized table this must be a subset of the primary key columns of the table.
- HASH, which partitions the table using the hash method. Rows are assigned to partitions using a hash function on values found in columns designated as the partitioning key.
- LIST, which partitions the table on lists of literal values from a column. This is useful for controlling how individual rows map to specific partitions.

If you have a RANGE partition on a table or index, you can additionally subpartition the table or index by HASH or LIST. Composite partitioning is a combination of two partitioning methods to further divide the data into subpartitions.

You can define subpartition templates which will be used in any partition for which you do not explicitly define subpartitions.

Note: A table or index in Oracle Database which uses a hash partition by quantity will be displayed in JDeveloper as having individual hash partitions. You can either specify your individual partitions manually using the Create or Edit Table or Materialized View dialog, or define them by quantity and let the database do the work for you. Whichever way you choose to define your partitions, when you edit the database table or materialized view in JDeveloper, they will be displayed as individual partitions.

About Indexes

You can create indexes on columns in tables in order to speed up queries. Indexes provide faster access to data for operations that return a small portion of a table's rows. In general, you may want to create an index on a column in any of the following situations:

- The column is queried frequently.
- A referential integrity constraint exists on the column.
- A unique key integrity constraint exists on the column.

You can create an index on any column; however, if the column is not used in any of these situations, creating an index on the column does not increase performance and the index takes up resources unnecessarily. Although the database creates an index for you on a column with an integrity constraint, explicitly creating an index on such a column is recommended. You can use the SQL Worksheet's execution plan to show a theoretical execution plan of a given query statement.

Indexes can be normal, where the index is either non-unique, unique, or bitmap, or they can be domain indexes.

In a non-unique normal index, the index can contain multiple identical values. In a unique normal index, no duplicate values are permitted. Use a unique normal index when values are unique in the column. In a bitmap normal index, rowids associated with a key value are stored as a bitmap. These are useful for systems in which data is not frequently updated by many concurrent systems, or where there is a small range of values.

Domain indexes are user-defined indexes, each of which indexes data in an application-specific domain. They are built using the indexing logic supplied by a user-defined indextype. An indextype provides an efficient mechanism to access data that satisfy certain operator predicates. Typically, the user-defined indextype is part of an Oracle option, like the Spatial option.

Working with User-Defined Data Types

JDeveloper allows you to define your own data types, which can be either object types and collection types.

Object types are abstractions of the real-world entities—for example, purchase orders—that application programs deal with. An object type is a schema object with three kinds of components:

- A name, which serves to identify the object type uniquely within that schema.
- Attributes, which model the structure and state of the real world entity. Attributes are built-in types or other user-defined types.

- Methods, which are functions or procedures written in PL/SQL and stored in the database, or written in a language like C and stored externally. Methods implement operations the application can perform on the real world entity.

An object type is a template. A structured data unit that matches the template is called an object.

JDeveloper allows you to create an object type specification, or an object type specification and body.

When you create a new object type spec, it is similar to

```
TYPE TYPE1 AS OBJECT (a null );
```

When you create an object type body, it is similar to

```
CREATE TYPE TYPE1 AS VARRAY(1) OF null;
/
```

Collection types are different. Each collection type describes a data unit made up of an indefinite number of elements, all of the same data type. The collection types are array types and table types.

Array types and table types are schema objects. The corresponding data units are called VARRAYs and nested tables. When there is no danger of confusion, we often refer to the collection types as VARRAYs and nested tables.

Collection types have constructor methods. The name of the constructor method is the name of the type, and its argument is a comma-separated list of the new collection's elements. The constructor method is a function. It returns the new collection as its value.

An expression consisting of the type name followed by empty parentheses represents a call to the constructor method to create an empty collection of that type. An empty collection is different from a null collection.

JDeveloper allows you to create array types and table types.

n array type is similar to

```
TYPE TYPE1 AS VARRAY(1) OF null;
```

A table type is similar to

TYPE TYPE1 AS TABLE OF null;

Note: In order to use data types when the project is configured for database emulation other than Oracle Database, the database it emulates must support type creation.

About Materialized Views

Materialized views are database objects that contain the results of a query. The FROM clause of the query can name tables, views, and other materialized views. You can model, create, and edit materialized views in a live database connection, and offline materialized views in an offline database in JDeveloper.

When importing materialized views from Oracle Database to a JDeveloper project:

- If a materialized view on the database specifies WITHOUT REDUCED PRECISION, when it is imported into JDeveloper it will use reduced precision, and the Reduced Precision option on the Properties page of the Edit Materialized

View dialog is selected. If it is important that the materialized view does not reduce precision, select No Reduced Precision in the dialog.

- If a materialized view on the database specifies USING ROLLBACK SEGMENT and USING TRUSTED CONSTRAINTS, when it is imported into JDeveloper no rollback segment is selected on the Properties page of the Edit Materialized View dialog, and the constraint is shown as Enforced. If necessary, change the options in the Edit Materialized View dialog

Validating Date and Time Values

When you create offline table definitions or tables in a database and use date and time default values, JDeveloper validates these values. For a date, you can use:

- Oracle date functions
- A quoted string of the form DD-MON-RR, where:
  - The month can be spelled out in full.
  - The year can be written in full, e.g., 2011.
  - The separators (-) can be absent, or any non-alphanumeric character combined with spaces.

For a time stamp, you can use a quoted string of the form DD-MON-RR HH.MI.SSXFF AM TZR, where:

- The month can be spelled out in full.
- The year can be written in full, e.g., 2011.
- The hours and minutes must be present.
- Seconds, fractions of second, AM/PM, and time zone are optional.
- The separators (-) can be absent, or any non-alphanumeric character combined with spaces.

When you import tables from a database, date and time values are validated according to the rules above. If the validation prevents you from importing a table from Oracle Database, you can turn it off.

To turn off date and time validation:

1. Choose **Tools > Preferences > Database Connections**.
2. Uncheck **Validate date and time default values**.

To create an offline type definition:

1. In the Application Navigator, expand the workspace and project you want to work in.
2. Right-click a project or anything in it, or an offline schema or anything in it, and choose **New** to display the New Gallery.
3. From the New Gallery, expand Database Tier, and select Offline Database Objects.
4. Select **Type** to launch the Create Offline Type dialog.
5. Enter parameters and select options to define the type.

   For more information at any time, press F1 or click **Help** from within the Create Offline Type dialog.

Context Menu Shortcut:

In the Application Navigator, right-click the offline schema, select **New Database Object**, then select **New Type**.

You can edit user-defined types by double-clicking the type in the Application Navigator. The SQL comprising the type opens in the source editor.

To create offline database object definitions:

> **Note:** You can only create and use relational table definitions in offline schemas, you cannot create and use object relational table definitions.

1. n the Application Navigator, right-click a project or anything in it, and choose **New** to display the New Gallery.
2. From the New Gallery, expand Database Tier, and select **Offline Database Objects**.
3. Select the offline database object you want to create to launch the Create dialog or wizard.

   For more information at any time, press F1 or click **Help** from within the Create Offline Type dialog.

Context Menu Shortcut:

In the Application Navigator, right-click the offline schema, select **New Database Object**, then select object you want to create.

To drop an offline database definition:

1. In the navigator, expand the project, offline database, and offline schema containing the offline object. Right-click the offline object, and choose **Delete**.
2. Right-click the offline object, and choose **Delete**.

   Alternatively, right-click the offline table and choose **File > Delete**.

> **Note:** If the offline table has any dependencies, the Confirm Delete dialog warns you and allows you to see the usages. If you still choose to delete the offline table, the Cascade Confirm Delete dialog warns you which objects will also be deleted.

27.2.1.7 How to Import Offline Database Definitions Based on Database Objects

You can drag tables, views, materialized views, synonyms, and sequences from a database schema onto a database diagram, where they become accessible as offline database objects.

Another way of working with database objects as offline database objects is to import them from a database to an offline database.

To drag objects onto a database diagram:

1. Create a new database diagram.

   Alternatively, open an existing diagram.

2. Choose the database connection. Go to either:
   - **View > Database > Database Navigator**.
   - **Application Resources** in the Application Navigator.

   Expand IDE Connections or application, and select a database connection.
3. In the connection, expand the schema and expand the node you want: **Tables**, **Views**, **Materialized Views**, **Sequences**, or **Synonyms**.
4. Select the object you want to model, and drag it onto the database diagram. This opens the Specify Location dialog. Ensure that **Copy Objects to Project** is selected, and click OK. The object is now displayed on the diagram and listed in the Application Navigator.

You can drag more than one object of the same type onto a database diagram, by holding down the Ctrl key as you select them.

> **Note:** If you import the same object more than once a warning message is displayed. If you are using Copy to Project and choose to proceed, you can replace or delete the existing object. If you drag a database object onto the diagram and choose to proceed the new object overwrites the existing one.

How to Copy Database Objects and Offline Database Definitions to Projects

You can copy database objects from a database schema to an offline database where they become available as offline database objects. You can also copy offline database objects to a project.

If you try to copy database objects to an offline database which emulates a different database version you will see an error message giving you guidance on how to proceed. In general, it is a good idea to make sure that the offline database uses the same database emulation as the source database.

You can apply filters in the wizard to only display the objects you are interested in, and when there are a large number of objects in the schema you can turn off auto-query so that the wizard does not refresh every time you type a filter character.

You can apply filters to select the objects that are displayed as available for import. In the Object Picker page (step 3 of the wizard), you can:

- Enter characters in Name Filter to filter the list of available objects by name. Name Filter is case sensitive.
- When there are a large number of objects, you can turn off Auto-Query, and click Query after you have entered the filter you want to use.

To import database objects:

1. In the Application Navigator, select the project you want to work in.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Database Tier** and select **Offline Database Objects**.
4. In the **Items** list, double-click **Copy Database Objects to a Project** to launch the Copy Database Objects to a Project wizard.

   For more information at any time, press F1 or click **Help** from within the wizard.

27.2.1.8 Offline Tables and Foreign Keys

When you import a table from a live database schema to JDeveloper, information about any foreign keys will not necessarily be available. The following sections discuss how foreign key information is treated in different cases.

Importing Tables at Both Ends of a Foreign Key

This is the simplest case. When JDeveloper imports tables that have foreign keys between them, information about the foreign key is also imported. Therefore the foreign key is correctly shown on a database diagram and on the Constraint Information page of the Edit Offline Table dialog.

After you have finished working on the tables you can choose to generate your changes directly back to the database.

Importing a Table at One End of a Foreign Key

In the case that JDeveloper imports a table that has a foreign key, but the table at the other end of the key is not imported, an unvalidated foreign key reference (called a name-based foreign key) is used. The foreign key is displayed in the Constraints compartment of the owning table, but not as association lines on the database diagram. The table dialog will show the foreign key as broken. The name-based foreign key is not validated, but the DDL is correct so that changes can be generated directly to a database.

Best Practice

From the information above you can see that if you are importing tables to act as the basis of a new database schema, then you do not need to worry about foreign keys to tables that you are not interested in. You can safely make your changes and generate the online tables in a new schema.

However if you are importing tables so that you can make the changes you want and then generate the changes back to the same database schema you should either:

- Import all tables that have a foreign key relationship, whether you intend to change them or not, so that you generate the correct information about the foreign keys to a SQL file or directly to the database. The Choose Operation page of the Copy Database Objects to a Project wizard allows you to import dependencies.
- Import just the tables you want to change and rely on the name-based foreign keys to hold the information

27.2.1.9 How to Refresh Offline Database Objects

You can refresh any imported offline object from the database connection it was originally imported from. Note that if you reimport the object, you will lose any changes you have made in the offline object.

> **Note:** You cannot refresh an object that was created as an offline object in JDeveloper and then generated to the database. If you make changes to the object in the database and want those changes to be reflected in the offline object, you must import the object from the database and overwrite the offline object by selecting **Automatically replace existing objects** on the Select Target Schema page, or selecting **Yes** on the Confirm Overwrite message at the end of the wizard.

To reimport an object from a database connection:

1. Right-click the offline object in the Application Navigator, and choose **Refresh** from db-connection.
2. When the Confirm Offline Object Overwrite dialog appears, check that you want to reimport the object and then click **Yes**. Otherwise click **No**. This may take a few seconds.

27.2.1.10 How to Create Objects from Templates

You can create offline database objects based on templates, for example: To use a default set of storage options for all tables created. To use a default set of user property values for all tables created. To use a set of default columns for all tables created. A template table can create a default primary key, a column sequence, and a trigger.

When you create a new object using the template, the properties that are set on the template are copied to the new object and therefore pre-populate the options in the create dialog. When the namespace of owned objects is not the parent object, the name must be unique within the schema, not just within the parent object. For example, Index and Constraint names must be unique within the schema, not just within the owning Table, Materialized View, or View.

How to Create Offline Templates

You can create offline database objects from templates.

To create default templates for an offline database:

1. Create a new offline database.
2. In the Create Offline Database dialog, select **Initialize Default Templates**. When you click **OK**, the offline database is created, along with default template objects which have the name `TEMPLATE_object`. You can edit the template database objects by right-clicking the one you want and choosing **Properties**, which opens the Edit object dialog.

To edit the default templates for a new offline database:

1. If you are creating a new offline database, select **Edit Default Templates** in the Create Offline Database dialog.
2. When you click **OK**, the Default Templates dialog opens where you can edit the default template for each object type. For more information at any time, press F1 or click **Help** from within the Default Templates dialog.

To edit the default templates for an existing offline database:

1. In the Application Navigator, right-click the offline database and choose Properties.
2. Navigate to the Default Templates page of the dialog, where you can edit the default template for each object type. For more information at any time, press F1 or click **Help** from within the Edit Offline Database dialog.

How to Create Offline Database Objects from Templates

You can create offline database objects from templates as offline database objects in the Application Navigator, or as modeled offline database objects on a database diagram.

Before creating offline database objects based on templates, you first need to create the templates.

To create an offline database object based on a template:

1. In the navigator, right-click a project or anything in it, and choose **New** to display the New Gallery.
2. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**.
3. Select **Database Object from Template** to launch the Choose Template Object dialog. For more information at any time, press F1 or click **Help** from within the Create object dialog.
4. Choose the object type to create from a template. When you click **OK**, the Create object dialog opens, pre populated with the values in the template. For more information at any time, press F1 or click **Help** from within the Create object dialog.

   When you click **OK** the object is created and listed in the Application Navigator under the offline database and schema.

You can also open the Choose Template Object dialog from the context menu of an offline database, or any node under it, in the Application Navigator.

To quickly create an offline database object based on a template:

1. Right-click the offline database node in the Application Navigator, or any node under it, and choose **New Database Object > From Template**.

To quickly create an offline database object based on another offline database object:

1. In the Application Navigator, right-click the offline database object you want to base a new object on and choose Use as Template.
2. The Edit dialog for the type of object opens, pre populated with the properties of the source object.

27.2.1.11 Working with User Property Libraries

You can add user defined properties to database objects. For an instance of a database object; these user defined properties can be assigned specific values.

You can work with libraries that can limit the properties you can define for an offline database object. For example, you can determine that all tables must have a column of a particular type, or that only certain values are allowed, or that a property is mandatory.

User property libraries are defined in the context of an offline database. First you have to define user properties for the types of offline database objects you want to use. Then you can use the libraries when creating offline database objects in the offline database that the library is defined for.

User property libraries can contain properties defined for:

- Tables, columns, constraints, indexes
- Functions, packages, procedures
- Materialized Views
- Materialized View Logs
- Sequences
- Synonyms

- Tablespaces
- Triggers
- Types
- Views

27.2.1.11.1 How to Create and Edit User Property Libraries

User property libraries are independent of offline databases, but can be added to them using the offline database edit dialog.

To create or use a user property library:

1. In the Application Navigator, expand the project, right-click the offline database and choose Properties.
2. In the Edit Offline Database dialog, choose User Property Libraries.
3. You can:
   - Create a new library. In this case, you enter a filename and location for the library.
   - Add a library that exists in the file system. In this case, you browse to the library location on the file system.
   - Edit an existing library by selecting it from the list.
4. Enter values for the user properties in the Edit User Property Library File dialog. For more information at any time, click F1 or press **Help** in the dialog.

Once you have created a user property library for an offline database object type, you can use it to store user property value specific to that instance of the object.

You can provide validation for the user defined property value to validate the database objects by writing your own validation code. This is an advanced procedure which is outside the scope of this user guide. For more information see `UserPropertyValidationManager` in *Oracle Fusion Middleware Java API Reference for Oracle Extension SDK*. For information about using the Extension SDK, see *Oracle Fusion Middleware Developer's Guide for Oracle JDeveloper Extensions*.

27.2.1.11.2 How to Use User Property Libraries

Use user properties for database objects

Before you can use user properties in offline database objects, you must define one or more user property libraries for the offline database you are working in.

To use user properties in an offline database object:

1. Create the offline database object.
2. Navigate to the User Properties page or tab of the offline database object dialog, and enter values for the user properties.

27.2.1.12 How to Generate Offline Database Objects to the Database

The Generate SQL from Offline Database Objects wizard allows you to choose how to update a database schema with the offline objects that you have created or edited. You can:

- Create or replace the objects in the database.

  If you choose to generate a SQL file, it will contain CREATE and DROP statements.

- Update existing database schema objects with the changes you have made to the offline database objects. JDeveloper first reconciles the offline database definitions against the objects in the database schema to identify the changes necessary. You can choose to do a manual reconcile and select only some of the changes.

  If you choose to generate a SQL file, it will contain ALTER statements.

Whether you are generating changes to a database, or reconciling changes, you can choose to:

- Generate a SQL file that you can examine, and run against the database later.
- Make the changes directly to the database.
- Make the changes to the database and also generate a SQL file.

Alternatively, if you just want to generate one or more offline tables back to the database connection they were originally imported from, you can do this directly from the navigator.

> **Note:** If you have made changes to tables that have foreign keys, it is possible that the foreign keys will be dropped when you generate your changes to the database.

27.2.1.12.1 Reconciliation issues This section contains information about problems you may come across when reconciling.

27.2.1.12.2 Cannot modify constraints Constraints can be created or dropped during reconciliation; they cannot be modified. The only `ALTER TABLE` reconciliations that can be performed are `ADD CONSTRAINT`, `DROP CONSTRAINT`, `ADD COLUMN`, `DROP COLUMN`, and `WIDEN COLUMN`.

27.2.1.12.3 Cannot reconcile renamed tables You can change the name of a table when you import it or while you are editing it offline. If you try to reconcile the renamed table back to the database, you will receive an error message because the database does not have a record of the table with its new name.

To avoid this, create the renamed table in the database, do not reconcile or replace it.

How to Generate Database Definitions to a Database

27.2.1.12.4 How to Generate Database Definitions to a File Create a SQL file containing the CREATE and DROP statements that you can run against an online database schema.

Note: If you have made changes to tables that have foreign keys, it is possible that the foreign keys will be dropped when you generate your changes to the database.

If you have one or more offline database definitions containing information that you want to generate to a database, you can use this method. However if you want to quickly generate one or more offline database definitions back to their original database connection, you can do this from the navigator.

When you have an offline version of an online database table, JDeveloper keeps track of the information comprising the offline database table columns behind the scenes. When the database is updated outside JDeveloper, for example when the generated SQL script is run in a SQL session, or when another user updates the database, JDeveloper cannot track the link between the offline database table and the table in the database. To get around this, you must refresh the offline schema objects from the database.

To create the file:

1. In the Application Navigator, expand the application and project.
2. Right-click an offline schema and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**.
4. Select **SQL Generated from Offline Database Objects** to launch the Generate SQL from Offline Database Objects wizard.
5. On the page specify details for the generated file, then click **Next**.
6. On the **Finish** page, click Finish to create the file.

Context Menu Shortcut:

In the navigator, right-click one or more offline database definitions and choose **Generate to**

or On the database diagram select one or more modeled database definitions, right-click and choose **Synchronize with Database > Generate To**.

27.2.1.13 Renaming Offline Database Objects

JDeveloper has a limited ability to keep track of renamed offline database objects such as tables and sub objects such as columns or constraints. In some circumstances JDeveloper will drop the database object with the unchanged name and create a new database object with the new name, which can lead to loss of data. You need to be aware of the situations when this can arise so as to avoid them.

This can occur when an offline database object is generated to a database connection. If you then change the name of the offline database object or of a sub object such as a column or index, and then generate the changed offline database object to a database connection, in the database the object with the original name is dropped and a new object using the new name is created.

A different situation which can lead to loss of data is when an object is imported from a database connection, then the name of an offline database sub object is changed. In this case, the first time you generate to a database connection the database sub object is correctly updated. However if you attempt to generate to the database connection a

second time the sub object with the original name is dropped and a new database sub object with the new name is created. The reason that this happens is because Copy to Project uses the original name in an internal reference to the online sub object.

27.2.1.14 Using Offline Database Reports

JDeveloper provides many reports about an offline database and its objects. You can also create your own user-defined reports for offline database objects.

27.2.1.14.1 Offline Database Reports JDeveloper comes with a set of pre-built report definitions, and you can also define your own report definitions.

You can use the pre-built reports directly to provide information about an offline database, or you can alter them to create a report tailored to your specific requirements.

Once you have created a pre-built report, you can examine the SQL that makes up the query for the report, and if necessary change it. You can also set parameters in the report query that are called when the report is run.

27.2.1.14.2 How to Use Pre-built Reports The pre-built reports quickly provide useful reports which provide the following queries for an offline database:

- OBJECT_COUNT, which lists the number of schema objects of each object type.
- OBJECT_LIST, which lists all schema objects in the offline database.
- TABLE_COLUMNS, which lists all tables with their column information.
- TABLE_COLUMN_COUNT, which lists all tables with their column count.
- TABLE_NO_PKS, which displays all tables that do not have a primary key.

When you run the Pre-Built Reports wizard, a separate file is generated for each of the pre-built reports that you choose to the location that you specified, and the file is listed in the Application Navigator under Resources.

You can examine the query for one of the pre-built reports by creating a new offline database report definition, and basing it on the pre-built report. You can also create a new report based on a pre-built report.

> **Note:** If you specify a location that is outside the current project the reports are generated, but they are not listed in the Application Navigator. The files have the file name pre-built-report.report, and they are structured as XML files.

How to use predefined reports:

1. In the Application Navigator, expand the application and project.
2. Right-click an offline database and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**.
4. Select **Pre-Built Reports** to launch the Pre-Built Reports wizard.
5. Choose the reports you want to generate and if necessary click **Next** to change the offline database that you want to run the report on.

6. Click **Finish**. The reports you have chosen are listed in the Application Navigator under the offline database node.

To edit a predefined report:

1. In the Application Navigator, expand the application, project and offline database.
2. Right-click the report and choose **Properties** to open the Edit Report dialog, where you can examine and change the properties.
3. On the Report Definition page, change the name of the report. Change other details as required, for example, you can change the offline database that the report is to run on.
4. To change the SQL query for the report, either change the SQL on the Query Definition page, or expand the Query Definition node and declaratively define the SQL query. You can use the **Check Syntax** button on the Query Definition page to check that the SQL parses.
5. To add parameters to the query, use the Report Parameters page.
6. To change the format that the report is published in, use the Publish Report page.

To run a pre-built report:

- In the Application Navigator, right-click the report and choose **Run**. The report is run against the offline database you specified. The results are either displayed in the Reports Log window (default), or in the location and format that you have chosen in the Publish Report page of the Edit Report dialog.

27.2.1.14.3 How to Define Report Definitions

You can define your own report definitions. You can either define a query from scratch, or you can base the new report definition on an existing report or on one of the pre-built reports.

You can specify that just the report definition is produced, or you can specify that when the report definition is run a comma-separated file is produced, or that a formatted HTML document is produced.

If you choose to generate an HTML document, you can optionally specify that a CSS file is used, and you can edit the default boilerplate text that formats the body of the HTML document.

How to create a report:

1. In the Application Navigator, expand the application and project.
2. Right-click an offline database and choose **New** to display the New Gallery.
3. From the New Gallery, expand **Database Tier**, and select **Offline Database Objects**.
4. Select **Reports** to launch the Create Report wizard.
5. Enter a name for the report, and choose whether to copy report details from a pre-built report, from an existing report, or whether to create a new report from scratch.
6. Change the offline database the report is to run on in the Offline Database page.
7. Create or examine the SQL query for the report in the Query Definition page. You can use the pages under the Query Definition node to declaratively create the SQL Query.

8. If you want to use parameters with the report, enter them in the Report Parameters page.
9. Click **Finish**. The new report is listed in the Application Navigator under the offline database node.

To run a report:

- In the Application Navigator, right-click the report and choose **Run**. The report is run against the offline database you specified. The results are either displayed in the Reports Log window (default), or in the location and format that you have chosen in the Publish Report page of the Edit Report dialog.

27.2.1.14.4 How to Use Boilerplate Text with HTML Reports JDeveloper provides some boiler-plate code to help you to format the report, and it includes three new HTML tags:

- <report/>, which defines the report output.
- <query/>, which defines the text of the query used to generate the report.
- <rows/>, which is the number of rows in the report.

The boiler-plate code provided is:

Example 27–1 Boiler-plate code for User-Defined Reports

```
<h1>Table Report</h1>
<p>Query used:</p>
<pre><query/></pre>
<p>The report output is:</p>
<report/>
<p>Report complete. <rows/> row(s) returned.</p>
```

You can edit this in the Create or Edit Report dialog to customize the report.

27.2.1.14.5 How to Edit User-Defined Reports You can change the properties of defined reports:

- Change the name of the report, or the directory where it is stored.
- Change the database connection you want to use.
- Use parameters to define a query for the report.
- Choose the format that the report should be published in.

To edit a report:

1. In the Application Navigator, expand the application, project and offline database.
2. Right-click the report and choose **Properties** to open the Edit Report dialog, where you can examine and change the properties.

27.2.1.15 Transforming from a UML Model

You can transform a UML Class model to an offline database model using the Offline Database Objects from UML Class Model wizard. For more information, see Section 22.7.1, "How to Transform UML and Offline Databases."

27.2.1.16 Working with Offline Database Objects in Source Control Systems

Developer provides a number of features for developing in teams, including several version control software systems. These are described in Chapter 6, "Versioning Applications with Source Control".

Offline table definitions can be version controlled and shared using a source control system. JDeveloper provides a compare tool optimized for working with offline table definitions:

- You can compare any offline db object. You can either compare with previous version, or get a full version history and compare any two versions.
- You can track name changes and the identity of objects.
- You can check for consistency, for example:
  - Ensuring that a column which is used in a key is not dropped.
  - That a constraint which uses an absent column is not added.
  - That a primary key column cannot be optional.

Note: While you can only compare versions of offline database objects using a source control system, for example to see the dependency of a constraint on a column, you can manually reconcile changes before committing them to a database using the Generate SQL from Offline Database Objects wizard. For more information, see Section 27.2.1.12, "How to Generate Offline Database Objects to the Database."

27.2.2 Working with Database Objects

You can create database objects in a database connection in the Database Navigator.

You must have a database connection in order to create database objects, and the user name used to create that connection must have the privilege to create the database object, either by having been granted the appropriate privileges (CREATE, DROP, and so on) or having been granted a role such as administrator that contains the privilege.

For more information about Oracle Database objects, see the *Oracle Database SQL Language Reference*.

To create a database object in the Database Navigator:

1. If necessary, choose **View > Database > Database Navigator**.
2. Expand IDE Connections or *application*, and expand the database connection.
3. Navigate to the node for the database object type you want to create. Right-click and choose **New object** from the context menu.

   Alternatively, click **File > New** to open the New Gallery. In the New Gallery, expand Database Tier, and select **Database Objects**. Select the offline database object type you want to create to launch the Create dialog or wizard.
4. Complete the Create object dialog.

   For more information at any time, press F1 or click **Help** from within the Create object dialog.

To edit a database object:

1. If necessary, choose **View > Database > Database Navigator**.
2. Expand IDE Connections or *application*, and expand the database connection, and navigate to the node and database object you want to edit.
3. Right-click and choose **Edit** to open the Edit object dialog.

   For more information at any time, press F1 or click **Help** from within the Create object dialog.

To drop a database object:

1. If necessary, choose **View > Database > Database Navigator**.
2. Expand IDE Connections or *application*, and expand the database connection, and navigate to the node and database object you want to drop.
3. Right-click and choose **Drop**.

27.2.3 Using Database Reports

JDeveloper provides a number of predefined reports about the database and its objects. You can also create your own user-defined database reports.

Database reports that query the database for latest information are run from the Database Reports Navigator. For more information, see Section 25.3, "Using the Database Reports Navigator."

27.3 Creating Scripts from Offline and Database Objects

You can generate database objects and offline database definitions to SQL scripts. You can also generate tables to Oracle MetaBase (OMB) scripts, which can be imported into Oracle Warehouse Builder.

27.3.1 How to Create SQL Scripts

You can create SQL scripts from offline database definitions in the Application Navigator, or from database objects in the Database Navigator.

To create a SQL script from a database object in the Database Navigator or from an offline database object in the Application Navigator:

1. Either choose **View > Database > Database Navigator**, expand the database connection and schema, and right-click the database object you want to create the script from

   or

   Choose **View > Application Navigator**, navigate to the offline database definition you want to create the script from.
2. Choose **Generate to SQL script**.
3. The Generate SQL from Database Objects dialog opens at the Choose Operation page. For more information at any time, press F1 or click **Help** in the dialog.
4. Complete the dialog. The script is created and opened in the SQL Worksheet.

27.3.2 How to Create OMB Scripts from Tables

You can create scripts formatted as Oracle MetaBase (OMB) scripts for Oracle Warehouse Builder from offline tables in the Application Navigator.

The default name of the script is `omb_scriptn.tcl`, and it is generated into the current project and appears in the navigator under the same offline database as the tables from which is it generated.

To create a OMB script from a table in the Application Navigator:

1. Choose **View > Application Navigator**, navigate to the offline table or tables you want to create the script from.
2. Choose **Generate to OMB script**. The script is created and opened in the source editor.

28

Using Java in the Database

JDeveloper supports features that allow you to write and execute Java programs that access Oracle Databases.

This chapter includes the following sections:

28.1 About Using Java in the Database

There are three aspects to using Java in the database:

- Using SQLJ or JDBC, both of which can be used to embed SQL in Java programs.
- Accessing database objects and PL/SQL packages from Java programs.
- Using Java stored procedures, which are Java methods that reside and run inside the database.

28.2 Choosing SQLJ or JDBC

JDeveloper supports two mechanisms for embedding SQL in Java programs:

- SQLJ: If you know the PL/SQL tables and columns involved at compile time (static application), you can use SQLJ. SQLJ is an industry standard for defining precompiled SQL code in Java programs.

  SQLJ allows you to code at a higher level than JDBC, by embedding SQL statements directly in your Java code. The SQLJ precompiler that is integrated into JDeveloper translates the SQL into Java plus JDBC code for you. SQLJ with JDeveloper lets you write and debug applications much faster than you can using just JDBC.

- JDBC: If you require fine-grained control over database access, or if you are developing an application that requires precise information about database (or instance) metadata, you can code your application entirely in Java using the JDBC API.

You can mix JDBC calls with SQLJ statements in your program. One way to do this is through connection context sharing.

28.2.1 Using SQLJ

SQLJ is a standard way to embed static SQL statements in Java programs. SQLJ applications are portable and can communicate with databases from multiple vendors using standard JDBC drivers.

SQLJ provides a way to develop applications both on the client side and on the middle-tier that access databases using Java. Developing in SQLJ is fast and efficient, and JDeveloper completely supports SQLJ development. You can create or include SQLJ files in your JDeveloper projects. When you compile a project that contains SQLJ source files, JDeveloper automatically calls the SQLJ translator, or precompiler. The translator produces completely standard Java source code, with calls to JDBC methods to provide the database support. JDeveloper then compiles the Java that the SQLJ translator generates.

For more information, see the *Oracle Database SQLJ Developer's Guide*.

28.2.2 Using Oracle JDBC Drivers

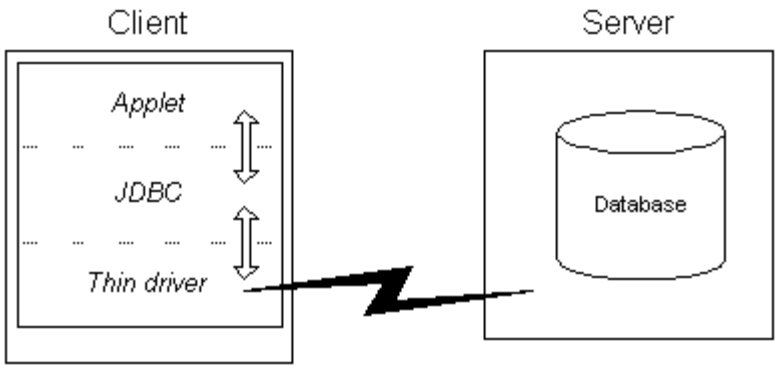
JDBC provides Java programs with low-level access to databases.

Oracle JDBC drivers can be grouped into two main categories with the following attributes:

- Java-based drivers (thin client / Type 4 driver):
  - are implemented entirely in Java
  - are highly portable
  - can be downloaded from the server system to a web browser
  - can connect using the TCP/IP protocol
  - are the only option for applets (due to security restrictions)
- OCI-based drivers (Type 2 driver):
  - are implemented using native method libraries (OCI DLLs)
  - have OCI libraries that must be available on the client system
  - cannot be downloaded to a browser
  - can connect using any Net8 protocol
  - deliver high performance

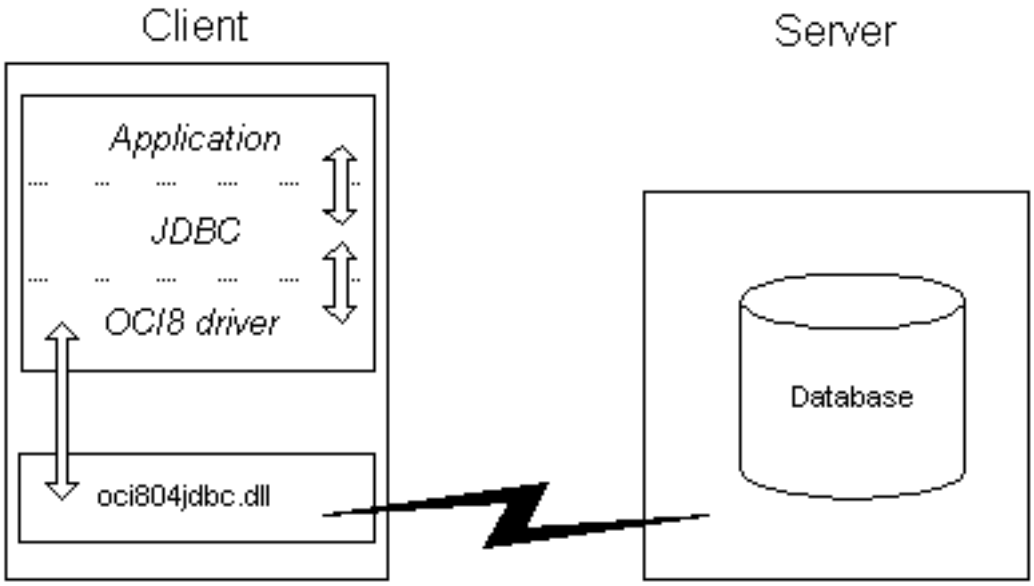
  The following figure illustrates how JDBC components and the driver run in the same memory space as an applet.

Figure 28–1 JDBC Components



The following figure illustrates how the Oracle JDBC OCI drivers run in a separate memory space from your Java application. These JDBC drivers make OCI calls to a separately loaded file.

Figure 28–2 Oracle JDBC OCI Drivers



Note: Take care not to confuse the terms JDBC and JDBC drivers. All Java applications, no matter how they are developed or where they execute, ultimately use the JDBC-level drivers to connect to Oracle. However, coding using the pure JDBC API is low-level development, akin to using the Oracle Call Interface (OCI) to develop a database application. Like the OCI, the JDBC API provides a very powerful, but also very code-intensive, way of developing an application.

28.2.3 SQLJ versus JDBC

How does SQLJ compare to JDBC? Here are some of the advantages that SQLJ offers over coding directly in JDBC:

- SQLJ programs require fewer lines of code than JDBC programs. They are shorter, and hence easier to debug.
- SQLJ can perform syntactic and semantic checking on the code, using database connections at compile time.
- SQLJ provides strong type-checking of query results and other return parameters, while JDBC values are passed to and from SQL without having been checked at compile time.
- SQLJ provides a simplified way of processing SQL statements. Instead of having to write separate method calls to bind each input parameter and retrieve each select list item, you can write one SQL statement that uses Java host variables. SQLJ takes care of the binding for you.

However, JDBC provides finer-grained control over the execution of SQL statements and offers true dynamic SQL capability. If your application requires dynamic capability (discovery of database or instance metadata at runtime), then you should use JDBC.

28.2.4 Embedding SQL in Java Programs with SQLJ

You have to perform a number of tasks to embed SQL in Java programs with SQLJ.

28.2.4.1 How to Create SQL Files

You can create a new SQL (.sql) file and add it to the current project.

To create a SQL file:

1. In the Application Navigator, select the project.
2. From the main menu, choose **File > New** to open the New Gallery.
3. In the New Gallery, in the **Categories** tree, select **Database Tier** then **Database Files.** In the **Items** list, double-click **SQL File**.
4. In the Create SQL File dialog, provide the details to describe the new file.

   For more information at any time, press F1 or click **Help** from within the dialog.
5. Click **OK**.

An empty SQL file is added to the current project and opened for editing.

28.2.4.2 How to Create SQLJ Classes

Create a new SQLJ (.sqlj) file and add it to the current project.

To create a new SQLJ file:

1. In the Navigator, select the project.
2. From the main menu, choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Database Tier** and select **Database Files.**

   For more information at any time, press F1 or click **Help** from within the dialog.
4. In the Items list, double-click **SQLJ Class** to open the Create SQLJ Class dialog.
5. In the Create SQLJ File dialog, provide the details to describe the new file.

   For more information at any time, press F1 or click **Help** from within the dialog.
6. Click **OK**.

A skeleton SQLJ class will be added to the current project and be opened for editing.

28.2.4.3 How to Compile SQLJ Classes

You can compile SQLJ classes into Java `.class` files.

To compile a SQLJ class:

1. Set the project's SQLJ translator options to control how the file is compiled.
2. In the Application Navigator, locate and select the SQLJ class.
3. Right-click the class, and choose **Make**.

The status bar at the bottom of the JDeveloper window shows the result of the compilation. Errors, if any, are listed in the log window.

28.2.4.4 How to Use Named SQLJ Connection Contexts

A SQLJ executable statement can designate a connection context object that specifies the database connection where the SQL operation in that clause will execute. If the SQLJ statement omits the connection context clause, then the default connection context is used.

28.2.4.5 How to Declare a SQLJ Connection Context Class

A connection context is an object of a connection context class, which you define using a SQLJ connection declaration.

To declare a context class:

1. Declare a context class.

   Named connection contexts are not required: SQLJ statements that omit the connection context name use the default connection context.

   For example, this statement declares the context class `MyConnectionContext`:

   ```
   #sql context MyConnectionContext;
   ```

   Context classes extend `sqlj.runtime.ref.ConnectionContextImpl` and `implement sqlj.runtime.ConnectionContext`.

After you have declared a context class, create a context object.

28.2.4.6 How to Create a Connection Context Object

Before it can be used in an SQLJ statement, a declared connection context must be created.

To create a context object:

1. Named connection contexts are not required: SQLJ statements that omit the connection context name use the default connection context.

   For example, use this statement to create an instance `thisCtx` for the connection context class `MyConnectionContext`:

   ```
   MyConnectionContext thisCtx = new MyConnectionContext (myPath, myUID, myPasswd,
   autocommit
   ```

28.2.4.7 How to Debug SQLJ Classes

You debug SQLJ code by debugging the SQLJ source directly, not the generated Java code.

SQLJ is debugged in JDeveloper in the same manner as other source code.

For more information, see the *Oracle Database SQLJ Developer's Guide*.

28.2.4.8 How to Debug SQLJ Classes

You debug SQLJ code by debugging the SQLJ source directly, not the generated Java code.

SQLJ is debugged in JDeveloper in the same manner as other source code.

For more information, see the *Oracle Database SQLJ Developer's Guide*.

28.2.4.9 How to Set SQLJ Translator Options

You can control the translation of SQLJ classes through the controls in the Project Properties dialog:

- Provide syntactic as well as semantic checking of SQL code.
- Provide syntax and type checking on the SQL statements.
- Test the compatibility of Java and SQL expressions at compile time.
- Specify a connection to a database server.
- Check the semantics of your SQL statements against the database schemas specified by connection contexts.

To set the SQLJ translator options:

1. In the Application Navigator, select the project that contains the SQLJ file.
2. Choose **Application > Project Properties > Compiler** and select SQLJ.
3. In the SQLJ panel, set the compilation options.
4. Click **OK**.

You can set SQLJ translator properties for all projects by choosing **Default Project Properties** from the Application menu

28.2.4.10 How to Use SQLJ Connection Options

SQLJ connection options specify the database connection for online checking. The general form for connection options is

```
-option@context=value
```

where `option` is one of the four options listed below.

The context tag is a connection context type, which permits the use of separate exemplar schemas for each of the connection contexts. If you omit the connection context type, the value will be used for any SQL statements that use the default connection context. The driver option does not allow a context tag.

The options are:

- `user` This option specifies the user name for connecting to a database in order to perform semantic analysis of the SQL expressions embedded in a SQLJ program. It contains the user name, for example:

```
-user=hr
```

The user command line option may include a connection context type. For example:

```
-user@Ctx1=hr
```

Whenever a user name is required for the connection to a database context Ctx1, SQLJ uses the user option that was tagged with `Ctx1`. If it can not find one, SQLJ issues a message and looks for an untagged user option to use instead.

Specifying a user value indicates to SQLJ that online checking is to be performed. If you do not specify the user option, SQLJ does not connect to the database for semantic analysis. There is no default value for the user option.

If you have turned on online checking by default (by specifying, for example, `-user=hr`), then in order to disable online checking for a particular connection context type `Ctx2`, you must explicitly specify an empty user name, for example:

```
-user@Ctx2Z
```

- `password` This option specifies a password for the user. The password will be requested interactively if it is not supplied. This option can be tagged with a connection context type. Examples of the two forms are:

  ```
  -password=hr
  -password@Ctx1=hr
  ```

- `url` This option specifies a JDBC URL for establishing a database connection. The default is `jdbc:oracle:oci9:@`. This option can be tagged with a connection context type. For example:

  ```
  -url=jdbc:oracle:oci8:@ -url@Ctx1=jdbc:oracle:thin:@<local_host>:1521:orcl
  ```

- `driver` This option specifies a list of JDBC drivers that should be registered in order to interpret JDBC connection URLs for online analysis. The default is `oracle.jdbc.driver.OracleDriver`. For example:

  ```
  -driver=sun.jdbc.odbc.JdbcOdbcDriver,oracle.jdbc.driver.OracleDriver
  ```

  This option cannot be tagged with a connection context type.

28.2.5 Embedding SQL in Java Programs with JDBC

JDBC provides Java programs with low-level access to databases.

For more information, see the *Oracle Database SQLJ Developer's Guide*.

28.2.5.1 How to Choose a JDBC Driver

JDBC uses a driver manager to support different drivers, so that you can connect to multiple database servers. To connect your database application to a data server, you must have available the appropriate JDBC driver. JDeveloper provides the Oracle Thin and OCI JDBC drivers. OCI for Oracle is the default driver. If you wish you may install a non-default JDBC driver.

Consider the following when choosing a JDBC driver to use for your application or applet:

- If you are writing an applet, you must use the JDBC Thin driver. JDBC OCI-based driver classes will not work inside a Web browser, because they call native (C language) methods.

> **Note:** When the JDBC Thin driver is used with an applet, the client browser must have the capability to support Java sockets.

- If you are writing a client application for an Oracle client environment and need maximum performance, then choose the JDBC OCI driver.
- For code that runs in an Oracle server acting as a middle tier, use the server-side Thin driver.

> **Note:** JDeveloper does not supply the server-side Thin driver.

- If your code will run inside the target Oracle server, then use the JDBC server-side internal driver to access that server. You can also access remote servers using the server-side Thin driver.

> **Note:** JDeveloper does not supply the server-side Thin driver.

- If performance is critical to your application, you want maximum scalability of the Oracle server, or you need the enhanced availability features like TAF or the enhanced proxy features like middle-tier authentication, then choose the OCI driver.

28.2.5.2 How to Modify a Project to Use a Non-Default JDBC Driver

If your JDeveloper programming environment as been modified to allow the use of a non-default JDBC driver, you can modify the current project to use the new driver by performing these steps.

To modify the project:

1. In the Application Navigator, select the project.
2. Choose **Application > Project Properties > Profiles > Development > Libraries.**
3. Select the driver's library from the list displayed, and transfer it to the **Selected Libraries** list. The driver's library was created when you registered the driver.
4. If necessary, order the list of selected libraries so that the library you have just added appears before other driver libraries, or libraries that pull in other driver libraries. These include:
   - Oracle JDBC
   - Enterprise Java Beans

   If necessary, select the library you added and drag it up to the top of the list.
5. Click **OK** to save your changes and close the dialog.

28.2.5.3 How to Code a JDBC Connection

You can establish a database connection in pure JDBC code.

A summary is given here, but for more information, see "Getting Started" in the *Oracle Database JDBC Developer's Guide*.

To code a JDBC Connection:

1. Import the JDBC classes using the statement

   ```
   import java.sql.*;
   ```

   This statement is required for all JDBC programming.

2. Register the JDBC drivers. If you are using an Oracle JDBC driver and use a constructor that uses the static `Oracle.connect()` method to set the default connection, the Oracle JDBC drivers are automatically registered.

   Alternatively, if you are using an Oracle JDBC driver, but do not use `Oracle.connect()`, then you must manually register the Oracle Driver class using the statement

   ```
   DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
   ```

   If you are not using an Oracle JDBC driver, then you must register an appropriate driver class:

   ```
   DriverManager.registerDriver(new mydriver.jdbc.driver.MyDriver());
   ```

   In any case, you must also set your connection URL, user name, and password.

3. Get a connection to a data server using a `getConnection()` method, for example

   ```
   Connection conn = DriverManager.getConnection(parameters...);
   ```

28.3 Accessing Oracle Objects and PL/SQL Packages using Java

Use Oracle JPublisher to access Oracle objects and PL/SQL packages from your Java programs. Oracle JPublisher lets you specify and customize the mapping of Oracle object types, reference types, and collection types to Java classes in a strongly typed paradigm

You can use JPublisher to access Oracle objects and PL/SQL packages from your Java programs. JPublisher lets you specify and customize the mapping of Oracle object types, reference types, and collection types to Java classes in a strongly typed paradigm.

Also, SQLJ programmers who want to call stored procedures declared in PL/SQL packages can use JPublisher to generate SQLJ wrapper classes for the packages. The SQLJ wrapper classes let you invoke the PL/SQL stored procedures, and pass and return values from them, directly from your SQLJ program.

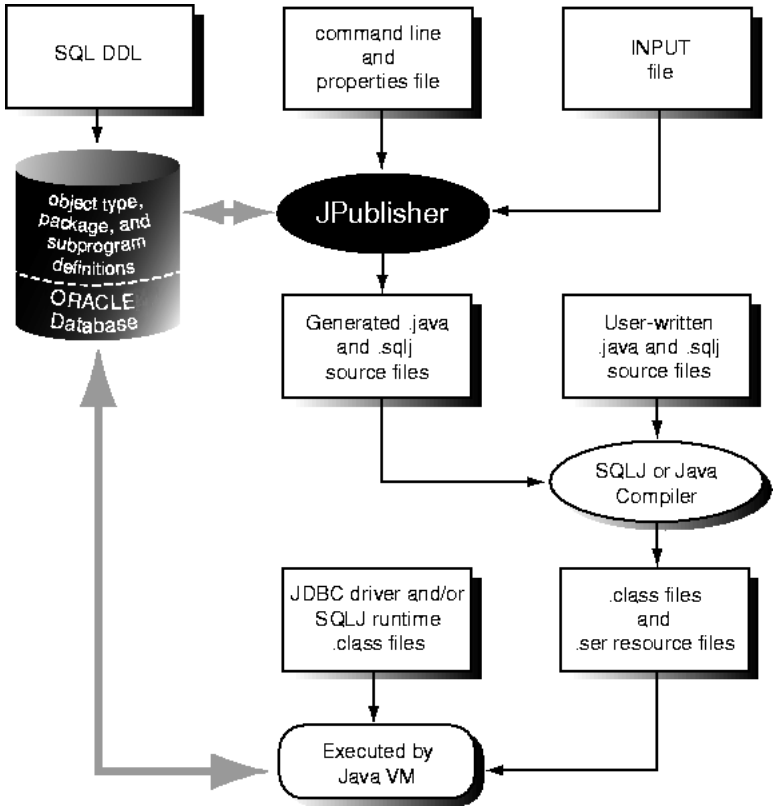
To access Oracle objects and PL/SQL packages using Java:

1. Create the desired object data types (Oracle objects) and PL/SQL packages in the database. It is recommended that any custom classes or interfaces you use in Oracle Database implement the `oracle.sql.CustomDatum` interface.
2. Use JPublisher to generate source code — Java and SQLJ files — that represents the Oracle objects, PL/SQL packages, user-defined types, and REF types.
3. Import these classes into your application code.
4. Use the methods in the generated classes to access and manipulate the Oracle Objects and their attributes.

5. Compile all classes (the code generated by Oracle JPublisher and your code). The SQLJ compiler compiles the `.sqlj` files, and the Java or SQLJ compiler compiles the `.java` files.
6. Run your compiled application.

This process is illustrated in the following figure:

Figure 28–3 Oracle Objects and PL/SQL Packages



28.3.1 How to Use JPublisher

Oracle JPublisher increases your productivity by letting you access Oracle objects and PL/SQL packages from your Java programs. Oracle JPublisher lets you specify and customize the mapping of Oracle object types, reference types, and collection types (VARRAYs or nested tables) to Java classes in a strongly typed paradigm

JPublisher

JPublisher increases your productivity by letting you access Oracle objects and PL/SQL packages from your Java programs. JPublisher lets you specify and customize the mapping of Oracle object types, reference types, and collection types (VARRAYs or nested tables) to Java classes in a strongly typed paradigm.

SQLJ programmers who want to call stored procedures declared in PL/SQL packages can use JPublisher to generate SQLJ wrapper classes for the packages. The SQLJ wrapper classes let you invoke the PL/SQL stored procedures, and pass and return values from them, directly from your SQLJ program.

For more information, see the Oracle Database JPublisher User's Guide.

Object Types and JPublisher

JPublisher allows your Java language applications to use user-defined object types in Oracle Database. These objects can be user-defined objects, VARRAYs, nested tables, index-by tables, or REFs to object types. If you intend to have your Java-language application access object data, then it must represent the data in a Java format. JPublisher helps you do this by creating the mapping between object types and Java classes, and between object attribute types and their corresponding Java types.

The mapping is determined by both:

- The selected Java mapping option.
- The object's data type category.

Additionally, JPublisher generates get and set accessor methods for each of the object's attributes, and optionally generates a wrapper method for each of the object's stored procedures. A wrapper method is a method that invokes a stored procedure that executes in the database. Wrapper methods generated by JPublisher are always instance methods even when the original object methods are static.

The following table summarizes the types of Java classes that JPublisher generates for objects.

Table 28–1 Mapping SQL Type to Java Class

SQL type	Java class mapping
user-defined object type	Java class with accessor methods to get and set each attribute of the object, and optional wrapper methods to call the object's stored procedures.
VARRAY, nested table, index-by table.	Java classes that can get and set the following: ■ The entire array ■ A subset of the array ■ An individual element of the array
REF to an object type	Java class to get and set the object to which the REF refers.

Classes generated by JPublisher implement either the `oracle.sql.CustomDatum` interface or the `java.sql.SQLData` interface. Either interface makes it possible to transfer object type instances between the database and your Java program. It is recommended that you use the `oracle.sql.CustomDatum` interface.

PL/SQL Packages and JPublisher

You might want to call stored procedures in a PL/SQL package from your Java application. The stored procedure can be implemented in PL/SQL, or it can be a Java method that has been published to PL/SQL. Java arguments and functions are passed to and returned from the stored procedure.

To help you do this, you can direct JPublisher to create a class containing a wrapper method for each subprogram in the package. Like object methods, the wrapper methods generated for each subprogram are always instance methods even when the original method is static. The wrapper methods that JPublisher generates provide a convenient way to invoke PL/SQL stored procedures from Java code or to invoke a Java stored procedure from a client Java program.

JPublisher lets you generate Java wrappers by selecting an individual package, or by selecting the Packages node to select all of the packages in the schema. If you call PL/SQL code that includes subprograms at the top-level, JPublisher generates a single class containing a wrapper method for each top-level subprogram.

For PL/SQL functions, whether you generate Java for a single PL/SQL function or multiple functions, JPublisher generates a single class. For a single function, the class contains a single wrapper method for the function. For multiple functions, the class contains a wrapper method for each function.

For PL/SQL procedures, whether you generate Java for a single PL/SQL procedure or multiple procedures, JPublisher generates a single class. For a single procedure, the class contains a single wrapper method for the procedure. For multiple procedures, the class contains a wrapper method for each procedure

Java Mapping Options

The mapping options you select for data type categories determine the set of type mappings that JPublisher uses to translate object types and PL/SQL packages into Java classes:

- For object types, JPublisher applies the mappings to the object's attributes and to the arguments and results of any methods included with the object. The mappings control the types that the generated accessor methods should support, that is, what types the get methods should return and the set methods should require.
- For PL/SQL packages, JPublisher applies the mappings to the arguments and results of the methods.
- For a collection type (that is, nested tables and VARRAYs), JPublisher applies the mappings to the element type of the collection.
- For user-defined types (`usertypes` category) JPublisher generates `CustomDatum` classes or SQLData classes and generates code for collection and REF types.

You may select from the following mapping options:

- Oracle Mapping represents data in PL/SQL format.
- JDBC Mapping represents simple data types as Java primitive types.
- Object JDBC Mapping represents simple data types as Java wrapper classes.
- BigDecimal Mapping uses a common class to represent all numeric types.

For more information, see the Oracle Database JPublisher User's Guide.

Mapping Built-in Types

Syntax: **jpub.builtintypes={jdbc|oracle}**

The `builtintypes` parameter (and its JPublisher wizard equivalent Built-in Types) controls type mappings for all the built-in database types except the LOB and BFILE types (controlled by the `lobtypes` parameter) and the different numeric types (controlled by the numbertypes parameter). The following table lists the database types affected by the `builtintypes` parameter, and shows their Java type mappings for `builtintypes=oracle` and for `builtintypes=jdbc` (the default).

Table 28–2 Built In Mapping Types

PL/SQL Data Type	Oracle Mapping Class	JDBC Mapping
CHAR	oracle.sql.CHAR	java.lang.String
CHARACTER		
LONG		
STRING		
VARCHAR		
VARCHAR2		
RAW	oracle.sql.RAW	byte[ ]
LONG RAW		
DATE	oracle.sql.DATE	java.sql.Timestamp

Mapping LOB Types

Syntax: `lobtypes={jdbc|oracle}`

The lobtypes parameter (and its JPublisher wizard equivalent LOB Types) controls type mappings for the LOB types. The following table shows how these types are mapped for `lobtypes=oracle` (the default) and for `lobtypes=jdbc`.

Table 28–3 LOB Type Mapping

PL/SQL Data Type	Oracle Mapping Class	JDBC Mapping Class
CLOB	oracle.sql.CLOB	java.sql.CLOB
BLOB	oracle.sql.BLOB	java.sql.BLOB

The BFILE type does not appear in this table, because it has only one mapping. It is always mapped to `oracle.sql.BFILE`, because there is no `java.sql.BFILE` class.

Mapping Numeric Types

Syntax: `jpub.numbertypes={jdbc|objectjdbc|bigdecimal|oracle}`

The `numbertypes` parameter (and its JPublisher wizard equivalent Number Types) controls type mappings for numeric PL/SQL types. Four choices are available:

- The `jdbc` mapping maps most numeric database types to Java primitive types such as `int` and float, `and` maps `DECIMAL` and `NUMBER` to `java.math.BigDecimal`.
- The `objectjdbc` mapping (the default) maps most numeric database types to Java wrapper classes such as `java.lang.Integer` and `java.lang.Float`, and maps `DECIMAL` and `NUMBER` to `java.math.BigDecimal`.
- The `bigdecimal` mapping maps all numeric database types to `java.math.BigDecimal`. The oracle mapping maps all numeric database types to `oracle.sql.NUMBER`.
- The `oracle` mapping maps all numeric database types to `oracle.sql.NUMBER`.

The following table lists the PL/SQL types affected by the `numbertypes` option, and shows their Java type mappings for `numbertypes=jdbc` and `numbertypes=objectjdbc` (the default).

Table 28–4 Numeric Type Mapping

PL/SQL Data type	JDBC Mapping Class	Object JDBC Mapping
BINARY_INTEGER	int	java.lang.Integer
INT		
INTEGER		
NATURAL		
NATURALN		
PLS_INTEGER		
POSITIVE		
POSITIVEN		
SIGNTYPE		
SMALLINT	short	java.lang.Float
REAL	float	java.lang.Double

Mapping User-Defined Types

Syntax: `jpub.usertypes={oracle|jdbc}`

The `usertypes` parameter (and its JPublisher wizard equivalent `User Types`) controls whether JPublisher generates `CustomDatum` classes or `SQLData` classes for user-defined types:

- When `usertypes=oracle` (the default), JPublisher generates `CustomDatum` classes for `object`, `collection`, and `REF` types.
-

When `usertypes=jdbc`, JPublisher generates SQLData classes for object types. JPublisher does not generate anything for `collection` or `REF` types. Use `java.sql.Array` for all `collection` types, and `java.sql.Ref` for all `REF` types.

28.3.2 JPublisher Output

JPublisher generates a Java class for each object type that it translates. For each object type, JPublisher generates a `type.java` file (or a `type.sqlj` file if wrapper methods were requested) for the class code and a `typeRef.java` file for the code for the REF class of the Java type. For example, if you define an EMPLOYEE PL/SQL object type, JPublisher generates an `employee.java` file and an `employeeRef.java` file.

For each collection type (nested table or VARRAY) it translates, JPublisher generates a `type.java file`. For nested tables, the generated class has methods to get and set the nested table as an entire array and to get and set individual elements of the table. JPublisher translates collection types when generating `CustomDatum` classes but not when generating SQLData classes. JPublisher does not generate a `typeRef.java` file for nested tables or VARRAYs. This is because PL/SQL does not allow a REF to be made to these types.

For PL/SQL packages, JPublisher generates classes containing wrapper methods as SQLJ files. JPublisher also generates method wrappers in your class that invoke the associated package methods executing in the server. This is specified by the **Include Methods** option.

Note: Since version 8.1.6, the wrapper methods that JPublisher generates to invoke stored procedures are in SQLJ only. Classes generated by JPublisher that contain wrapper methods must be compiled by SQLJ.

28.3.3 Properties Files

A properties file is an optional text file where you can specify frequently used parameters or parameters that you cannot specify in the JPublisher wizard. Note that if you need only the default output of JPublisher, then you do not need a properties file.

The properties file is designated in the JPublisher wizard.

In a properties file, you enter one (and only one) parameter and its associated value on each line. Each parameter name must be preceded with the prefix "jpub." and you cannot use any white space within a line. You can enter any parameter except the props parameter in the properties file. JPublisher processes the parameters, in order, from the top of the list to the bottom. If you specify a parameter more than once, JPublisher uses the last encountered value.

A properties file might contain the following:

```
jpub.case=lower
jpub.package=package1
jpub.numbertypes=jdbc
jpub.lobtypes=jdbc
jpub.builtintypes=jdbc
jpub.usertypes=jdbc
jpub.omit_schema_names
jpub.methods=true
jpub.input=mySchema.txt
jpub.sql=employee:oracleEmployee
```

28.3.4 How to Enhance JPublisher-Generated Classes

You can enhance the functionality of a custom Java class generated by JPublisher by adding methods and transient fields to it. For example:

- Extend the class. That is, treat the JPublisher-generated class as a superclass, write a subclass to extend its functionality, and then map the object type to the subclass.
- Write a new class that delegates the functionality provided by the JPublisher-generated class to a field whose type is the generated class.
- Add methods to the class. This is not recommended if you anticipate running JPublisher at some future time to regenerate the class. If you regenerate a class that you have modified in this way, your changes (that is, the methods you have added) will be overwritten. Even if you direct JPublisher output to a separate file, you will still need to merge your changes into the file.

28.3.5 How to Extend JPublisher-Generated Classes

The **Declaration Name** and **Use Name** fields in the JPublisher wizard give you the flexibility of extending generated classes. In the **Declaration Name** field, enter the name of the class that you want JPublisher to generate from the given database object. In the **Use Name** field, enter the name of the class that your Java program will use to represent the database object.

When publishing an object type where Use Name is different from Declaration Name, JPublisher creates a `declaration_name.sqlj` file and a `use_nameRef.java` file, where `use_name` represents the object type in your Java program.

JPublisher expects that you have written the class `use_name`, which extends `declaration_name`. If you do not provide this class, then the `use_nameRef.java` file will not compile.

For example, suppose you want JPublisher to generate the class `JAddress` from the PL/SQL object type `ADDRESS`. You have also written a class, `MyAddress`, to represent `ADDRESS` objects, where `MyAddress` either extends the functionality provided by `JAddress` or has a `JAddress` field.

Under this scenario, select `ADDRESS` in the Database Browser and right-click **Generate Java**. In the JPublisher wizard, enter `JAddress` in the **Declaration Name** field and `MyAddress` in the **Use Name** field. JPublisher will generate the custom Java class `JAddress`, and map the `ADDRESS` object to the `MyAddress` class—not to the JAddress class. JPublisher will also produce a reference class for `MyAddress`, not `JAddress`.

This is how JPublisher will alter the code it generates:

- JPublisher generates the REF class `MyAddressRef` rather than `JAddressRef`.
- JPublisher uses the `MyAddress` class, instead of the `JAddress` class, to represent attributes whose database type is `ADDRESS`. This situation occurs in classes generated by JPublisher, or in classes written by the user.
- JPublisher uses the `MyAddress` class, instead of the `JAddress` class to represent `VARRAY` and nested table elements whose database type is `ADDRESS`.
- JPublisher will use the `MyAddress` factory, instead of the `JAddress` factory, when the `CustomDatumFactory` interface is used to construct Java objects whose database type is `ADDRESS`. This situation will occur both in classes generated by JPublisher, and in classes written by the user.

The class that you create (for example, `MyAddress.java`) must have the following features:

- The class must have a no-argument constructor. The easiest way to construct a properly initialized object is to invoke the constructor of the superclass, either explicitly or implicitly.
- The class must implement the `CustomDatum` interface. The simplest way to do this is to inherit the `toDatum()` method from the superclass.
- You must also implement the `CustomDatumFactory` interface, either in the same class or in a different one. For example, you could have a class `Employee` that implements `CustomDatum` and a class `EmployeeFactory` that implements `CustomDatumFactory`.

28.3.6 JPublisher Options

JPublisher options can be set for these types of PL/SQL subprograms in your schema.

How to Set JPublisher Options

JPublisher options can be set for these types of PL/SQL subprograms in your schema:

- Functions
- Package Bodies
- Packages

- Procedures

For more information, see the *Oracle Database JPublisher User's Guide*.

To set JPublisher options for PL/SQL subprograms in a schema:

1. In the Connection Manager, navigate a schema to find and select the node for the subprogram type
2. Right-click and choose **Generate Java** to launch the JPublisher wizard. For more help at any time, press F1 or click **Help** in the wizard.

How to Generate Classes for Packages and Wrapper Methods for Methods

Set the JPublisher methods option in the JPublisher wizard by checking **Include Methods**

The value of the methods option determines whether JPublisher generates classes for PL/SQL packages and wrapper methods for methods in packages and object types.

If selected, JPublisher generates PL/SQL classes and methods. This is default behavior.

If not selected, JPublisher does not generate PL/SQL classes and methods.

How to Omit the Schema Name from Generated Names

Set the JPublisher `omit_schema_names` option in the JPublisher wizard by checking the **Omit Schema Names** box.

The value of the `omit_schema_names` option determines whether certain object type and PL/SQL wrapper names generated by JPublisher include the schema name. If an object type or wrapper name generated by JPublisher does not include the schema name, the type or wrapper is looked up in the schema associated with the current connection when the code generated by JPublisher is executed. This makes it possible for you to use classes generated by JPublisher with a connection other than the one used when JPublisher was invoked. However, the type or package must be declared identically in the two schemas.

If selected, an object type or wrapper name generated by JPublisher is qualified with a schema name only if either:

- You declare the object type or wrapper in a schema other than the one to which JPublisher is connected; or
- You declare the object type or wrapper with a schema name in the properties file or INPUT file.

That is, an object type or wrapper from another schema requires a schema name to identify it, and the use of a schema name with the type or package in the properties file or INPUT file overrides the `omit_schema_names` option.

If not selected, every object type or wrapper name generated by JPublisher is qualified with a schema name. This is default behavior.

How to Set the Package Name for Generated Classes

The package option specifies the name of the Java package JPublisher generates. The name of the package appears in a package declaration in each Java file. The directory structure also reflects the package name. An explicit name in the `INPUT` file, after the sql option, overrides the value given to the package option.

To set the package option:

1. Set the JPublisher package option in the JPublisher wizard by providing a name in the **Package** field.

28.4 Using Java Stored Procedures

A Java stored procedure is a Java method that resides and runs in a database. Stored procedures can help improve the performance of database applications because they are efficient: they are stored in the RDBMS in executable form, and run in the RDBMS (rather than the client) memory space.

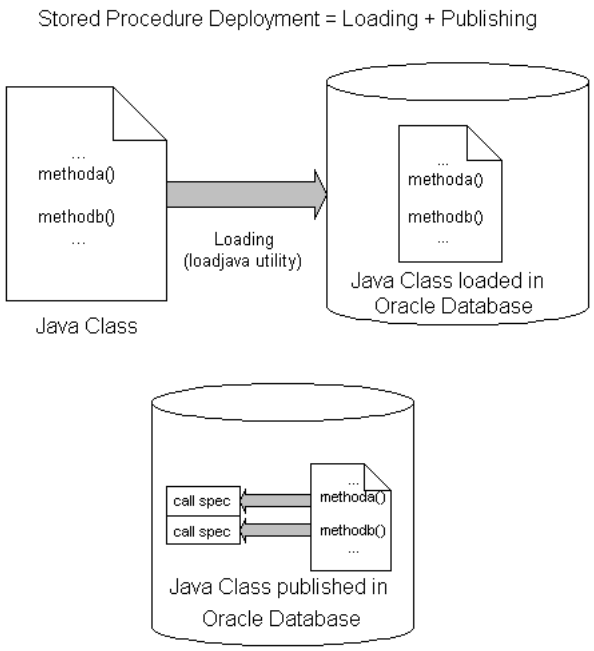
Use JDeveloper to write methods in Java for new stored procedures and deploy them to Oracle Database. When you deploy a Java class to Oracle, you can select the methods that you want to publish to PL/SQL for use as stored procedures. Methods can be deployed together in a package or separately.

For more information, see "Developing Java Stored Procedures" in the *Oracle Database JPublisher User's Guide*.

A stored procedure is a program that resides and runs in a database. Application developers can use stored procedures to help improve the performance of a database application. Procedure calls are quick and efficient because a stored procedure is compiled once and stored in an executable form. Because a stored procedure runs in the RDBMS memory space, complex functions run faster than a routine run by a client. You can also use stored procedures to group PL/SQL statements so that they are executed in a single call. This reduces network traffic and improves round-trip response times. By designing applications around a common set of stored procedures, you can avoid redundant coding and increase your productivity.

A Java stored procedure contains Java public static methods that are published to PL/SQL and stored in Oracle Database for general use. To publish Java methods, you write call specifications, that map Java method names, parameter types, and return types to their PL/SQL counterparts. This allows a Java stored procedure to be executed from an application as if it were a PL/SQL stored procedure. When called by client applications, a Java stored procedure can accept arguments, reference Java classes, and return Java result values.

Figure 28–4 Java Stored Procedure Deployment



Any Java class can be deployed to Oracle Database and the conforming methods of the class can be published to PL/SQL as stored procedures. These Java stored procedures can then be executed from an application as if they were PL/SQL stored procedures. Java stored procedures can be an entry point for your application into other (Java and non-Java) procedures deployed to Oracle Database.

Deploying and publishing Java stored procedures to Oracle Database generates call specifications that act as PL/SQL wrappers for each of the methods selected for publishing. The PL/SQL wrappers allow the stored procedures to be accessible from SQL*Plus, JDBC, or any other Oracle application environment.

The call specifications (the PL/SQL wrappers) for Java stored procedure packages and methods deployed to a database schema can be inspected through Oracle Database connection. Only published Java stored procedures appear as PL/SQL blocks, and only public static methods in Java classes can be published to PL/SQL when deployed. Java classes can be deployed without being published, in which case they are not seen in the PL/SQL nodes.

Depending on how Java stored procedures were published, they appear in one of the following nodes under a schema:

- Packages include call specs for Java stored procedures deployed in packages.
- Functions include call specs for Java stored procedures deployed as functions (that return a value).
- Procedures include call specs for Java stored procedures deployed as procedures (that do not return a value).

To view a Java stored procedure's call specification, find its node in the schema's hierarchy, and double-click it.

How to Create Java Stored Procedures

You create Java stored procedures by first developing business application logic in a Java class file. Declare methods that are to become stored procedures as public static.

Use the editor in JDeveloper to add and edit business logic in the Java class. During deployment to Oracle Database, all public static methods included in the class file are available to be published to PL/SQL as stored procedures. You can choose which public static methods in the class to be published to PL/SQL.

There are different JDeveloper Java stored procedure creation scenarios:

- Use an existing Java class and make any necessary edits to the public static methods in the class that will be deployed to Oracle Database. The existing class could include public static methods used for validation or database triggers. The methods in the class might also be in local use by several applications. These methods could be deployed to Oracle Database and used by multiple applications from the database. The deployed methods could also supplement existing PL/SQL stored procedures and functions.
- Create a new class with methods designed for publishing as stored procedures. Use the editor in JDeveloper to create the public static methods that will be exposed as stored procedures. Write in industry-standard Java and use the original Java class for other application deployments. In Oracle Database, this programming could supplement existing PL/SQL stored procedures.

For example, assume the following Java package `Welcome` was created with the public class `Greeting` and the public static method `Hello()`.

```
package Welcome;
 public class Greeting {
 public static String Hello() {
 return "Hello World!";
 }
```

When this package is deployed to Oracle Database and the `Hello()` method is published there, the call spec for the package as viewed in the source editor looks like this:

```
PACKAGE WELCOME AS
FUNCTION HELLO RETURN VARCHAR2
AS LANGUAGE JAVA
NAME 'Welcome.Greeting.Hello() return java.lang.String'
END WELCOME;
```

How to Deploy Java Stored Procedures

You create a deployment profile for Java stored procedures, then deploy the classes and, optionally, any public static methods in JDeveloper using the settings in the profile.

Deploying to the database uses the information provided in the Deployment Profile wizard and two Oracle Database utilities:

- `loadjava` loads the Java class containing the stored procedures to Oracle Database.
- `publish` generates the PL/SQL call spec wrappers for the loaded public static methods. Publishing enables the Java methods to be called as PL/SQL functions or procedures.

To deploy Java stored procedures in JDeveloper:

1. If necessary, create a database connection in JDeveloper.
2. If necessary, create a deployment profile for Loadjava and Java stored procedures.

3. Deploy the objects.

How to Create a Deployment Profile for Loadjava and Java Stored Procedures

The Loadjava and Java stored procedure deployment profile is very similar to the simple archive profile, except that the selected contents of the profile will be uploaded into Oracle Database via the command-line tool loadjava or in the case of Java stored procedures, they are stored in Oracle Database for general use.

> **Note:** Make sure that you have configured a database connection in JDeveloper before you complete this task.

To create a deployment profile for Loadjava or Java stored procedures in JDeveloper:

1. In the Application Navigator, select the project in which you want to create the deployment profile.
2. Choose **File > New** to open the New Gallery.
3. In the **Categories** tree, expand **Database Tier** and select **Database Files**. In the **Items** list, double-click **Loadjava and Java Stored Procedures**.

   If the category or item is not found, make sure the correct project is selected, and select `All Technologies` in the **Filter By** dropdown list.
4. In the Create Deployment Profile dialog, specify a location for the deployment profile or accept the defaults. The deployment profile is named with a `.deploy` filename extension.
5. Click **Save** to display the Loadjava and Java Stored Procedures Deployment Profile Settings dialog. Configure the settings for each page as appropriate, and click `OK` when you are done.

   The newly created `storedProc.deploy` deployment profile appears in the navigator below the specified project.
6. Select and right-click `storedProc.deploy` in the Navigator. Choose from the available context menu options.
7. (Optional) If you choose **Add Stored Procedure Package**, choose the methods you want to load as a stored procedure. For each Java method callable from SQL a call spec is required, which exposes the method's top-level entry point to the database. Typically, only a few call specs are needed. JDeveloper generates the call spec for you from this page.
8. Select a method and click **Settings**.

   If a method on the list is dimmed, this indicates a problem with deploying this method as a Java stored procedure. Click **Why not?** for an explanation.

   For more information, see "Developing Java Stored Procedures" in the *Oracle Database JPublisher User's Guide*.
9. Configure the Method Settings as required. These settings allow you to customize the parts of the `CREATE PROCEDURE` and `CREATE FUNCTION SQL` statements that are used to customize the stored procedure.
10. (Optional) Right-click and choose **Preview SQL Statements** to display a dialog that shows the SQL statements used to load the specifically selected item in the Navigator. In the case of top-level procedures or functions and packages, you will see complete SQL statements. In the case of packaged procedures or functions, you

will only see fragments of SQL statements which represent the portion of the `CREATE PACKAGE BODY` statement corresponding to the packaged procedure or function.

- (Optional) If you choose **Add PL/SQL Package**, enter the name of a PL/SQL package that you want to start building.
- (Optional) Right-click and choose **Preview SQL Statements** to display a dialog that shows the SQL statements used to load the specifically selected item in the Navigator. In the case of top-level procedures or functions and packages, you will see complete SQL statements. In the case of packaged procedures or functions, you will only see fragments of SQL statements which represent the portion of the `CREATE PACKAGE BODY` statement corresponding to the packaged procedure or function.
- To deploy the profile, see Deploying Loadjava and Java Stored Procedures Profile.

How to Deploy to Oracle Databases

If necessary:

- Create a database connection in JDeveloper.
- Create a deployment profile for Loadjava and Java stored procedures.

> **Note:** If you are deploying to Oracle9*i* Database release 2 (9.2) or later, set the compiler's target to 1.1 or 1.2. in the Project Properties dialog, available from the **Application** menu.

To deploy Loadjava and Java stored procedures in JDeveloper:

1. Right-click `storedProc.deploy` which appears in the Navigator below the specified project.
2. From the context menu, choose `Deploy to` and select one of the already existing database connections; the Java application's source files are uploaded directly into the selected database.

   Or, choose `New Connection` to display the Create Database Connection Wizard.
3. (Optional) If you want to edit the deployment profile, right-click `storedProc.deploy` in the Navigator below the specified project and choose **Settings**.

> **Note:** If you are deploying your files as both compiled files and source files and you have selected either `-resolve` or `-andresolve` in the Resolver page, then the deployment profile will only upload the source files. The reason is that when loadjava resolves the uploaded .java source files, loadjava also compiles the .java source files into .class files. You will only see the source files when previewing the loadjava deployment profile settings.

How to Invoke Java Stored Procedures

The SQL CALL statement lets you call Java stored procedures.

To invoke a Java Stored Procedure using SQL:

1. In SQL*Plus, execute the CALL statement interactively to invoke a Java stored procedure, using the syntax:

```
CALL [schema_name.][{package_name | object_type_name}][@dblink_name]
 { procedure_name ([param[, param]...])
 | function_name ([param[, param]...]) INTO :host_variable};
where param represents this syntax:
{literal | :host_variable}
```

   Host variables, that is variables declared in a host environment, must be prefixed with a colon. The following examples show that a host variable cannot appear twice in the same CALL statement, and that a subprogram with no parameters must be called with an empty parameter list:

```
CALL swap(:x, :x); -- illegal, duplicate host variables

CALL balance() INTO :current_balance; -- () required
```

To invoke a Java stored procedure using JDBC:

1. Java stored procedures invoked from JDBC must be encapsulated in CallableStatement objects.

   Create a callable statement object:

   - Declare a callable statement object. For example:

     ```
     private CallableStatement checkIn;
     ```

   - Initialize the callable statement object by calling `prepareCall` on the connection with a SQL CALL statement for the stored procedure. For example:

     ```
     checkIn = connection.prepareCall(quot;{call NPL.CHECKIN(?, ?, ?)}");
     ```

     Note: The number of parameters in the stored procedure is represented by the number of place-holders in the SQL call.

2. Register the callable statement object's output parameters. Call `registerOutParameter` for each output parameter, identifying it by position, and declaring its type. For example, if the second parameter is an `SQL INTEGER` (which maps to a Java `int`), and the third is a `SQL VARCHAR` (which maps to a Java `String`), then:

```
newCustomer.registerOutParameter(2, Types.INTEGER);
newCustomer.registerOutParameter(3, Types.VARCHAR);
```

3. Execute the callable statement object:

   - Provide the callable statement object's input parameters by calling a set method, identifying the parameter by position, and assigning it a value. For example, if the first parameter is an int input parameter:

     ```
     checkIn.setInt(1, bookID);
     ```

   - Execute the callable statement object. For example:

     ```
     checkIn.execute();
     ```

- Extract the callable statement object's output parameters. Call a get method for each output parameter, identifying the parameter by position. The get methods return values of corresponding Java types. For example:

```
int daysLate = checkIn.getInt(2);
String title = checkIn.getString(3);
```

To invoke a Java stored procedure using SQLJ:

1. Declare and initialize input and in-out variables. For example, if the first parameter is an int input parameter:

```
int bookID = scanID();
```

2. Declare output variables. For example:

```
int daysLate; String title;
```

3. Invoke the stored procedure in a SQLJ statement. In the statement identify the parameters by name, and designate them as :in, :out, or :inout. For example:

```
#sql { call NPL.CHECKIN (:in bookID, :out daysLate, :out title)}
```

   Return values will be assigned to output and input variables.

To Invoke a Java Stored Procedure using PL/SQL

1. Use a CALL statement in the trigger body clause of a PL/SQL statement to invoke a stored procedure, and pass arguments to it.

   The CALL statement's arguments can be:

   - Literal values.
   - SQL expressions, but not bind variables.
   - Column references, qualified by correlation names.

   Correlation names are prefixes to column references. Use these names to qualify whether the reference applies to the existing column value of the row being processed by the trigger or the value being written by the triggering event:

   - OLD refers to the value of the column prior to the triggering operation.
   - NEW refers to the value being assigned to the column by the triggering operation. It is possible for the trigger body to redefine this value before the triggering operation occurs.

   An example of a complete trigger definition:

```
CREATE TRIGGER check_salary
BEFORE UPDATE OF salary ON employee
CALL salaryCheck(:new.job, :old.salary, :new.salary, :old.employee
```

   CREATE TRIGGER check_salary

   BEFORE UPDATE OF salary ON employee

   CALL salaryCheck(:new.job, :old.salary, :new.salary, :old.employeeID)

How to Test Java Stored Procedures

For stored procedures deployed in packages, access the stored procedure by the package name and/or the stored procedure name set during deployment. The package name may be the default name taken from the project or another name entered during

deployment. The stored procedure name may be the default name taken from the method name or a name chosen for the stored procedure during deployment. Stored Procedures may also be deployed without packages.

For example, assume a public static method `hello()` is in the Java package `Welcome` and the public class `Greeting`. Further assume it is deployed in a package `Openings`.

You could execute a PL/SQL query to the deployed stored procedure that executes the public static method deployed there and returns the result. To invoke SQL*Plus from within JDeveloper, right-click a connection or select it from the Tools menu.

With a working connection to the database, your SQL*Plus client could execute the following:

```
package Welcome;
 public class Greeting {
 public static String Hello() {
 return "Hello World!";
 }
 }
```

You could execute a PL/SQL query to the deployed stored procedure that executes the public static method deployed there and returns the result. To invoke SQL*Plus from within JDeveloper, right-click a connection or select it from the Tools menu.

With a working connection to the database, your SQL*Plus client could execute the following:

```
select Openings.Hello() from dual;
Openings.Hello()
```

Executing the code displays:

```
Hello World!
```

> **Note:** The reference to the stored procedure call spec uses package.method syntax; the name of the class from which the method originated is not part of the call.

For stored procedures deployed separately (not in packages), access the stored procedure by the stored procedure name set during deployment. The stored procedure name may be the default name taken from the method name or a name chosen for the stored procedure during deployment.

For example, for a public static method `hello()` that was deployed as `hello` from a class `greeting` and package `welcome`, you could execute a PL/SQL query to the deployed stored procedure that returns the result.

Assume the above `hello()` method as the example method, but this time assume it was deployed without a package.

With a working connection to the database, your SQL*Plus client could execute the following:

```
select Openings.Hello() from dual;

Openings.Hello()
```

The executed code displays:

```
Hello World!
```

28.4.1 How to Debug Java Stored Procedures

Debug Java stored procedures through a database connection.

To debug PL/SQL:

1. Choose **View > Database > Database Navigator**.
2. Expand **IDE Connections** or **application**, and select a database connection.
3. Expand a schema, and find a node with the name of the object type (for example, Package), and expand the node.
4. In the node, right-click the PL/SQL program, and choose **Debug**.
5. A Debug PL/SQL window opens. Select a target and parameter(s), and click **OK**.
6. JDeveloper debugs the program. Check status windows for progress and information.

Additional information is available in Debugging PL/SQL Programs and Java Stored Procedures.

28.4.2 How to Remove Java Stored Procedures

To drop a stored procedure:

1. Choose View **> Database > Database Navigator**.
2. Expand IDE Connections or application, and select a database connection.
3. Expand the connection and select a schema.
4. Expand the schema and locate the object you wish to remove. Depending on how Java stored procedures were published, they appear in one of these nodes:
   - Packages includes call specs for Java stored procedures deployed in packages.
   - Functions includes call specs for Java stored procedures deployed as functions (that return a value).
   - Procedures includes call specs for Java stored procedures deployed as procedures (that do not return a value).
5. Select the object and right-click to display the context menu and choose **Drop**.

29

Running and Debugging PL/SQL and Java Stored Procedures

You can use JDeveloper to write methods in Java for stored procedures and deploy them to the database.

This chapter includes the following sections:

29.1 About Running and Debugging PL/SQL and Java Stored Procedures

A Java stored procedure is a Java method that resides and runs in a database. Stored procedures can help improve the performance of database applications because they are efficient: they are stored in the RDBMS in executable form, and run in the RDBMS (rather than the client) memory space.

When you deploy a Java class to the database, you can select the methods that you want to publish to PL/SQL for use as stored procedures. Methods can be deployed together in a package or separately.

29.2 Running and Debugging Functions, Procedures, and Packages

JDeveloper lets you run and debug PL/SQL program units. For example, you can specify parameters being passed or return values from a function giving you more control over what is run and providing you output details about what was tested.

> **Note:** The procedures or functions in Oracle Database can be either standalone or within a package.

To run or debug functions, procedures, and packages:

1. Choose **View > Database > Database Navigator**.
2. Expand **IDE Connections** or **application**, and select a database connection.
3. Expand a schema and expand the appropriate node depending on what you are debugging (Procedure, Function, or Package body):

- Optional for debugging only) Right-click and choose **Compile for Debug** from the context menu of the node for the object that you are debugging. This compiles the PL/SQL program in `INTERPRETED` mode.
- (Optional for debugging only) Select the function, procedure, or package that you want to debug and double-click to open it in the editor.
- (Optional for debugging only) Set a breakpoint in your PL/SQL code by clicking to the left of the margin.

> **Note:** The breakpoint must be set on an executable line of code. If the debugger does not stop, the breakpoint may have not been set on an executable line of code (verify that the breakpoint was verified).

4. Make sure that either the editor or the procedure in the navigator is currently selected.
5. Click **Debug**, or if you want to run without debugging, click **Run**.
6. The Run PL/SQL dialog is displayed.
    1. Select a **Target** which is the name of the procedure or function that you want to debug. Notice that the content in the **Parameters** and **PL/SQL Block** boxes change dynamically when the target changes.

    > **Note:** You will have a choice of target only if you choose to run or debug a package that contains more than one program unit

    2. The **Parameters** box lists the target's arguments (if applicable).
    3. The **PL/SQL Block** box displays code that was custom generated by JDeveloper for the selected target. Depending on what the function or procedure does, you may need to replace the NULL values with reasonable values so that these are passed into the procedure, function, or package. In some cases, you may need to write additional code to initialize values to be passed as arguments. In this case, you can edit the PL/SQL block text as necessary.
7. Click **OK** to execute or debug the target.
8. Analyze the output information displayed in the Log window. In the case of functions, the return value will be displayed. `DBMS_OUTPUT` messages will also be displayed.

29.3 Debugging PL/SQL Programs and Java Stored Procedures

In addition to debugging Java programs, the JDeveloper debugger enables you to debug PL/SQL programs and Java stored procedures in Oracle Databases.

29.3.1 Debugging PL/SQL Objects

JDeveloper supports both PL/SQL and Java stored procedures debugging in a single IDE tool. When debugging PL/SQL, the source code you are debugging must be stored in Oracle Database. For Java stored procedures, the source code should be in your JDeveloper project and the compiled code should be deployed in the database.

Also, the way the debug action is initiated is different depending on whether you are performing local or remote debugging. When debugging PL/SQL, this distinction is described as follows:

- Local debugging - JDeveloper automatically launches the program you want to debug, also referred to as the debuggee process, and then attaches the debugger to that program.
- Remote debugging - You must manually launch the program you want to debug with an Oracle client such as SQL*Plus, Dbms_Job, an OCI program, or a trigger firing. You must then establish the connection from the database debuggee process to the JDeveloper debugger. After the debuggee is launched and the JDeveloper debugger is attached to it, remote debugging is very similar to local debugging.

PL/SQL and Java stored procedure debugging information is displayed in the various JDeveloper debugger windows including the Smart Data, Data, Watches, Inspector, Stack, and Classes windows.

The Threads window, Heap window, and Monitors window are not applicable when debugging PL/SQL code.

When debugging PL/SQL, the user can use PL/SQL expressions in the Watches and Inspector windows as well as conditional breakpoints, including table element access; for example, mytable(i*10). This capability includes tables which are declared in functions, procedures, packages, and package bodies.

29.3.1.1 PL/SQL objects you can debug with JDeveloper

You can debug a PL/SQL program calling PL/SQL, PL/SQL calling a Java stored procedure (Oracle9i Release 2 and later databases), and a PL/SQL program issuing a SQL statement that fires a trigger.

You can initiate debugging PL/SQL from the following objects:

- Stand-alone procedures
- Stand-alone functions
- Packaged procedures
- Packaged functions

Any other PL/SQL object can be traced into as long as it meets the prerequisites, and as long as it is invoked from one of the above. For more information, see Section 29.3.3, "Debugging PL/SQL and Java Stored Procedures Prerequisites."

29.3.1.2 What You May Need to Know

Consider the following when debugging triggers, Java stored procedures, and Oracle object types:

- Although you cannot initiate debugging for these objects, you can step into them. For example, you cannot start debugging a trigger, but you can debug a procedure that adds records. To debug a trigger, set a breakpoint in the trigger, then debug the procedure that causes the trigger to fire. The debugger will stop at that breakpoint.
- Debugging and stepping into Java stored procedures is supported in the Oracle9i Release 2 and later databases. These procedures should be included in the JDeveloper project and the source should be consistent with what is deployed in Oracle Database. To debug a Java stored procedure, set a breakpoint in the Java stored procedure, then debug the PL/SQL that calls the Java stored procedure. Alternatively, you can debug the PL/SQL and step into the Java stored procedure.

29.3.1.3 Appearance of debug information in supported Oracle Database

The debugger uses the database's JPDA (Java Platform Debugger Architecture) implementation. JPDA is the industry standard for Java debugging and the JPDA implementation in the database allows you to seamlessly debug Java and PL/SQL.

What You May Need to Know

- If you want to configure the debugging behavior (for remote debugging or for setting the Classes Include and Exclude lists), you must have an active application and project to access the project's debugger settings in the Application > Project Properties - Run/Debug/Profile page.
- The following command is used to connect the debuggee session to the debugger:

  ```
  DBMS_DEBUG_JDWP.CONNECT_TCP( <host_name>, <port> )
  ```

  For local debugging, JDeveloper issues this command for you. For remote debugging, you will need to issue this command in the same session that you use to call the PL/SQL you want to debug.
- When entering an expression in the Watches window, local variables can be entered in any case; for example, `v_value` or `V_Value`. Package variables are also case-insensitive, but the prefix leading up to the variable name is case-sensitive; for example:

  ```
  $Oracle.Package.SCOTT.MY_PACKAGE.g_var
  ```

  The simplest way to add a package variable to the Watches window is to drag and drop the variable from the Data Window or to drag and drop the package from the Classes Window.

29.3.2 How to Specify the Database Debugger Port

When the database debugger is running, for example to debug PL/SQL through a database connection, the ports used are randomly assigned. This can cause problems with firewalls, and to avoid them you can edit the ide.properties file to ensure that a specific port is used.

To specify the port:

1. If necessary, close JDeveloper.
2. In a text editor, open `jdev_install/jdeveloper/jdev/system/oracle.jdeveloper.release_number/ide.properties`.
3. Type the following:

   ```
   DatabaseDebuggerPortOverride=port_number
   ```

   where `port_number` is the port number you want the debugger to use.
4. Save `ide.properties`. When you restart JDeveloper, the port you specified will be used.

29.3.3 Debugging PL/SQL and Java Stored Procedures Prerequisites

You can debug PL/SQL and Java stored procedures in JDeveloper.

Refer to the appropriate section below for additional information.

29.3.3.1 Prerequisites for Debugging PL/SQL and Java Stored Procedures

Ensure that the following prerequisites have been met before performing PL/SQL debugging:

- Your database user account must have these privileges:

  ```
  DEBUG ANY PROCEDURE

  DEBUG CONNECT SESSION
  ```

- The PL/SQL code must be compiled in `INTERPRETED` mode. You cannot debug PL/SQL code that is compiled in `NATIVE` mode. You set this mode in the database's `init.ora` file. See Oracle Database documentation for more information about this file.
- If you do not have an active application and project, the debugger will use the properties defined in the Default Project Properties dialog, available from the Application menu. However, it is recommended that you create a application and a project that you will use when you debug PL/SQL. In the Launch Settings page of the Edit Run Configuration dialog (Edit button on the Run/Debug/Profile page of the Project Properties dialog, which is available from the Application menu), you should ensure that the **Attempt to Run Active File Before Default** check box is selected (default setting). This will instruct the debugger to run the active file (for example a PL/SQL procedure selected in the navigator or open the active file in the editor) when you start debugging.
- PL/SQL objects must be compiled with the `DEBUG` option enabled. Choose one of these techniques to accomplish this task:
  - Ensure that **Generate PL/SQL Debug Information** is selected in Database Connections page of the Preferences dialog (available from the Tools menu), then create or recompile the objects you want to debug.
  - In SQL*Plus, execute `ALTER SESSION SET PLSQL_DEBUG = true`, then create or recompile the object you want to debug.
  - In SQL*Plus, execute `ALTER <procedure, function, package> <name> COMPILE DEBUG;`

29.3.3.2 Prerequisites for Debugging Java Stored Procedures

Ensure that the following prerequisites have been met before performing Java stored procedures debugging:

- The Java code must be deployed to the database and compiled with debug information. From JDeveloper, make sure the Include Debug Information check box is selected in the Compiler page of the Project Properties dialog (available from the Application menu), then deploy the Java stored procedure.
- To step through a Java stored procedure, the Java source must be available in your JDeveloper project and must be consistent with what is deployed to the database.

29.3.4 How to Locally Debug PL/SQL Programs

When locally debugging PL/SQL programs, the call to initiate debugging is made directly from within JDeveloper. JDeveloper automatically launches the program you want to debug, also referred to as the debuggee process, and then attaches the debugger to that program.

Make sure that you've completed the prerequisites listed above.

To locally debug a PL/SQL program in JDeveloper:

1. Choose View > **Database > Database Navigator**.
2. Expand IDE Connections or application, and select a database connection.
3. Expand a schema and expand the appropriate node depending on what you are debugging: Procedure, Function, or Package Body.
4. Select the procedure, function, or package that you want to debug and double-click to open it in the editor.
5. Set a breakpoint in the PL/SQL code by left-clicking in the margin.

> **Note:** The breakpoint must be set on an executable line of code. If the debugger does not stop, the breakpoint may have not been set on an executable line of code (check that the breakpoint was verified). Also, verify that the debugging PL/SQL prerequisites were met. In particular, make sure that the PL/SQL program is compiled in INTERPRETED mode.

6. Make sure that the PL/SQL program unit you want to debug is currently selected in the Navigator.
7. Click the **Debug** toolbar button.
8. JDeveloper halts the execution at the first breakpoint (providing that this was set in the Start Debugging Option in the Project Properties dialog) and displays the state in the debugger windows.
9. Look at the debug information displayed in the JDeveloper debugger windows. For more information, see Section 19.6, "About the Debugger."
10. Resume debugging the PL/SQL program until you are satisfied.

29.3.5 How to Remotely Debug PL/SQL Programs

The main difference between remote debugging and local debugging PL/SQL programs is how you start the debugging session. For remote debugging, you must manually launch the program you want to debug with an Oracle client such as SQL*Plus, Dbms_Job, an OCI program, or a trigger firing. You must then establish the connection from the database program you want to debug (debuggee) to the JDeveloper debugger. After the debuggee is launched and the JDeveloper debugger is attached to it, remote debugging is very similar to local debugging.

You can use the debugger with PL/SQL programs and Java stored procedures in Oracle Database.

Make sure that you've completed the documented prerequisites, listed in Section 29.3.3, "Debugging PL/SQL and Java Stored Procedures Prerequisites."

To remotely debug a PL/SQL program using JDeveloper:

1. If you don't already have one, create a database connection.
2. If you don't already have one, create a project.
3. In the Application Navigator, right-click the project and choose Project Properties.
4. Choose **Run/Debug/Profile.**
5. Either select an existing run configuration or create a new one, and click Edit.

6. In the Edit Run Configuration dialog, select **PL/SQL** and choose the database connection.
7. Select **Tool Settings - Debugger - Remote** and set the remote debugging preferences.
8. In the Database Navigator, right click the connection and chose **Remote Debug**.
9. In the Database Navigator, expand the Database node and navigate to the procedure, function, or package that you want to debug and double-click to open it in the source editor.
10. In the source editor, set a breakpoint in your PL/SQL code by left-clicking in the margin.
11. In the Application Navigator, right-click the project and choose **Debug**.
12. In the displayed dialog, enter the appropriate listening port number and click OK. You can choose any valid port number that is not in use by another process. In this example, the port number used is 4000.

    Note: If you want to bypass this dialog the next time you are debugging on this port, select the Save Parameters check box from this dialog.

    In the Run Manager window, you should see which indicates that the debugger is listening for debugging connections.

13. Use an Oracle client such as SQL*Plus to issue the debugger connection command. Whatever client you use, make sure that the session which issues the debugger connection commands is the same session which executes your PL/SQL program containing the breakpoints.

    For example, if you are using SQL*Plus, issue the following commands to open a TCP/IP connection to the designated machine and port for the JDWP session:

    ```
    EXEC DBMS_DEBUG_JDWP.CONNECT_TCP( '123.456.789.012', '4000' )
    ```

    where `123.456.789.012`is the IP address or host name where JDeveloper is running, and `4000` is the port number on which the debugger is listening.

    From this point on, when you make a call to the PL/SQL code containing the breakpoint, the JDeveloper debugger is activated.

14. When the debugger accepts a debugging connection, the new debugging process is reflected in the Processes folder in the Run Manager. Also, the Log window should display a message similar to the following:

    ```
    Debugger accepted connection from remote process on port 4000.
    ```

    In addition, notice that the layout in JDeveloper has switched from Design layout to Debugging layout (bottom-right of window). Also, the debugging windows including Stack, Data, and Watches, should now be visible.

    In the Run Manager, an icon indicates that the port is continuing to listen and can accept multiple debugging connections.

15. Back in the Oracle client, issue a command which invokes the PL/SQL program unit containing your breakpoint. For example, in SQL*Plus, issue a command similar to the following:

    ```
    EXEC FOO;
    ```

where `FOO` is the name of a PL/SQL procedure.

16. JDeveloper halts the execution at the first breakpoint (providing this was set in the Start Debugging Option in the Project Properties dialog, available from the Application menu) and displays the state in the debugger windows. For more information, see Section 19.6.5, "How to Set the Debugger Start Options."
17. Step into and resume debugging the PL/SQL procedure until you are satisfied. For more information, see Section 19.6, "About the Debugger."
18. When you are finished debugging, disconnect the debuggee using the disconnect command. For example, from SQL*Plus, enter:

    ```
    EXEC DBMS_DEBUG_JDWP.DISCONNECT;
    ```

    The following message appears:

    ```
    Debugger disconnected from remote process.
    ```

19. To terminate the listening port, right-click the Run icon in the Run Manager and choose **Stop Listening**.

29.3.6 Using Acceptable Legal PL/SQL Expressions in the Debugger

If you are debugging PL/SQL, then you can use PL/SQL expressions in the Watches window, Inspector window, Breakpoint conditions, and Breakpoint Log expressions.

The following table lists examples of acceptable legal PL/SQL expressions that you can use in the debugger.

Table 29–1 PL/SQL Expressions that can be used in the debugger

PL/SQL Expression	Example
Simple variable name	`counter`
Field Access	`myrecord.Dept_No`
Table element	`mytable(3)`

Comparison operation	`myrecord.Dept_No = 100`
	`mytable(3) > 7`
	`counter IS NULL`
	`counter IS NOT NULL`
	`employee.salary BETWEEN 25000 AND 50000`

Arithmetic operation	`counter * size` `x + y + z`
Logical operation	`employee.exempt AND employee.active` `employee.exempt OR employee.active`
Package variable name	`$Oracle.Package.HR.MyPackage.MyVariable`
Fully-qualified Package name	`$Oracle.Package.HR`
PackageBody variable name	`$Oracle.PackageBody.HR.MyPackage.MyVariable`
Fully-qualified PackageBody name	`$Oracle.PackageBody.HR`